

Divergence Test

Copyright and license notice

The notebook computes the divergence and curl of the poloidal ionospheric field model that is presented in the journal article "Polar ionospheric currents and high temporal resolution geomagnetic field models" (DOI: 10.1093/gji/ggad325).

Copyright © 2023 Technical University of Denmark

This version of the software was developed by Clemens Kloss, Postdoc, at the Division of Geomagnetism and Geospace, DTU Space, Technical University of Denmark.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Overview

This notebook investigates the divergence and curl of our estimated poloidal ionospheric field.

In our models the ionospheric magnetic field is represented in terms of the negative gradient of a potential expanded into spherical harmonic functions where apex coordinates and magnetic local time are used instead of the usual geocentric polar coordinates. We assume that, despite the use of the non-orthogonal coordinate systems, the poloidal ionospheric potential (Eq. 7a) defines a harmonic potential, which satisfies $\nabla^2 V^{ion} = 0$ in the source-free region. However, it is not obvious that this should be the case.

The most convincing way to show this is through direct analytical calculation of the Laplace equation given a potential as a function of the quasi-dipole co-latitude and the magnetic local time. If the Earth were spherical and had a purely dipolar magnetic field, the magnetic apex coordinates and components would be identical to centred dipole coordinates and components, in which case the condition $\nabla^2 V^{ion} = 0$ is satisfied. Unfortunately, for the case of a non-dipolar magnetic field and a spheroidal Earth, there is, to our knowledge, no closed expression of the Laplacian in magnetic apex

coordinates. Hence, we cannot analytically show that the Laplace equation is strictly satisfied. Nevertheless, we are able to perform numerical tests to check the validity of this condition. The results of these tests are shown below.

Table of Contents

- [1 Copyright and license notice](#)
- [2 Overview](#)
- [3 Least-squares fit of an internal potential](#)
 - [3.1 Load model of the divergence-free sheet current density](#)
 - [3.2 Generate equal-area surface grids at different radii using healpy](#)
 - [3.3 Compute synthetic data](#)
 - [3.4 Define target model based on geographic SH](#)
 - [3.5 Compute least-squares estimate of the target model parameters](#)
 - [3.6 Print misfit statistics](#)
 - [3.7 Global map of RMS misfit values](#)
- [4 Numerical computation of divergence and curl](#)
 - [4.1 Load model of the divergence-free sheet current density](#)
 - [4.2 Generate 3D grid](#)
 - [4.3 Compute synthetic data](#)
 - [4.4 Partial derivatives through differencing](#)
 - [4.5 Print statistics](#)
 - [4.6 Global maps of the divergence and curl](#)

```
In [1]: import multifit.core_utils as mco # multifit is our in-house modelling software
import multifit.tools as tools
import healpy as hp
import numpy as np
import chaosmagpy as cp
import pandas as pd
import quaternion as qt
import matplotlib.pyplot as plt
import apexpy

# this class evaluates our parameterization of the poloidal ionospheric field
class DFCModelExtension2(mco.DFCModelExtension):

    def colloc_matrix(self, data):

        N = max(len(data['X_0']), 1)

        coll = np.zeros((N, self.nbases))
        coll[:, 0] = 1.

        for n in range(1, self.nbases):
            coll[:, n] = data[f'X_{n}']

        return coll
```

```
R_SURF = cp.basicConfig['params.r_surf']
```

Least-squares fit of an internal potential

Load model of the divergence-free sheet current density

```
In [2]: dfc = DFCModelExtension2.from_json('../models/Model-FAC/DFC.json')
dfc.config['DFC.Apex'] = {'date': 2015.0, 'refh': 0.0, 'datafile': None}
# refh to 0.0, internal potential only

apex = apexpy.Apex(**dfc.config['DFC.Apex'])
```

