Turbgen:

Turbulence field generation and constrained turbulence simulation package

Hipersim 0.0.24 Turbgen documentation.

Nikolay Dimitrov, DTU Wind Energy, 30/11/2021.

1. Introduction

The *turbgen* package is part of the *hipersim* collection.

The turbulence fields use the Mann spectral definition (Mann, 1994)[1].

The turbulence field generation procedure follows the algorithm defined in (Mann, 1998)[2].

The constrained turbulence generation functions use the algorithms defined in (Dimitrov & Natarajan, 2017)[3].

2. Installation and use

Installing *turbgen* is done by installing the parent *hipersim* package:

```
pip install hipersim
```

Importing *turbgen* into the Python environment:

```
from hipersim.turbgen import turbgen
```

Turbulence field objects

Turbgen uses the `turb_field` object class for representing turbulence fields. Object definition takes place with the following command:

```
T = turbgen.turb_field()
```

Field attributes can be specified together with the initial definition:

```
T = turbgen.turb_field(Nx = 1024, Ny = 32, Nz = 32, dx = 2, dy = 2, dz = 2)
```

The `turbgen.turb_field.params` dictionary stores all the field properties. They can be specified and re-assigned after field initialization, such as:

```
T.params['Nx'] = 8192
```

Attributes directly assigned as with the above command will overwrite any initial definition. Missing attribute definitions will lead to default values being assigned automatically during field generation. The complete list of the properties in `turbgen.turb_field.params` is given below (default values in brackets):

- `Nx` : Dimension of the turbulence box in longitudinal (x) direction (default = 8192)
- `Ny` : Dimension of the turbulence box in transverse horizontal (y) direction (default = 32)
- `Nz` : Dimension of the turbulence box in transverse vertical (z) direction (default = 32)
- `dx` : Spacing in meters between data points along the x-coordinate (default = 1)
- `dy` : Spacing in meters between data points along the y-coordinate (default = 1)
- `dz` : Spacing in meters between data points along the z-coordinate (default = 1)
- `L` : Mann model turbulence length scale parameter L (Mann, 1994) (default = 29.4)
- `Gamma` : Mann model turbulence anisotropy parameter Γ (Mann, 1994) (default = 3.9)
- `alphacpsilon` : Mann model turbulence parameter $(\alpha\varepsilon)^{2/3}$ (Mann, 1994) (default = 1.0)
- `HighFreqComp` : Defines whether high-frequency compensation is applied to the turbulence generation as defined in (Mann, 1998). Default = 0 (no high-frequency compensation).
- `SaveToFile` : Option specifying whether the generated fields should be saved in the local folder (default = 0).

- `BaseName` : The initial part of the generated turbulence file names (default = 'turb*py'). The complete file name is defined as `[BaseName , , SeedNo , _ , component , '.bin']`.

Methods

`turbgen.turb_field.generate()` simulates the turbulence field with properties as specified in `turbgen.turb_field.params` , and outputs the field components u, v, w as object attributes and as variables if requested. If `params.SaveToFile == 1` , the field components are saved to the local directory as binary files.

`turbgen.turb_field.constrain(Constraints)` simulates a turbulence field with properties as specified in `turbgen.turb_field.params` , which satisfies the constraints specified in the `Constraints` array. The field components u, v, w are output as object attributes, as variables, or saved to disk. `Constraints` must be a 2-D numpy array with 6 columns. The first 3 columns specify the x, y, z locations of the constraint points, respectively, while the last 3 columns specify the u, v, w components of the desired wind velocity at the desired constraint location. Each row of the array corresponds to one constraint point.

`turbgen.turb_field.output()` adds steady-state components (e.g. mean wind speed, wind shear, wind field rotations) and variance scaling to the zero-mean wind field which is generated by default. The properties of the mean field are defined in the `TurbOptions` dictionary which should be added as an attribute to the `turb_field` object. Returns the u, v, w components of the updated field as variables or saves them to a file if specified by the `SaveToFile` option. Note that the field attributes are not changed and the `turbgen.turb_field.(u,v,w)` attributes remain the original zero-mean values.

Functions

`turbgen.generate_field.generate_field()` - alias to the `turbgen.turb_field.generate()` method. Does not require that a turbulence field object is defined in advance. Returns the u, v, w turbulence components as numpy arrays. Requires full input definition in order to run successfully. The `turbgen.turb_field.generate()` method runs by calling the `turbgen.generate_field.generate_field()` function.

`turbgen.constrain_field.constrain_field()` - alias to the `turbgen.turb_field.constrain()` method. Does not require that a turbulence field object is defined in advance. Returns the u, v, w turbulence components as numpy arrays. Requires full input definition (including constraint specifications) in order to run successfully. The `turbgen.turb_field.constrain()` method runs by calling the `turbgen.generate_field.constrain_field()` function.

`turbgen.manntensor.manntensor()` \ `turbgen.manntensor.manntensorcomponents()` \ `turbgen.manntensor.manntensorsqrt()` \ `turbgen.manntensor.manntensorsqrtcomponents()` \ The above functions provide the Mann tensor components with various array shapes and using slightly different algorithms. The different implementations are required for increasing the computational efficiency of the turbulence generation/constraint code.

