(4) Geostatistical modeling

March 1, 2022

0.0.1 (3) Geostatistical modeling

This notebook will combine the Geo-structural model (2) and the results of the variogram analysis (3) to create a full 3D porosity model of the New Jersey Shelf using Gaussian simulation. The result will be a single realization of this model. Resolution can be adjusted.

[1]:
```python
# Import dependencies
import gempy as gp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import copy
import gstools as gs
import pyvista as pv
import pyvistaqt as pvqt
import PVGeo
import scipy.spatial.distance as dist
import scipy
from scipy.optimize import curve_fit
import datetime

import warnings
warnings.filterwarnings("ignore")

pd.set_option("display.precision", 2)
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda
install m2w64-toolchain`
C:\Users\Ariel\anaconda3\lib\site-packages\theano\configdefaults.py:560:
UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and
with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to
execute optimized C-implementations (for both CPU and GPU) and will default to
Python implementations. Performance will be severely degraded. To remove this
warning, set Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS
functions.
```

[2]:
```python
### User-defined functions
def perform_SGS(x, y, z, strike_angle, dip_angle, cond_data, x_range, y_range,␣
↪z_range):
    '''
    Function to create Random field based on GSTools routines.

        Arguments:
            x,y,z: Grid coordinates to estimate over.
            strike_angle, dip_angle: Rotational angles of Shelf system in␣
↪degrees.
            cond_data: Conditioning data for Random field.
            x_range, y_range, z_range: directional variogram ranges.
        Returns:
            cond_srf: GSTools Spatial Random field object (3D)
    '''

    # Convert given rotation angle to radians
    strike_angle = np.deg2rad(strike_angle)
    dip_angle = np.deg2rad(dip_angle)

    # Angle naming to GSTools
    alpha = 0
    beta = strike_angle
    gamma = dip_angle

    # Coordinate preparation
    coordinates = np.array([x,y,z])
    coordinates = coordinates.swapaxes(0,1)

    # Conditioning data preparation
    cond_pos = cond_data[:,:3]
    cond_pos = cond_pos.transpose(1,0)
    cond_val = cond_data[:,3]

    # Set conditioning data and anisotropies
    model = gs.Exponential(dim=3, var=np.var(cond_val), len_scale=[x_range,␣
↪y_range, z_range], angles=[alpha, beta, gamma])
    krige = gs.krige.Ordinary(model, cond_pos, cond_val)
    cond_srf = gs.CondSRF(krige)

    # Perform SGS
    cond_srf((coordinates[:,0],coordinates[:,1], coordinates[:,2]),␣
↪mesh_type='unstructured')

    return cond_srf

def plot_block_model(field):
```

```
    '''
    Function for plotting 3D Block Model based on GStools result

    Arguments:
        field: GSTools Spatial Random Field object (3D)
    Returns:
        p: pyvista plotter with voxel model
    '''

    # Create pyvista mesh for field
    pc = field.to_pyvista()

    # Find voxel size
    spacing = lambda arr: np.unique(np.diff(np.unique(arr)))
    voxelsize = spacing(pc.points[:,0]), spacing(pc.points[:,1]), spacing(pc.
↪points[:,2])

    # Pvgeo way of voxelizing semi-unstructured grid
    grid = PVGeo.filters.VoxelizePoints(dx=voxelsize[0][0], dy=voxelsize[1][0],␣
↪dz=voxelsize[2][0], estimate=False).apply(pc)

    # Plotting
    #p = pv.Plotter(notebook=True)
    p = pvqt.BackgroundPlotter()
    #p.add_mesh(grid, opacity=1, show_edges=True)
    cmap = plt.cm.get_cmap("viridis", 5)
    #p.add_mesh(grid, opacity=1, show_edges=False, lighting=False, cmap=cmap)
    p.add_mesh(grid, opacity=1, show_edges=False, lighting=False,␣
↪cmap="viridis") # continuous cmap
    #p.add_mesh(pc, point_size=5, cmap='viridis')

    return p

def extract_domain(sol, unit):
    '''
    Extract domain coordinates from gempy model by unit name

    Arguments:
        sol: Gempy solution object.
        unit: string name of gempy surface
    Returns:
        dom_x, dom_y, dom_z: coordinates of domain
    '''

    # Round Lithlogy block from gempy
    rounded_lithblock = sol.lith_block.round(0)
    rounded_lithblock = rounded_lithblock.astype(int)
```

```
    # Mask by array of input surfaces (by id, can be from different series)
    mask = np.isin(rounded_lithblock, [ref_dict[unit]])

    # Get coordinates by mask
    dom_grid = sol.grid.values[mask]

    # Split coordiantes
    dom_x = dom_grid[:,0]
    dom_y = dom_grid[:,1]
    dom_z = dom_grid[:,2]

    return dom_x, dom_y, dom_z
```