Generate equal-area surface grids at different radii using healpy

```
In [3]: nside = 2**7
npix = hp.nside2npix(nside)
print('Number of pixels:', npix)
print('Approximate pixel size in degrees:', np.degrees(hp.nside2resol(nside)))
theta, phi = hp.pix2ang(nside, np.arange(npix)) # radians
theta, phi = np.degrees(theta), np.degrees(phi)
radius = np.linspace(R_SURF + 110., R_SURF + 750., num=15)
```

Number of pixels: 196608

Approximate pixel size in degrees: 0.45806485490898746

```
In [4]: date = np.datetime64('2015-01-01T00:00:00')

# specify external input parameters
data = pd.DataFrame({
    'Time': cp.mjd2000(date),
    'Radius': radius[0],
    'Theta': theta,
    'Phi': phi,
    'Pix': np.arange(npix)
})

dfs = []
for r in radius:
    df = data.copy(deep=True)
    df['Radius'] = r

    dfs.append(df)

data = pd.concat(dfs, axis=0, ignore_index=True)

# get apex coordinates
data['QDLat'], _, _, data['MLT'] = tools.qdipole(
    data['Time'], data['Radius'], data['Theta'], data['Phi'], apex=apex)

# get geodetic coordinates
data['Height'], data['Beta'] = cp.coordinate_utils.geo_to_gg(
    data['Radius'], data['Theta'])

with np.load('../data/solar_cycle_dataset_Model-FAC.npz') as file:
```

```

colloc_all = file['colloc_all']

X = [f'X_{k}' for k in range(dfc.nbases)]

for label, value in zip(X, colloc_all):
    data[label] = value

print('Number of data:', len(data))
data.head()

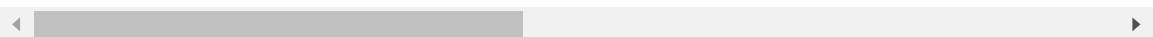
```

Number of data: 2949120

Out[4]:

| | Time | Radius | Theta | Phi | Pix | QDLat | MLT | Height | Beta | X_0 |
|---|--------|--------|----------|-------|-----|-----------|----------|------------|----------|-----|
| 0 | 5479.0 | 6481.2 | 0.365483 | 45.0 | 0 | 83.672432 | 5.950990 | 124.447134 | 0.363084 | 1.0 |
| 1 | 5479.0 | 6481.2 | 0.365483 | 135.0 | 1 | 83.604935 | 6.287543 | 124.447134 | 0.363084 | 1.0 |
| 2 | 5479.0 | 6481.2 | 0.365483 | 225.0 | 2 | 84.051254 | 6.335286 | 124.447134 | 0.363084 | 1.0 |
| 3 | 5479.0 | 6481.2 | 0.365483 | 315.0 | 3 | 84.123787 | 5.973794 | 124.447134 | 0.363084 | 1.0 |
| 4 | 5479.0 | 6481.2 | 0.730971 | 22.5 | 4 | 83.672371 | 5.676159 | 124.444537 | 0.726171 | 1.0 |

5 rows × 29 columns



Compute synthetic data

```

In [5]: # compute synthetic data
B = dfc.synth_values(data)

# add geocentric spherical up-south-east components to the dataframe
data['B_USE[0]'] = B[:, 0]
data['B_USE[1]'] = B[:, 1]
data['B_USE[2]'] = B[:, 2]

```

Define target model based on geographic SH

Define a spherical harmonic model based on geocentric spherical coordinates that will be fit to the synthetic data.

```

In [6]: model = mco.GaussModel(
    name='Geographic SH',
    params=0.,
    breaks=[data['Time'].min() - 1., data['Time'].max() + 1.],
    order=1,
    nmax=90, # truncation degree
    nmin=1
)
print('Number of model parameters:', model.size)