`turbgen.turb_utils.output_field()` - alias to the `turbgen.turb_field.output()` method. The `turbgen.turb_field.output()` method runs by calling the `turbgen.turb_utils.output_field()` function.

Output options

The default file format for turbulence box files is the binary format used by Hawc2 (`.bin` extension), with three separate files for the u, v and w components of the field. The fields generated are with zero mean and the variance is dictated by the choice of turbulence spectrum parameters ($L, \Gamma, \alpha\varepsilon$). The `output()` method and `turbgen.turb_utils.output_field()` function allow adding a mean wind correction to the field, as well as variance scaling and rotation of the wind field. The resulting wind fields are consistent with the TurbSim definition and can be saved in a file with `.wnd` extension, compatible with the TurbSim/Bladed file format. The inputs required for the wind field modifications are provided in the `TurbOptions` dictionary which must be added as an attribute to the `turb_field` object, or directly given as an input to the `turbgen.turb_utils.output_field()` function. The following options are accepted:

- `FileFormat` : `0` (default), `'Hawc2'` , or `'bin'` will result in a turbulence box output in the standard format used by Hawc2. `'wnd'` , `'TurbSim'` , or `'Bladed'` will result in a single binary file output with a `.wnd` extension and with a format consistent with the TurbSim output.
- `Umean` is the mean wind speed in m/s
- `zHub` is the height above ground at the center of the turbulence box (typically the hub height), given in meters. Required in order to compute the wind shear profile.
- `ShearLaw` specifies the wind shear profile definition. `pwr` corresponds to a power-law profile using a wind shear exponent, and `log` corresponds to a logarithmic profile using the surface roughness length as a parameter. The default shear law is `pwr` .

- `alpha` is the wind shear exponent used in the power-law definition of wind shear
- `z0` is the surface roughness length used in the logarithmic law definition of wind shear
- `TI_u`, `TI_v`, `TI_w` are the target turbulence intensities in the u, v, w components respectively
- `Yaw` specifies the rotation of the wind field around the yaw (z) axis in degrees.
- `Pitch` specifies the rotation of the wind field around the pitch (y) axis in degrees. Corresponds to flow inclination.
- `Roll` specifies the rotation of the wind field around the roll (x) axis in degrees.

References

[1] Mann, J. (1994). The spatial structure of neutral atmospheric surface-layer turbulence. Journal of Fluid Mechanics, 273, 141–168. https://doi.org/10.1017/S0022112094001886 \ [2] Mann, J. (1998). Wind field simulation. Probabilistic Engineering Mechanics. https://doi.org/10.1016/S0266-8920(97)00036-2 \ [3] Dimitrov, N. K., & Natarajan, A. (2017). Application of simulated lidar scanning

Examples

Installation

Hipersim is available as a registered package in the PyPi repository. Installation happens by calling pip install:

In [1]:
```
pip install hipersim==0.0.24
```

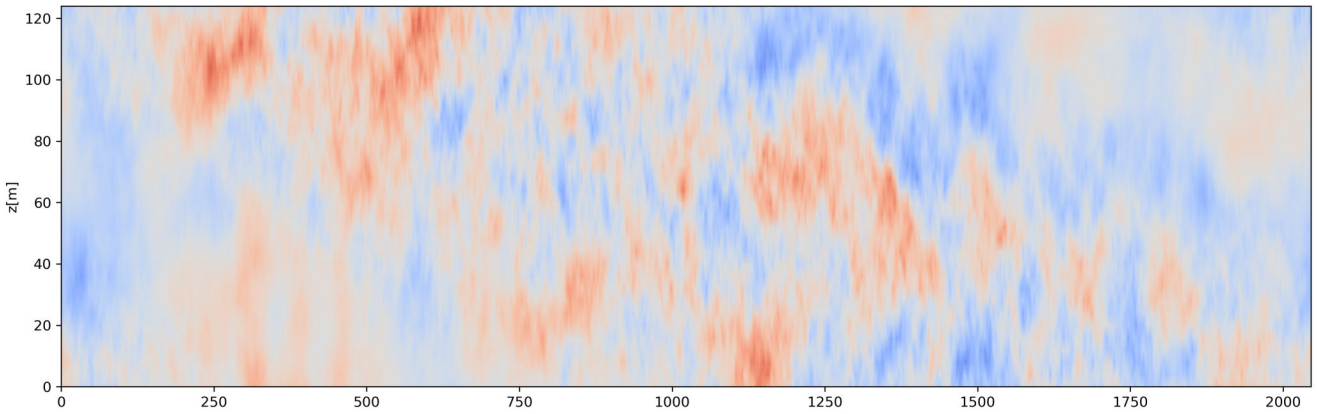
```
Collecting hipersim==0.0.24
  Downloading hipersim-0.0.24-py3-none-any.whl (255 kB)
Installing collected packages: hipersim
  Attempting uninstall: hipersim
    Found existing installation: hipersim 0.0.23
    Uninstalling hipersim-0.0.23:
      Successfully uninstalled hipersim-0.0.23
Successfully installed hipersim-0.0.24
Note: you may need to restart the kernel to use updated packages.
```

Example 1.1: generation of a turbulence box

A simple example of generating a single turbulence box with size 1024x32x32. The plot shows a vertical cross section of the turbulence box.