0.0.2 1. Reload and recalcualte Geo-structural model

[3]:
```
# Load model from notebook (2)
geo_data = gp.load_model('Geo-structural model NJ shelf')
```

```
Active grids: ['regular']
```

[4]:
```
%%time
# Set interpolator
interp_data = gp.set_interpolator(geo_data, compile_theano=True,
                                  theano_optimizer='fast_compile')
```

```
Setting kriging parameters to their default values.
Compiling theano function…
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                                  values
range                          150731.18
$C_o$                        540949761.9
drift equations  [3, 3, 3, 3, 3, 3, 3, 3, 3]
Wall time: 4.89 s
```

[5]:
```
# Ajdust resolution here if necessary
geo_data.set_regular_grid([0, 69000, 0, 134000, -1700, 0], [138,268,85]);

# Alternatives
#geo_data.set_regular_grid([0, 69000, 0, 134000, -1700, 0], [69,134,42]) # Half␣
 ↪resolution
```

```
#geo_data.set_regular_grid([41999, 42000, 0, 134000, -1700, 0], [2,1600,320]) #␣
↪2.5D model
```

```
Active grids: ['regular']
```

[6]:
```
%%time
# Compute model solution
sol = gp.compute_model(geo_data)
```

```
Wall time: 26min 4s
```

[8]:
```
# View model 3D
gpv = gp.plot_3d(geo_data, ve=30, plotter_type='background', show_data=False)
gpv.p.camera_position = (320, 200, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_13952/829275504.py in <module>
      1 # View model 3D
----> 2 gpv = gp.plot_3d(geo_data, ve=30, plotter_type='background',␣
↪show_data=False)
      3 gpv.p.camera_position = (320, 200, 3)

~\anaconda3\lib\site-packages\gempy\plot\plot_api.py in plot_3d(model,␣
↪plotter_type, show_data, show_results, show_surfaces, show_lith, show_scalar,␣
↪show_boundaries, show_topography, scalar_field, ve,␣
↪kwargs_plot_structured_grid, kwargs_plot_topography, kwargs_plot_data, image,␣
↪off_screen, **kwargs)
    324     gpv = GemPyToVista(model, plotter_type=plotter_type, **kwargs)
    325     if show_surfaces and len(model.solutions.vertices) != 0:
--> 326         gpv.plot_surfaces()
    327     if show_lith is True and model.solutions.lith_block.shape[0] != 0:
    328         gpv.plot_structured_grid('lith', **kwargs_plot_structured_grid)

~\anaconda3\lib\site-packages\gempy\plot\vista.py in plot_surfaces(self,␣
↪surfaces, surfaces_df, clear, **kwargs)
    462         select_active = surfaces_df['isActive']
    463         for idx, val in surfaces_df[select_active][['vertices', 'edges', ␣
↪'color', 'surface', 'id']].dropna().iterrows():
--> 464             surf = pv.PolyData(val['vertices'], np.insert(val['edges'],␣
↪0, 3, axis=1).ravel())
    465             # surf['id'] = val['id']
    466             self.surface_poly[val['surface']] = surf

~\anaconda3\lib\site-packages\pyvista\core\pointset.py in __init__(self, *args,␣
↪**kwargs)
    179                 self._from_arrays(args[0], args[1], deep)
    180             else:
```

```
--> 181                 raise TypeError('Invalid input type')
    182         else:
    183             raise TypeError('Invalid input type')

TypeError: Invalid input type
```