```

Number of model parameters: 8280

Compute least-squares estimate of the target model parameters

```
In [7]: # accumulate GtG and Gtd
N = len(data)
chunks = tools.crange(0, N, step=20_000, dtype=int)

GtG = np.zeros((model.size, model.size))
Gtd = np.zeros((model.size,))

for a, b in chunks:

    print(f'Running chunk: [{a}, {b}] (total of {N} data)', end='\r')

    subset = data.iloc[a:b]
    G = np.concatenate(model.create_design_matrix(subset), axis=0)
    d = subset[['B_USE[0]', 'B_USE[1]', 'B_USE[2]']].to_numpy().ravel(order='F')

    GtG += G.T @ G
    Gtd += G.T @ d
```

Running chunk: [2940000, 2949120] (total of 2949120 data)

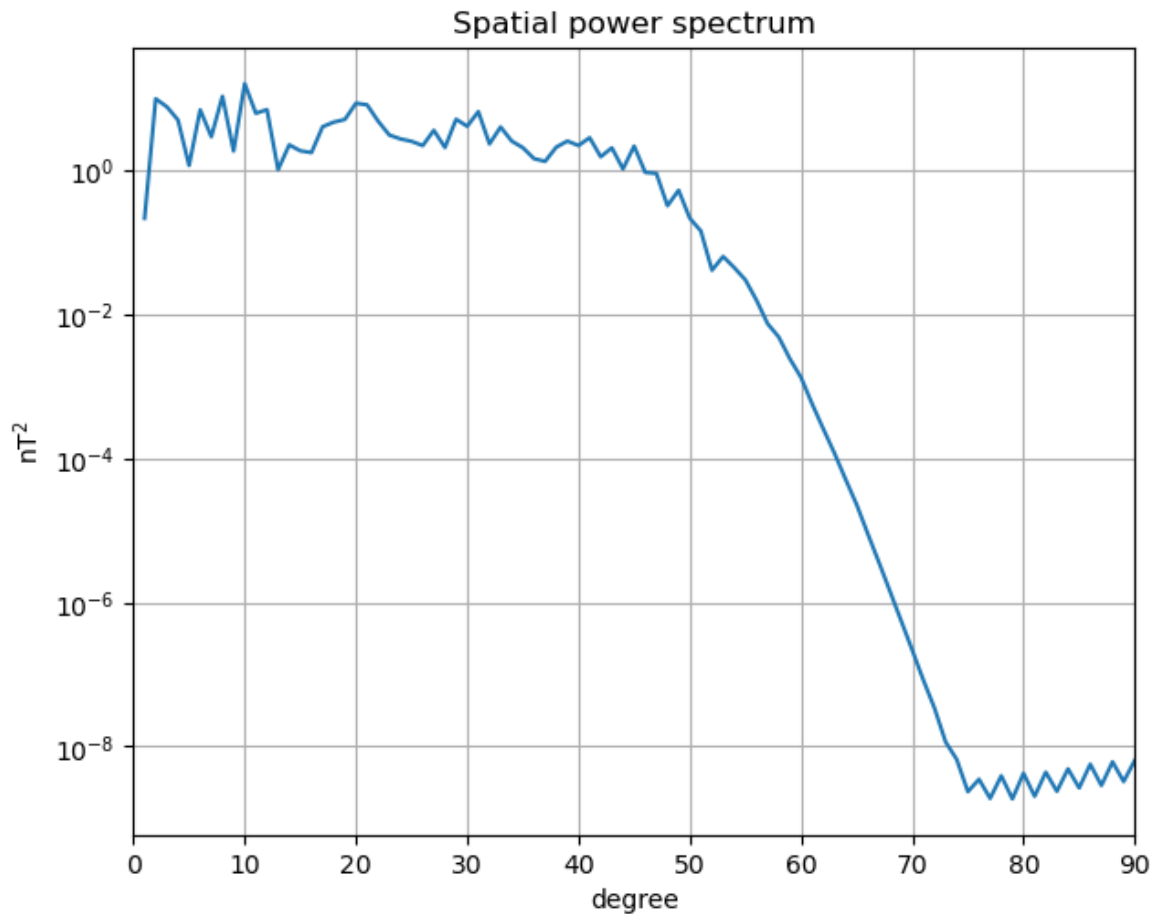
Solve for the target model parameters in the least-squares sense and plot the power spectrum.

```
In [15]: # solve for the target model parameters
model.params = np.linalg.solve(GtG, Gtd)

# plot the power spectrum
coeffs = np.squeeze(model.synth_coeffs(data.iloc[:1, :], pad=True))
spec = cp.model_utils.power_spectrum(coeffs)
cp.plot_power_spectrum(spec)

ax = plt.gca()
ax.set(ylabel='nT$^2$', title='Spatial power spectrum')

plt.savefig('power_spectrum_SH_fit.png')
plt.show()
```