In [2]:
```
from hipersim.turbgen import turbgen
import numpy as np

# Input parameters (non-specified parameters equal their default values)
Nx = 1024
Ny = 32
Nz = 32
dx = 2
dy = 4
dz = 4

# Initialize field object
T = turbgen.turb_field(Nx = 1024, Ny = 32, Nz = 32, dx = 2, dy = 4, dz = 4, HighFreqComp = 1,SeedNo = 10)

# Generate turbulence field
u, v, w = T.generate()

# Plot a cross section of the resulting field
x = dx*np.arange(Nx)
y = dy*np.arange(Ny)
z = dz*np.arange(Nz)
xgrid,zgrid = np.meshgrid(x,z)
vmax = np.max((np.max(T.u),-np.min(T.u)))
vmin = -vmax
import matplotlib.pyplot as plt
norm = plt.Normalize(vmin=vmin, vmax = vmax)
fig = plt.figure(figsize = (16,5), dpi = 300)
plt.pcolormesh(xgrid,zgrid,T.u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
plt.xlabel('x[m]')
plt.ylabel('z[m]')
plt.show()
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 4.428203105926514
```



Example 1.2: generation of multiple turbulence boxes with identical Mann parameters but different seeds

Multiple turbulence boxes with identical parameters but different seeds can be created in a single function call, by simply specifying a list of seeds (e.g. `SeedNo = [1, 2, 3]`). Then all the resulting turbulence files will be saved on disk. The function will also return in memory the results from the last seed. This approach saves some computation time because the Fourier coefficients are generated only once (hence, the more seeds are used simultaneously, the bigger benefit).

In [3]:
```
from hipersim.turbgen import turbgen
import numpy as np

# Input parameters (non-specified parameters equal their default values)
Nx = 1024 # Note that Ny, Nz are assumed with default values (=32)
dx = 2
dy = 4
dz = 4

# Initialize field object
T = turbgen.turb_field(Nx = Nx, dx = dx, dy = dy, dz = dz, HighFreqComp = 0,SaveToFile = 1, SeedNo = [10,

# Generate turbulence field
u, v, w = T.generate()
```

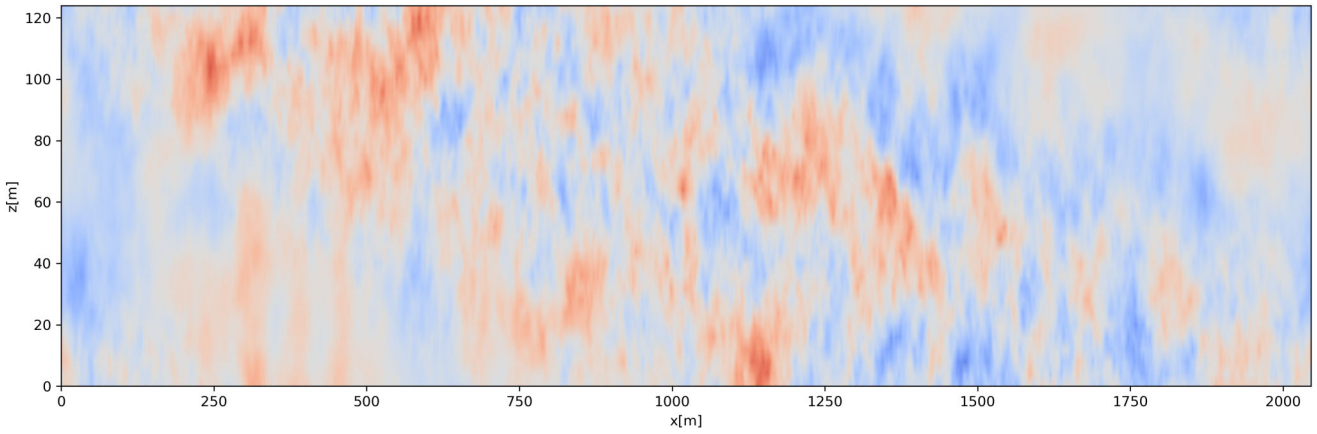
```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 4.065577507019043
Turbulence box generation complete.
Time elapsed is 6.058923721313477
Turbulence box generation complete.
Time elapsed is 8.037379741668701
```

Example 1.3: generation of a turbulence field by calling the generate_field() function

In [4]:

```
from hipersim.turbgen.generate_field import generate_field
import numpy as np

# Input parameters (non-specified parameters equal their default values)
Nx = 1024
dx = 2
dy = 4
dz = 4

# Generate turbulence field
u,v,w = generate_field(BaseName = 'Turb', \
                       alphaepsilon = 1,L = 30,Gamma = 3, \
                       SeedNo = 10, \
                       Nx = 1024,Ny = 32,Nz = 32,dx = 2,dy = 4,dz = 4, \
                       HighFreqComp = 0, SaveToFile = 0)

# Plot a cross section of the resulting field
x = dx*np.arange(Nx)
y = dy*np.arange(32)
z = dz*np.arange(32)
xgrid,zgrid = np.meshgrid(x,z)
vmax = np.max((np.max(u),-np.min(u)))
vmin = -vmax
import matplotlib.pyplot as plt
norm = plt.Normalize(vmin=vmin, vmax = vmax)
fig = plt.figure(figsize = (16,5), dpi = 300)
plt.pcolormesh(xgrid,zgrid,u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
plt.xlabel('x[m]')
plt.ylabel('z[m]')
plt.show()
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 4.146209001541138
```