0.0.3 2. Loading and preparing conditioning data (Porosity well data)

[9]:
```
# Load data
df = pd.read_csv("Data/Wells/Complete_set_corrected.csv")
#df = pd.read_csv("Data/Wells/Complete_set_corrected_M28blind.csv")  #uncomment␣
↪for blind test data

df.head()
```

[9]:

	X	Y	Z	Porosity	Sequence	Well
0	27400	59300	-46.39	17.5	m1	m27
1	27400	59300	-35.77	24.7	m1	m27
2	27400	59300	-41.38	24.9	m1	m27
3	27400	59300	-171.37	25.4	m41	m27
4	27400	59300	-113.41	25.5	m1	m27

0.0.4 2.1 Detrend

[10]:
```
# Routine for detrending from notebook (3)

# Trend models
def exponential_trend(x, a, b):
    return a*np.exp(b*x)

def linear_trend(x, a, b):
    return a+b*x

# Vertical trend
xdata = df["Z"].values*(-1)# make depth positve
ydata = df["Porosity"].values

p0 = [41,0.001]
popt_exp, pcov_exp = curve_fit(exponential_trend, xdata, ydata, p0)

df["Residuals_temp"] = df["Porosity"].values - exponential_trend(df["Z"].
↪values, *popt_exp)
df.head()

# Horizontal trend
xdata = np.unique(df["Y"].values)
```

```
# ydata = np.array([np.mean(df[df["Well"]=="m27"]["Residuals_temp"]),
#                     np.mean(df[df["Well"]=="m28"]["Residuals_temp"]),
#                     np.mean(df[df["Well"]=="m29"]["Residuals_temp"])])
ydata = np.array([np.mean(df[df["Well"]=="m27"]["Residuals_temp"]),
                  np.mean(df[df["Well"]=="m29"]["Residuals_temp"])])
popt_lin, pcov_lin = curve_fit(linear_trend, xdata, ydata)


df["Residuals"] = df["Residuals_temp"].values - linear_trend(df["Y"].values,␣
 ↪*popt_lin)

df.head()
```

[10]:

	X	Y	Z	Porosity	Sequence	Well	Residuals_temp	Residuals
0	27400	59300	-46.39	17.5	m1	m27	-26.79	-29.08
1	27400	59300	-35.77	24.7	m1	m27	-19.64	-21.93
2	27400	59300	-41.38	24.9	m1	m27	-19.41	-21.70
3	27400	59300	-171.37	25.4	m41	m27	-18.31	-20.61
4	27400	59300	-113.41	25.5	m1	m27	-18.48	-20.77

0.0.5 2.2 n-score transform

[11]:

```
# Routine for n-score transform from notebook (3)

def cdf(d, bins=12 ):
    N = len( d )
    counts, intervals = np.histogram( d, bins=bins )
    h = np.diff( intervals ) / 2.0
    f, finv = np.zeros((N,2)), np.zeros((N,2))
    idx, k, T = 0, 0, float( np.sum( counts ) )
    for count in counts:
        for i in range( count ):
            x = intervals[idx]+h[0]
            y = np.cumsum( counts[:idx+1] )[-1] / T
            f[k,:] = x, y
            finv[k,:] = y, x
            k += 1
        idx += 1
    return f, finv

def fit(d):
    x, y = d[:,0], d[:,1]
    def f(t):
        if t <= x.min():
            return y[ np.argmin(x) ]
        elif t >= x.max():
            return y[ np.argmax(x) ]
```

```
        else:
            intr = scipy.interpolate.interp1d( x, y )
            return intr(t)
    return f

# transform data to normal dist
def to_norm( data, bins=10000):
    mu = np.mean( data )
    sd = np.std( data )
    z = ( data - mu ) / sd
    f, inv = cdf( z, bins=bins )
    z = scipy.stats.norm(0,1).ppf( f[:,1] )
    z = np.where( z==np.inf, np.nan, z )
    z = np.where( np.isnan( z ), np.nanmax( z ), z )
    param = ( mu, sd )
    return z, inv, param, mu, sd

# transform data from normal dist back
def from_norm( data, inv, param, mu, sd ):
    h = fit( inv )
    f = scipy.stats.norm(0,1).cdf( data )
    z = [ h(i)*sd + mu for i in f ]
    return z
```