Compute model predictions and add them to the dataframe

```
In [16]: chunks = tools.crange(0, N, step=1e5, dtype=int)
B_mod = np.concatenate([model.synth_values(data.iloc[a:b, :]) for (a, b) in chunks
                        axis=0])
B_res = B - B_mod

data['B_USE_Mod[0]'] = B_mod[:, 0]
data['B_USE_Mod[1]'] = B_mod[:, 1]
data['B_USE_Mod[2]'] = B_mod[:, 2]

data['B_USE_Res[0]'] = B_res[:, 0]
data['B_USE_Res[1]'] = B_res[:, 1]
data['B_USE_Res[2]'] = B_res[:, 2]
```

Print misfit statistics

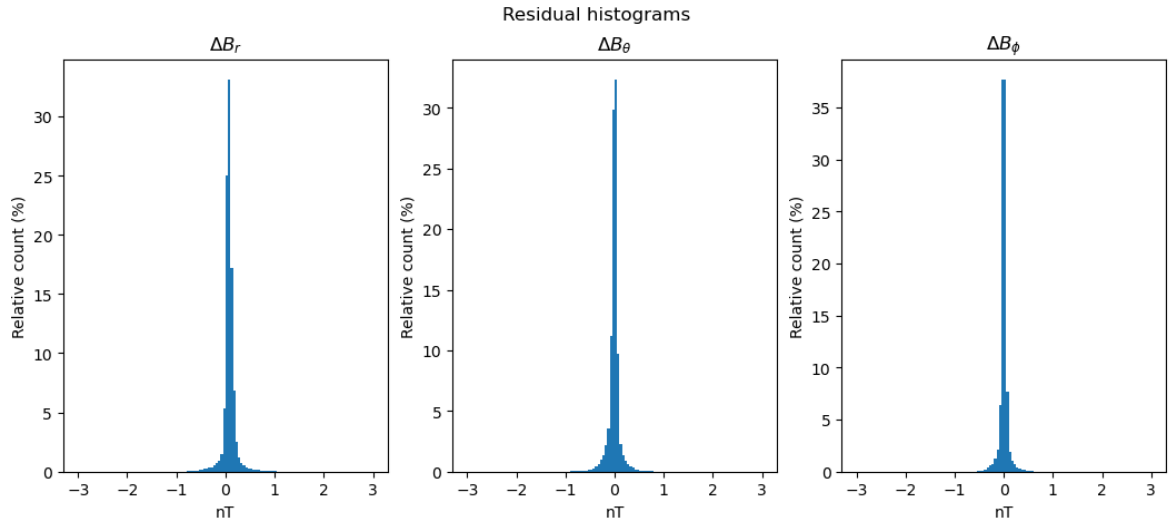
```
In [17]: fig, axes = plt.subplots(1, 3, figsize=(12.8, 4.8))

for n, ax in enumerate(axes):
    ax.hist(B_res[:, n], bins=np.arange(-3., 3.05, 0.05),
            weights=100./len(B_res)*np.ones((len(B_res),)))
    ax.set_ylabel('Relative count (%)')
    ax.set_xlabel('nT')

axes[0].set_title('$\Delta B_r$')
axes[1].set_title('$\Delta B_{\theta}$')
axes[2].set_title('$\Delta B_{\phi}$')

fig.suptitle('Residual histograms')
```

```
plt.show()
```



```
In [18]: print('Per component\n-----')
print('RMS Error:', np.round(tools.rms(B_res, axis=0), 3), 'nT')
print('Percentage of |error| < 1nT:', np.round(np.sum(np.abs(B_res) <= 1.0, axis=
len(B_res) * 100, 3))
print('Maximum Absolute Error:', np.round(tools.mae(B_res, axis=0), 3), 'nT', en

print('All components\n-----')
print('RMS Error:', np.round(tools.rms(B_res), 3), 'nT')
print('Percentage of |error| < 1nT:', np.round(np.sum(np.abs(B_res) <= 1.0) /
B_res.size * 100, 3))
print('Maximum Absolute Error:', np.round(tools.mae(B_res), 3), 'nT')
```