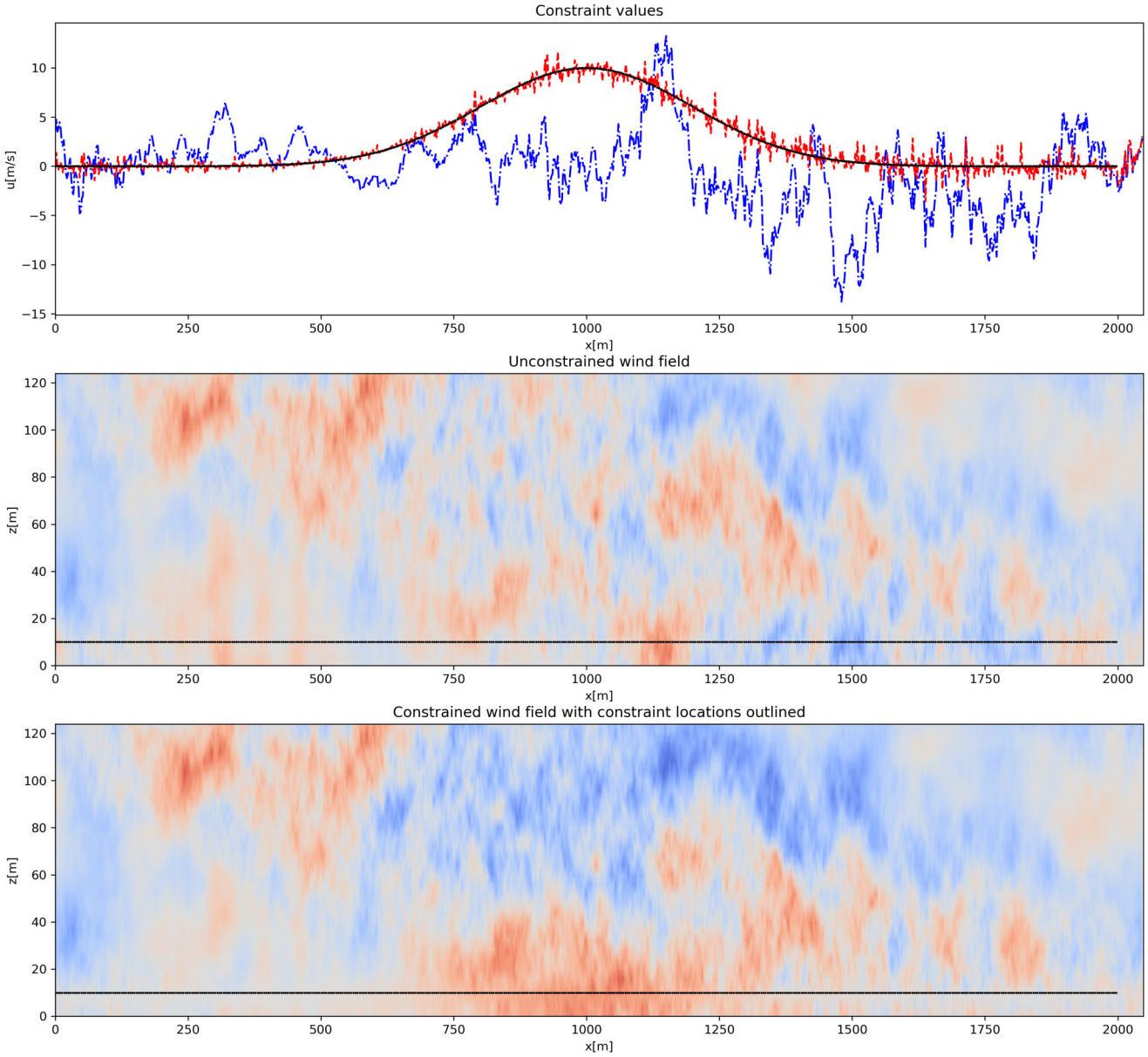
Example 2.1: generation of a turbulence box with constraints in all three components