[12]:
```
# N-score transformation all
por_residuals_norm_transform, inv, param, m, sd = to_norm(df["Residuals"].
↪values)

# Add to dataframe
df["Nscore Residuals"]=por_residuals_norm_transform

df.head()
```

[12]:

	X	Y	Z	Porosity	Sequence	Well	Residuals_temp	Residuals	\
0	27400	59300	-46.39	17.5	m1	m27	-26.79	-29.08	
1	27400	59300	-35.77	24.7	m1	m27	-19.64	-21.93	
2	27400	59300	-41.38	24.9	m1	m27	-19.41	-21.70	
3	27400	59300	-171.37	25.4	m41	m27	-18.31	-20.61	
4	27400	59300	-113.41	25.5	m1	m27	-18.48	-20.77	

	Nscore Residuals
0	-2.96
1	-2.74
2	-2.60
3	-2.50
4	-2.42

0.0.6 2.3 Reference dictionary (between Geo-strucutral model and borehole data)

[13]:
```
# Dictionary reference between unit name and gempy reference id
ref_dict = dict(geo_data.surfaces.df[['surface', 'id']].values)
ref_dict2 = {
    "m1": "m1",
    "m41": "m4_1",
    "m5": "m5",
    "m54": "m5_4",
    "m58": "m5_8",
    "m6": "m6",
    "o1": "o1",}

print(ref_dict)
```

```
{'SeaFloor': 1, 'm1': 2, 'm4_1': 3, 'm5': 4, 'm5_4': 5, 'm5_8': 6, 'm6': 7,
'o1': 8, 'basement': 9}
```

0.0.7 3. Loading variogram parameters

[14]:
```
# Getting varigoram parameters from (3)
variogram_results = pd.read_csv("Results/Variogram results table.csv",␣
↪index_col=0)
variogram_results
```

[14]:
```
  Sequence        Model  Z Range (Well Data)  Y Angle (Attr)  Y Range (Attr)  \
0       m1  Exponential                13.67           -0.12         2403.51
1      m41  Exponential                13.67           -0.19         2393.22
2       m5  Exponential                13.67           -0.11         3446.76
3      m54  Exponential                13.67           -0.07         2433.96
4      m58  Exponential                13.67           -0.55         1822.08
5       m6  Exponential                13.67           -0.58         2271.58
6       o1  Exponential                13.67           -0.34         1364.54

   X Angle (Attr)  X Range (Attr) Anisotropy Ratio (X/Y/Z)  \
0            0.00         3796.74             [278 176   1]
1           -0.01         3796.74             [278 175   1]
2           -0.00         4180.39             [306 252   1]
3            0.08         4180.39             [306 178   1]
4            0.18         3075.89             [225 133   1]
5            0.06         4398.60             [322 166   1]
6            0.47         7425.56             [543 100   1]

                                            Comment
0  *Strike direction corrected (underlying sequen…
1                                               NaN
2  *Strike direction corrected (underlying sequen…
```

```
3                                                   NaN
4                                                   NaN
5                                                   NaN
6                                                   NaN
```