Per component

RMS Error: [0.164 0.117 0.082] nT

Percentage of |error| < 1nT: [99.719 99.967 99.998]

Maximum Absolute Error: [3.099 1.199 1.072] nT

All components

RMS Error: 0.126 nT

Percentage of |error| < 1nT: 99.895

Maximum Absolute Error: 3.099 nT

Next, compute the flux density for the poloidal ionospheric field passing through spheres of 15 different radii.

$$\frac{\Phi}{S} = \frac{1}{S} \oint \mathbf{B} \cdot \mathbf{ndS} = \frac{1}{S} \oint B_r dS \approx \frac{1}{N} \sum_{n=1}^N B_r(\mathbf{r}_n)$$

```
In [19]: flx = data[['Radius', 'B_USE[0]']].groupby(by='Radius').sum() / npix
flx['Flux ($10^6$ nT km$^2$)'] = flx[['B_USE[0]']].apply(
    lambda x: x.to_numpy() * 4*np.pi*radius**2 / 1e6)
flx.rename(columns={'B_USE[0]': 'Flux density (nT)'}, inplace=True)
flx.index.name = 'Radius (km)'
display(flx)
```

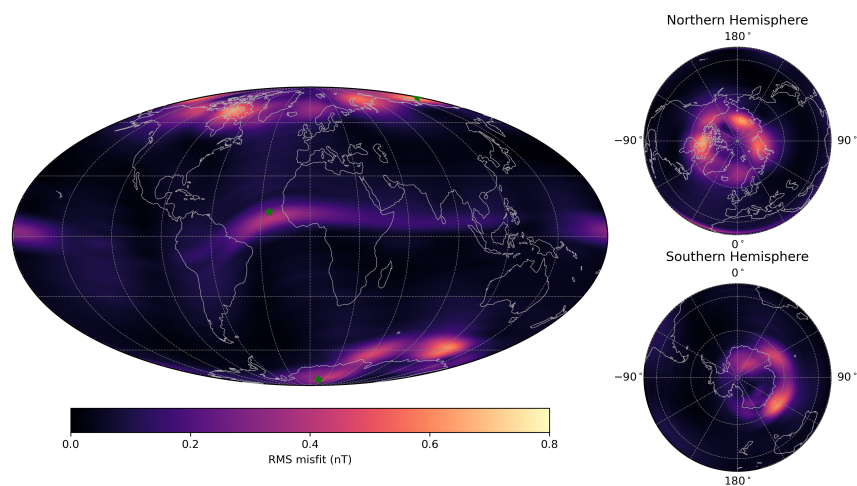
| | Flux density (nT) | Flux (10^6 nT km ²) |
|-------------|-------------------|------------------------------------|
| Radius (km) | | |
| 6481.200000 | 0.098098 | 51.782105 |
| 6526.914286 | 0.094083 | 50.365945 |
| 6572.628571 | 0.090253 | 48.994740 |
| 6618.342857 | 0.086598 | 47.666786 |
| 6664.057143 | 0.083109 | 46.380472 |
| 6709.771429 | 0.079778 | 45.134290 |
| 6755.485714 | 0.076596 | 43.926759 |
| 6801.200000 | 0.073557 | 42.756511 |
| 6846.914286 | 0.070652 | 41.622212 |
| 6892.628571 | 0.067876 | 40.522602 |
| 6938.342857 | 0.065222 | 39.456473 |
| 6984.057143 | 0.062685 | 38.422665 |
| 7029.771429 | 0.060258 | 37.420070 |
| 7075.485714 | 0.057936 | 36.447621 |
| 7121.200000 | 0.055714 | 35.504295 |