In [5]:

```
from hipersim.turbgen import turbgen
import numpy as np

# Input parameters (non-specified parameters equal their default values)
Nx = 1024
dx = 2
dy = 4
dz = 4
T = turbgen.turb_field(Nx = Nx, dx = dx, dy = dy, dz = dz, HighFreqComp = 1,SaveToFile = 0, SeedNo = 10)

# Generate unconstrained turbulence field
u, v, w = T.generate()

# Specification of constraints array
Xconstraints = np.atleast_2d(np.arange(1,2000,4)).T
Yconstraints1 = 60*np.ones(Xconstraints.shape)
Zconstraints1 = 10*np.ones(Xconstraints.shape)
Yconstraints2 = 10*np.ones(Xconstraints.shape)
Zconstraints2 = 60*np.ones(Xconstraints.shape)
Uconstraints = 10*np.exp(-0.5*( (Xconstraints - 1000)/200)**2)
Vconstraints = -5*np.exp(-0.5*( (Xconstraints - 1000)/200)**2)
Wconstraints = -2*np.exp(-0.5*( (Xconstraints - 1000)/200)**2)
Constraints = np.concatenate([np.concatenate([Xconstraints, Xconstraints]),
                              np.concatenate([Yconstraints1, Yconstraints2]),
                              np.concatenate([Zconstraints1, Zconstraints2]),
                              np.concatenate([Uconstraints, -Uconstraints]),
                              np.concatenate([Vconstraints, -Vconstraints]),
                              np.concatenate([Wconstraints, -Wconstraints])],axis = 1)

# Plot initial results - the unconstrained field as well as the constraint specifications
x = dx*np.arange(Nx)
y = dy*np.arange(32)
z = dz*np.arange(32)
xgrid,zgrid = np.meshgrid(x,z)
vmax = np.max((np.max(T.u),-np.min(T.u)))
vmin = -vmax
import matplotlib.pyplot as plt
norm = plt.Normalize(vmin=vmin, vmax = vmax)

fig,(ax0,ax1,ax2) = plt.subplots(3,figsize = (16,15), dpi = 300)
ax0.plot(xgrid[15,:],T.u[:,15,2],'-.b')
ax0.set_title('Constraint values')
ax0.set_xlabel('x[m]')
ax0.set_ylabel('u[m/s]')
ax0.set_xlim((0,2048))

ax1.pcolormesh(xgrid,zgrid,T.u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
ax1.plot(Xconstraints,Zconstraints1,'ok', markersize = 0.75)
ax1.set_title('Unconstrained wind field')
ax1.set_xlabel('x[m]')
ax1.set_ylabel('z[m]')
ax1.set_xlim((0,2048))

# Apply constraints and return modified (constrained) wind field components
u, v, w = T.constrain(Constraints = Constraints)

# Plot constrained field
ax2.pcolormesh(xgrid,zgrid,T.u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
ax2.plot(Xconstraints,Zconstraints1,'ok', markersize = 0.75)
ax2.set_title('Constrained wind field with constraint locations outlined')
ax2.set_xlabel('x[m]')
ax2.set_ylabel('z[m]')
ax2.set_xlim((0,2048))

ax0.plot(xgrid[15,:],T.u[:,15,2],'--r')
ax0.plot(Xconstraints,Uconstraints,'ok',markersize = 1)

plt.show()

#fig.savefig('ConstrainedSimulationFieldExample.png', format = 'png', dpi = 96)
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 4.609789133071899
Computing Mann tensor / correlation arrays for constrained simulation:
Correlation computations complete
Time elapsed is 1.9051108360290527
Populating covariance matrix for the constraints:
Constraint-constraint covariance matrix has been assembled
```

```
Time elapsed is 0.21997332572937012
Applying constraints...
Constraint application complete
Time elapsed is 46.49681305885315
Constrained simulation complete
Total time elapsed is 48.62189722061157
```