0.0.8 4. Geostatistics on single domains

[14]:
```
# Defining unit for analysis
unit="m41"
```

[15]:
```
# Check drillhole locations in 3D plot

# Create plotter object with gempy model
gpv = gp.plot_3d(geo_data, plotter_type='background', show_data=False)

# Add conditioning data for specific sequence
poly = pv.PolyData(df[df["Sequence"]==unit].values[:,:3].astype("float64"))
poly["My Labels"] = df[df["Sequence"]==unit].values[:,3].astype("float64")
gpv.p.add_point_labels(poly, "My Labels", point_size=10, font_size=36,␣
↪point_color='black')

# Vertical exaggeration
gpv.p.set_scale(zscale=30)

gpv.p.show()
```

[15]:
```
# Extract domain from gempy model
domain_x, domain_y, domain_z = extract_domain(sol, ref_dict2[unit])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_13952/864679193.py in <module>
      1 # Extract domain from gempy model
----> 2 domain_x, domain_y, domain_z = extract_domain(sol, ref_dict2[unit])

NameError: name 'unit' is not defined
```

0.0.9 4.1 Gaussian simulation

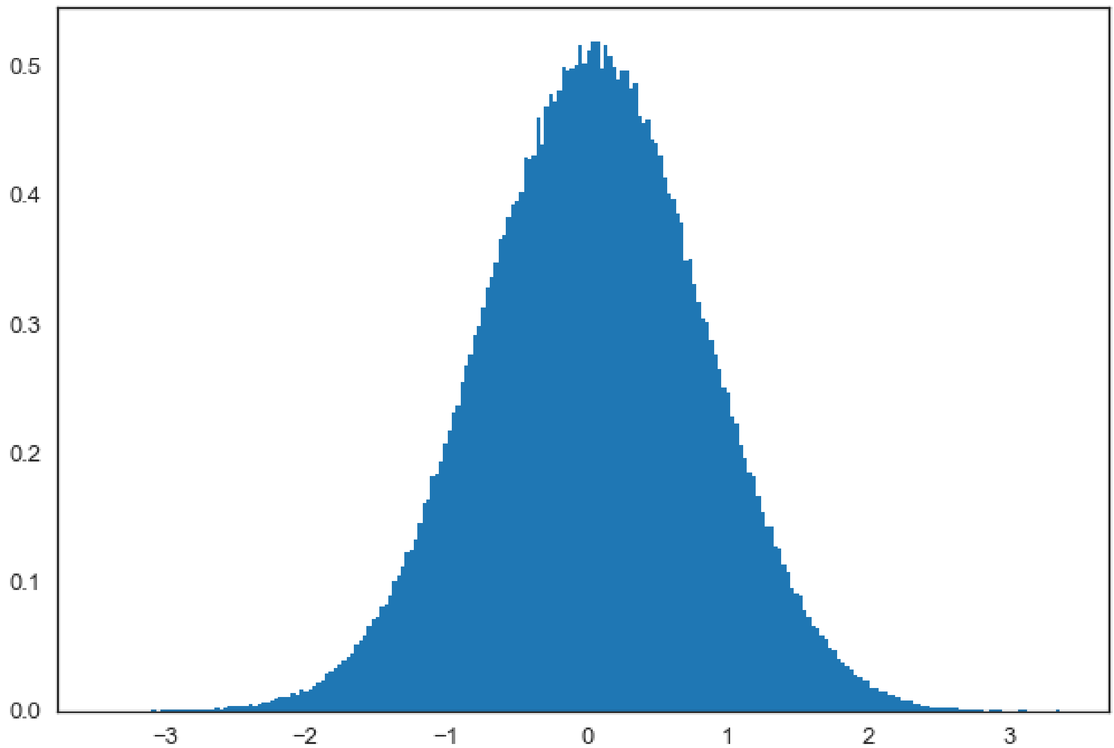
[17]:
```
%%time

# Performing the SGS with specified directional ranges and angle for rotation␣
↪around x-axis
field = perform_SGS(domain_x, domain_y, domain_z,
```

```
                    ␣
↪strike_angle=variogram_results[variogram_results["Sequence"]==unit]["X Angle␣
↪(Attr)"].values[0],
                    ␣
↪dip_angle=variogram_results[variogram_results["Sequence"]==unit]["Y Angle␣
↪(Attr)"].values[0],
                     cond_data=np.hstack((df[df["Sequence"]==unit].values[:,:3].
↪astype("float64"),
                                 df["Nscore Residuals"][df["Sequence"]==unit].
↪values.astype("float64").reshape(-1,1))),
                    ␣
↪x_range=variogram_results[variogram_results["Sequence"]==unit]["X Range␣
↪(Attr)"].values[0],
                    ␣
↪y_range=variogram_results[variogram_results["Sequence"]==unit]["Y Range␣
↪(Attr)"].values[0],
                    ␣
↪z_range=variogram_results[variogram_results["Sequence"]==unit]["Z Range␣
↪(Well Data)"].values[0])
```