Global map of RMS misfit values

```
In [20]: # rms error per surface pixel (all components and radii)

m = tools.rms(data[['B_USE_Res[0]', 'B_USE_Res[1]', 'B_USE_Res[2]']].groupby(
    by=data['Pix']).apply(lambda x: tools.rms(x, axis=0)
), axis=1)
```

The following shows a global map of the computed RMS.



Numerical computation of divergence and curl

Load model of the divergence-free sheet current density

```
In [42]: # Load dfc model
dfc = DFCModelExtension2.from_json('../models/Model-FAC/DFC.json')
dfc.config['DFC.Apex'] = {'date': 2015.0, 'refh': 0.0, 'datafile': None}

apex = apexpy.Apex(**dfc.config['DFC.Apex'])
```

Generate 3D grid

```
In [43]: N_theta = 480
N_phi = 1200
N_r = 25

theta, dt = np.linspace(1e-4, 180. - 1e-4, N_theta, retstep=True);
phi, dp = np.linspace(-180., 180., N_phi, endpoint=False, retstep=True)

r = np.linspace(R_SURF + 400., R_SURF + 405., N_r)

theta_grid, phi_grid, r_grid = np.meshgrid(theta, phi, r)

print('Approximate pixel size is', np.sqrt(dt*dp*np.amin(np.sin(np.radians(theta
    'degree'))
print('Grid shape is', r_grid.shape)
print('Grid size is', r_grid.size)
```

Approximate pixel size is 0.0004435755152761934 degree
Grid shape is (1200, 480, 25)
Grid size is 14400000

```
In [6]: date = np.datetime64('2015-01-01T00:00:00')

# specify external input parameters
data = pd.DataFrame({
    'Time': cp.mjd2000(date),
    'Radius': r_grid.ravel(),
    'Theta': theta_grid.ravel(),
    'Phi': phi_grid.ravel()
})

# get apex coordinates
data['QDLat'], _, data['MALat'], data['MLT'] = tools.qdipole(
    data['Time'], data['Radius'], data['Theta'], data['Phi'], apex=apex)

# get geodetic coordinates
data['Height'], data['Beta'] = cp.coordinate_utils.geo_to_gg(
    data['Radius'], data['Theta'])

with np.load('../data/solar_cycle_dataset_Model-FAC.npz') as file:
    colloc_all = file['colloc_all']

X = [f'X_{k}' for k in range(dfc.nbases)]
```

```
for label, value in zip(X, colloc_all):
    data[label] = value
```

Compute synthetic data

```
In [787... B = dfc.synth_values(data)

Br = B[:, 0].reshape(r_grid.shape)
Bt = B[:, 1].reshape(r_grid.shape)
Bp = B[:, 2].reshape(r_grid.shape)
```

```
In [782... model = cp.CHAOS.from_mat('CHAOS-7.14.mat')

Br, Bt, Bp = model(cp.mjd2000(date), r_grid, theta_grid, phi_grid,
                    source_list='internal', nmax_static=5)
```