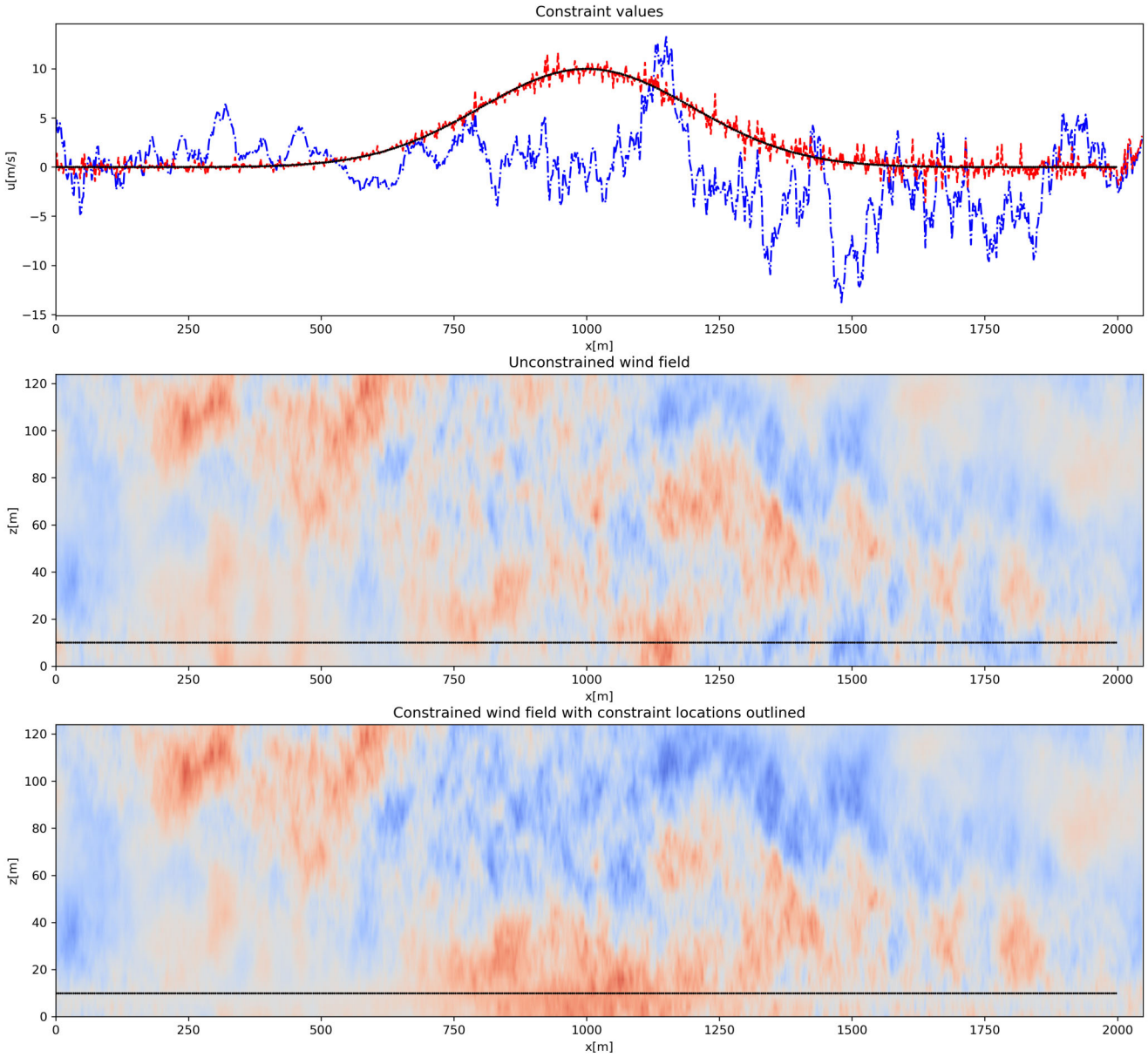


Example 2.2: generation of a turbulence box followed by constraining a single component

In [6]:

```
from hipersim.turbgen import turbgen
import numpy as np

# Input parameters (non-specified parameters equal their default values)
Nx = 1024
dx = 2
dy = 4
dz = 4
T = turbgen.turb_field(Nx = Nx, dx = dx, dy = dy, dz = dz, HighFreqComp = 1,SaveToFile = 0, SeedNo = 10)

# Generate unconstrained turbulence field
u, v, w = T.generate()

# Specification of constraints array
Xconstraints = np.atleast_2d(np.arange(1,2000,4)).T
Yconstraints1 = 60*np.ones(Xconstraints.shape)
Zconstraints1 = 10*np.ones(Xconstraints.shape)
Yconstraints2 = 10*np.ones(Xconstraints.shape)
Zconstraints2 = 60*np.ones(Xconstraints.shape)
Uconstraints = 10*np.exp(-0.5*( (Xconstraints - 1000)/200)**2)
Constraints = np.concatenate([np.concatenate([Xconstraints, Xconstraints]),
                              np.concatenate([Yconstraints1, Yconstraints2]),
                              np.concatenate([Zconstraints1, Zconstraints2]),
                              np.concatenate([Uconstraints, -Uconstraints])],axis = 1)

# Plot initial results - the unconstrained field as well as the constraint specifications
x = dx*np.arange(Nx)
y = dy*np.arange(32)
z = dz*np.arange(32)
xgrid,zgrid = np.meshgrid(x,z)
vmax = np.max((np.max(T.u),-np.min(T.u)))
vmin = -vmax
import matplotlib.pyplot as plt
norm = plt.Normalize(vmin=vmin, vmax = vmax)

fig,(ax0,ax1,ax2) = plt.subplots(3,figsize = (16,15), dpi = 300)
ax0.plot(xgrid[15,:],T.u[:,15,2],'-.b')
ax0.set_title('Constraint values')
ax0.set_xlabel('x[m]')
ax0.set_ylabel('u[m/s]')
ax0.set_xlim((0,2048))

ax1.pcolormesh(xgrid,zgrid,T.u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
ax1.plot(Xconstraints,Zconstraints1,'ok', markersize = 0.75)
ax1.set_title('Unconstrained wind field')
ax1.set_xlabel('x[m]')
ax1.set_ylabel('z[m]')
ax1.set_xlim((0,2048))

# Apply constraints and return modified (constrained) wind field components
u = T.constrain(Constraints = Constraints, Component = 'u')

# Plot constrained field
ax2.pcolormesh(xgrid,zgrid,T.u[:,15,:].T, cmap = 'coolwarm', shading = 'gouraud', norm = norm)
ax2.plot(Xconstraints,Zconstraints1,'ok', markersize = 0.75)
ax2.set_title('Constrained wind field with constraint locations outlined')
ax2.set_xlabel('x[m]')
ax2.set_ylabel('z[m]')
ax2.set_xlim((0,2048))

ax0.plot(xgrid[15,:],T.u[:,15,2],'--r')
ax0.plot(Xconstraints,Uconstraints,'ok',markersize = 1)

plt.show()
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 5.146637439727783
Computing Mann tensor / correlation arrays for constrained simulation:
Correlation computations complete
Time elapsed is 1.8590724468231201
Populating covariance matrix for the constraints:
Constraint-constraint covariance matrix has been assembled
Time elapsed is 0.08463597297668457
Applying constraints...
Constraint application complete
Time elapsed is 22.758671045303345
Constrained simulation complete
Total time elapsed is 24.70237946510315
```



Example 3.1: Generation of a turbulence box and saving in TurbSim (Bladed) format