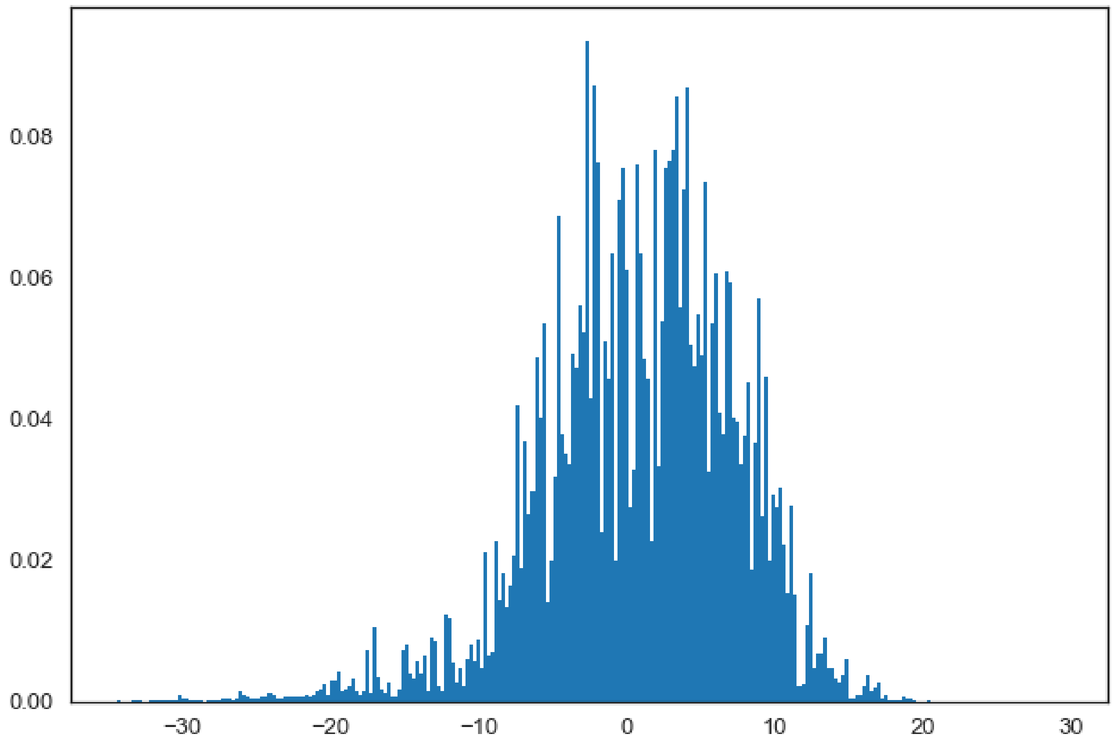
```
Wall time: 4.36 s
```

0.0.10 4.2 Back-transformation and adding trends

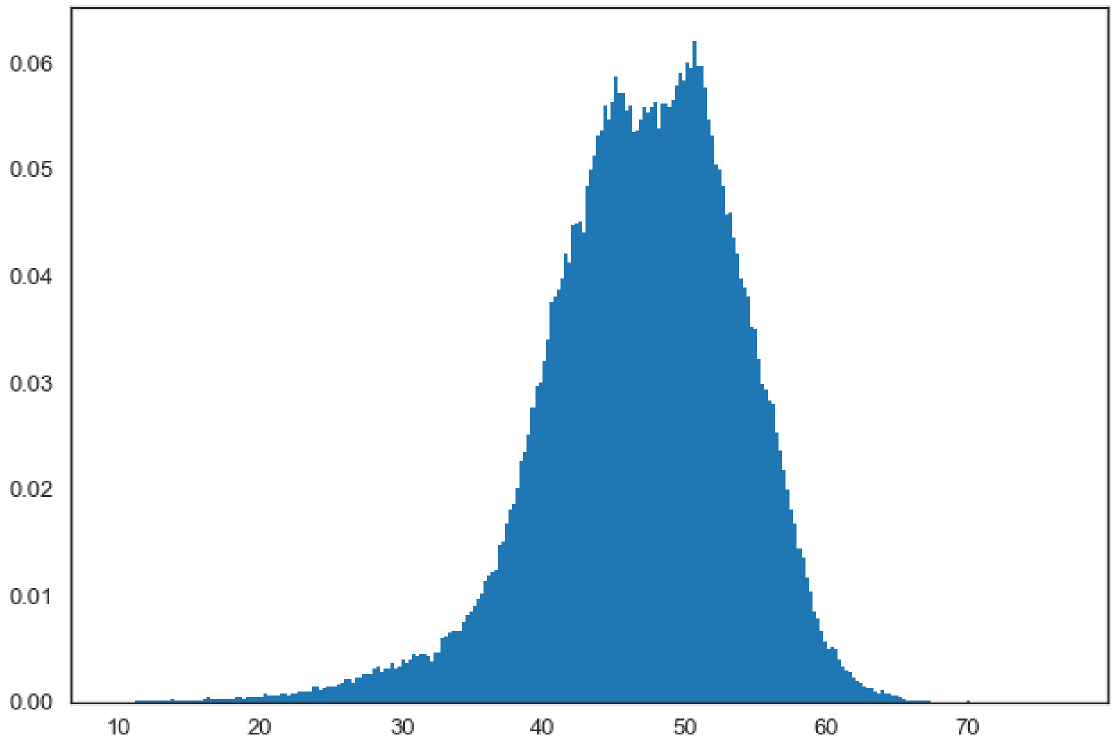
[18]:
```
# Plot histogram of Random field result
plt.hist(field.field, bins='fd', density=True);
```



[19]:
```
# N-score back-transform
back_transformed = from_norm(field.field, inv, param, m, sd)
plt.hist(back_transformed, bins='fd', density=True);
```



[20]:
```
# Reapply trends
# Add linear y trend back
retrended_temp = back_transformed + linear_trend(field.pos[1,:], *popt_lin)
# Add exponential z trend back
retrended = retrended_temp + exponential_trend(field.pos[2,:], *popt_exp)

plt.hist(retrended, bins='fd', density=True);
```



[21]:
```
# Overwrite field result with back_transformed and retrended data
field.field=retrended
```

[22]:
```
# Plot simulated field and drillhole data
p = plot_block_model(field)

# Vertical exaggeration
p.set_scale(zscale=30)

p.show()
```