Partial derivatives through differencing

```
In [788... mu0 = 4*np.pi*1e-7

dBr_dp, dBr_dt, dBr_dr = np.gradient(Br, np.radians(phi), np.radians(theta), r)#
_, _, drBt_dr = np.gradient(r_grid*Bt, np.radians(phi), np.radians(theta), r)#
_, _, drBp_dr = np.gradient(r_grid*Bp, np.radians(phi), np.radians(theta), r)#
_, dsBp_dt, _ = np.gradient(np.sin(np.radians(theta_grid))*Bp, np.radians(phi),
                             np.radians(theta), r)#
dBt_dp, dBt_dt, _ = np.gradient(Bt, np.radians(phi), np.radians(theta), r)#
dBp_dp, _, _ = np.gradient(Bp, np.radians(phi), np.radians(theta), r)#

rotBr = 1e-3 / mu0 / (r_grid * np.sin(np.radians(theta_grid))) * (dsBp_dt - dBt_
rotBp = 1e-3 / mu0 / r_grid * (drBt_dr - dBr_dt)#
rotBt = 1e-3 / mu0 / r_grid * (dBr_dp/np.sin(np.radians(theta_grid)) - drBp_dr)#

divBr = 1e-3 / mu0 * (dBr_dr + 2 / r_grid * Br)#
divBt = 1e-3 / mu0 / r_grid * (dBt_dt + np.cos(np.radians(theta_grid)) /
                               np.sin(np.radians(theta_grid)) * Bt)#
divBp = 1e-3 / mu0 / (r_grid*np.sin(np.radians(theta_grid))) * dBp_dp#
divB = divBr + divBt + divBp
```

Print statistics

```
In [832... index = (
    (r_grid == np.amin(r)) | (r_grid == np.amax(r))
    | (theta_grid == np.amin(theta)) | (theta_grid == np.amax(theta))
    | (phi_grid == np.amin(phi)) | (phi_grid == np.amax(phi))
    | data['QDLat'].between(85., 90.).to_numpy().reshape(r_grid.shape)
    | data['QDLat'].between(-90., -85.).to_numpy().reshape(r_grid.shape)
) # remove boundary surface and points within 5 deg of magnetic poles

stats = pd.DataFrame({
    'RMS (nA/m$^2$)': [tools.rms(rotBr, where=~index),
                      tools.rms(rotBt, where=~index),
                      tools.rms(rotBp, where=~index),
                      tools.rms(divB, where=~index)],
    'MAE (nA/m$^2$)': [np.amax(np.abs(rotBr), initial=0., where=~index),
                      np.amax(np.abs(rotBt), initial=0., where=~index),
                      np.amax(np.abs(rotBp), initial=0., where=~index),
```

```

np.amax(np.abs(divB), initial=0., where=~index)]
}, index=[
'$\\frac{1}{\\mu_0}(\\nabla\\times\\mathbf{B}^{pol})\\cdot\\mathbf{\\hat{r}}$'
'$\\frac{1}{\\mu_0}(\\nabla\\times\\mathbf{B}^{pol})\\cdot\\mathbf{\\hat{\\theta}}$'
'$\\frac{1}{\\mu_0}(\\nabla\\times\\mathbf{B}^{pol})\\cdot\\mathbf{\\hat{\\phi}}$'
'$\\frac{1}{\\mu_0}\\nabla\\cdot\\mathbf{B}^{pol}$'
])

display(stats)

```

| | RMS (nA/m ²) | MAE (nA/m ²) |
|---|--------------------------|--------------------------|
| $\frac{1}{\mu_0}(\nabla \times \mathbf{B}^{ion}) \cdot \hat{\mathbf{r}}$ | 0.128200 | 1.182124 |
| $\frac{1}{\mu_0}(\nabla \times \mathbf{B}^{ion}) \cdot \hat{\boldsymbol{\theta}}$ | 0.325655 | 2.312794 |
| $\frac{1}{\mu_0}(\nabla \times \mathbf{B}^{ion}) \cdot \hat{\boldsymbol{\phi}}$ | 0.408041 | 3.050917 |
| $\frac{1}{\mu_0}\nabla \cdot \mathbf{B}^{ion}$ | 0.832435 | 5.972496 |

Global maps of the divergence and curl

The following shows global maps of the computed divergence and curl on the sphere at the center of the range of radii.