In [12]:

```
from hipersim.turbgen import turbgen
import numpy as np

T = turbgen.turb_field(Nx = 600, Ny = 10, Nz = 10, dx = 5, dy = 5, dz = 5, SeedNo = 10,SaveToFile = 1, Ba
T.TurbOptions = {'FileFormat': 'wnd',
                 'Umean': 10,
                 'alpha': 0.07,
                 'TI_u': 0.08,
                 'TI_v': 0.064,
                 'TI_w': 0.04}
u,v,w = T.generate()

# NOTE THAT WITH THIS COMMAND IT IS ONLY THE SAVED FILE WHERE THE MEAN, SHEAR AND ROTATIONS ARE ADDED
# THE VARIABLES RETURNED ARE STILL WITH ZERO-MEAN:
print('Mean wind speed in turbulence box: ' + str(np.mean(u)))
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 0.8003644943237305
Mean wind speed in turbulence box: -0.18100096303239804
```

Example 3.2: Generation of a turbulence box with yaw offset, and saving it in Mann and TurbSim format

In [22]:

```
from hipersim.turbgen import turbgen
import numpy as np
T = turbgen.turb_field(Nx = 600, Ny = 10, Nz = 10, dx = 5, dy = 5, dz = 5, SeedNo = 10,SaveToFile = 0, Ba
u,v,w = T.generate()

# OUTPUT WITH MEAN FIELD CORRECTIONS IN HAWC2 FORMAT:
T.params['SaveToFile'] = 1
T.TurbOptions = {'FileFormat': 0,
                 'Umean': 10,
                 'alpha': 0.07,
                 'TI_u': 0.08,
                 'TI_v': 0.064,
                 'TI_w': 0.04,
                 'Yaw': 10}

u1,v1,w1 = T.output()

# UPDATE FILE FORMAT OPTIONS TO SAVE THE OUTPUT IN TURBSIM FORMAT:
T.TurbOptions['FileFormat'] = 'wnd'
u2,v2,w2 = T.output()

print('Mean wind speed in original turbulence box: ' + str(np.mean(u)))
print('Mean wind speed in modified turbulence box: ' + str(np.mean(u2)))
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 0.5740735530853271
Mean wind speed in original turbulence box: -0.18100096303239804
Mean wind speed in modified turbulence box: 8.822557068768837
```

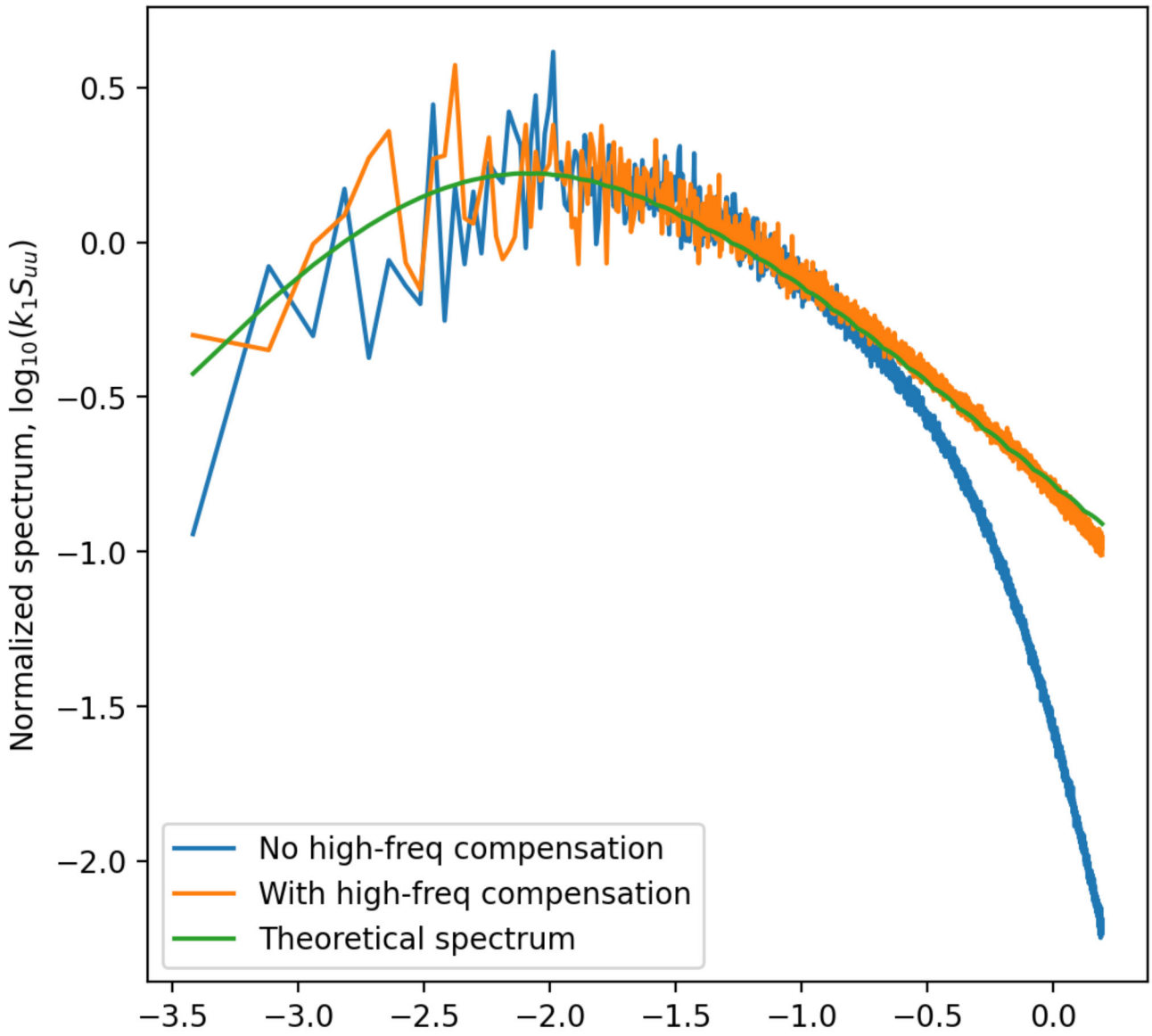
Example 4.1: Check of spectra of the generated turbulence boxes