0.0.11 5. Complete Model

[16]:
```
%%time
# Routine to calculate for all sequences:

# Empty dict for results
res_dict = {}

# Calculate model for each sequence
for unit in variogram_results["Sequence"].values:

    # Extract domain
```

```
    domain_x, domain_y, domain_z = extract_domain(sol, ref_dict2[unit])

    # Create Gaussian field
    field = perform_SGS(domain_x, domain_y, domain_z,
                  ␣
↪strike_angle=variogram_results[variogram_results["Sequence"]==unit]["X Angle␣
↪(Attr)"].values[0],
                  ␣
↪dip_angle=variogram_results[variogram_results["Sequence"]==unit]["Y Angle␣
↪(Attr)"].values[0],
                        cond_data=np.hstack((df[df["Sequence"]==unit].values[:,:3].
↪astype("float64"),
                                        df["Nscore Residuals"][df["Sequence"]==unit].
↪values.astype("float64").reshape(-1,1))),
                  ␣
↪x_range=variogram_results[variogram_results["Sequence"]==unit]["X Range␣
↪(Attr)"].values[0],
                  ␣
↪y_range=variogram_results[variogram_results["Sequence"]==unit]["Y Range␣
↪(Attr)"].values[0],
                  ␣
↪z_range=variogram_results[variogram_results["Sequence"]==unit]["Z Range␣
↪(Well Data)"].values[0])

    # Back-transform and reapply trends
    back_transformed = from_norm(field.field, inv, param, m, sd)
    retrended_temp = back_transformed + linear_trend(field.pos[1,:], *popt_lin)
    retrended = retrended_temp + exponential_trend(field.pos[2,:], *popt_exp)

    field.field=retrended

    res_dict[unit]=field
```

Wall time: 3min 41s

[17]:
```
# Create single dataframe with unified results
results_df  = pd.DataFrame(columns=("X", "Y", "Z", "Porosity", "Sequence"))

for unit in variogram_results["Sequence"].values:

    temp_list =  np.empty(len(res_dict[unit].pos[0,:]), dtype='U100')
    temp_list[:] = unit
```

```
    temp_df = pd.DataFrame(
        np.array([res_dict[unit].pos[0,:],
        res_dict[unit].pos[1,:],
        res_dict[unit].pos[2,:],
        res_dict[unit].field,
        temp_list]).T, columns=("X", "Y", "Z", "Porosity", "Sequence"))

    results_df = results_df.append(temp_df)


results_df = results_df.astype({'X': 'float64', 'Y': 'float64', 'Z': 'float64',␣
 ↪'Porosity': 'float64',})

results_df
```

[17]:

	X	Y	Z	Porosity	Sequence
0	250.0	250.0	-30.0	37.70	m1
1	250.0	750.0	-30.0	40.26	m1
2	250.0	1250.0	-30.0	37.82	m1
3	250.0	1750.0	-30.0	35.56	m1
4	250.0	2250.0	-30.0	41.24	m1
…	…	…	…	…	…
253698	68750.0	133750.0	-1530.0	27.87	o1
253699	68750.0	133750.0	-1510.0	36.71	o1
253700	68750.0	133750.0	-1490.0	42.29	o1
253701	68750.0	133750.0	-1470.0	40.39	o1
253702	68750.0	133750.0	-1450.0	37.46	o1

[1412810 rows x 5 columns]

[18]:

```
# Plot full model without gempy

#cmap = plt.cm.get_cmap("viridis", 6)
p = pvqt.BackgroundPlotter()

pc = pv.PolyData(np.c_[results_df["X"].values, results_df["Y"].values,␣
 ↪results_df["Z"].values])

pc["Porosity"]=results_df["Porosity"].values

spacing = lambda arr: np.unique(np.diff(np.unique(arr)))
voxelsize = spacing(pc.points[:,0]), spacing(pc.points[:,1]), spacing(pc.
 ↪points[:,2])

pc = pc.cast_to_unstructured_grid()
```

```
grid = PVGeo.filters.VoxelizePoints(dx=voxelsize[0][0], dy=voxelsize[1][0],␣
↪dz=voxelsize[2][0], estimate=False).apply(pc)

#p.add_mesh(grid, opacity=1, show_edges=False, lighting=False, cmap=cmap)
p.add_mesh(grid, opacity=1, show_edges=False, lighting=False, cmap="viridis")

p.set_scale(zscale=30)
p.camera_position = (-320, -200, 3)
p.show_grid(xlabel="X [m]", ylabel="Y [m]", zlabel="Z [m]")

p.show()
```

[19]:

```
# Save realization as result to csv
realization_name = str("Model_realization_"+datetime.datetime.now().
↪strftime("%Y%m%d"))
results_df.to_csv("Results/"+realization_name+".csv", index=False)
```