The script below generates two turbulence boxes - one without high-frequency compensation, and a second one - with high-frequency compensation. The resulting u-spectrum is compared to the reference analytical spectrum with the same Mann model parameters.

In [10]:

```
from hipersim.turbgen.generate_field import generate_field
import numpy as np

Nx = 8192
Ny = 32
Nz = 32
dx = 2
dy = 4
dz = 4
L = 30
Gamma = 3
alphaepsilon = 1
SeedNo = 10
BaseName = 'turb'

[u,v,w] = generate_field(BaseName,alphaepsilon,L,Gamma,SeedNo,Nx,Ny,Nz,dx,dy,dz, \
                         HighFreqComp = 0, SaveToFile = 0)
[u1,v1,w1] = generate_field(BaseName,alphaepsilon,L,Gamma,SeedNo+1,Nx,Ny,Nz,dx,dy,dz, \
                         HighFreqComp = 1, SaveToFile = 0)

# SPECTRA OF THE GENERATED SERIES
L1 = Nx*dx
m1 = np.concatenate([np.arange(0,Nx/2),np.arange(-Nx/2,0)])
k1 = m1*2*np.pi/L1

SUensemble = np.zeros(int(Nx/2))
SUensemble1 = np.zeros(int(Nx/2))

for ik2 in range(Ny):
    for ik3 in range(Nz):
        Sui = np.fft.fft(u[:,ik2,ik3])
        Sui = (1/(np.sqrt(2)*Nx*dx))*np.abs(Sui)**2
        SUensemble+=Sui[:int(Nx/2)]

        Su1i = np.fft.fft(u1[:,ik2,ik3])
        Su1i = (1/(np.sqrt(2)*Nx*dx))*np.abs(Su1i)**2
        SUensemble1+=Su1i[:int(Nx/2)]


SUensemble = SUensemble/(Ny*Nz)
SUensemble1 = SUensemble1/(Ny*Nz)

ks1 = k1[:int(Nx/2)]

from hipersim.turbgen.mannspectrum import MannSpectrum_TableLookup
kref,Psiref = MannSpectrum_TableLookup(Gamma,L,alphaepsilon,ks1)

import matplotlib.pyplot as plt

fig,ax = plt.subplots(1,1,figsize = (6,6), dpi = 200)
ax.plot(np.log10(ks1), np.log10(ks1*SUensemble), label = 'No high-freq compensation')
ax.plot(np.log10(ks1), np.log10(ks1*SUensemble1), label = 'With high-freq compensation')
ax.plot(np.log10(kref),np.log10(kref*Psiref[0]), label =  'Theoretical spectrum')
ax.legend()
ax.set_xlabel('Wave number, $\log_{10}(k_1)$')
ax.set_ylabel('Normalized spectrum, $\log_{10}(k_1S_{uu})$')
plt.show()
```

```
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 24.407437086105347
Generating turbulence boxes..
Turbulence box generation complete.
Time elapsed is 28.874061584472656
C:\Users\nkdi\Anaconda3\envs\ScienceProgramming\lib\site-packages\hipersim\turbgen\mannspectrum.py:4571:
RuntimeWarning: divide by zero encountered in log10
  klog = np.log10(kinput)
C:\Users\nkdi\AppData\Local\Temp/ipykernel_15300/2180206836.py:51: RuntimeWarning: divide by zero encount
ered in log10
  ax.plot(np.log10(ks1), np.log10(ks1*SUensemble), label = 'No high-freq compensation')
C:\Users\nkdi\AppData\Local\Temp/ipykernel_15300/2180206836.py:52: RuntimeWarning: divide by zero encount
ered in log10
  ax.plot(np.log10(ks1), np.log10(ks1*SUensemble1), label = 'With high-freq compensation')
C:\Users\nkdi\AppData\Local\Temp/ipykernel_15300/2180206836.py:53: RuntimeWarning: divide by zero encount
ered in log10
  ax.plot(np.log10(kref),np.log10(kref*Psiref[0]), label =  'Theoretical spectrum')
```


0.5
0.0
−0.5
−1.0
−1.5
−2.0
Normalized spectrum, $\log_{10}(k_1 S_{uu})$
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
No high-freq compensation
With high-freq compensation
Theoretical spectrum

In [ ]: