2d Model Genbroek (waterschap Limburg)

Onderstaande Jupyter notebook bevat de code voor de automatische opbouw van het 2D model van de regenwaterbuffer doorbraak te Genbroek. In de code zijn mogelijkheden opgenomen voor het aanpassen van parameters die het overstroomde gebied en de snelheid van inundatie bepalen. De volgende parameters kunnen worden gevarieerd:

- het effect van de roosterresolutie
- het effect van onzekerheden in de ruwheid
- het effect van een bresgroei
- het effect van gridrotatie in de hoofdstroom richting

De resultaten van de berekeningen met variatie van deze parameters zijn opgenomen in de notebook 'Gevoeligheidsanalyse_Genbroek'.

De code voor de opbouw van het model is opgedeeld in logische stappen. Elke stap wordt ingeleid met een toelichting. De toelichting kan starten en eindigen met regels die beginnen met een *. Deze regels geven aan dat er een actie mogelijk of nodig is om instellingen te veranderen.

Inladen Python packages

In onderstaande stap worden de standaard python packages geladen die nodig zijn voor dit script.

In [1]:
```
import os
import sys

# Voeg pad vooraan toe waar 'dhydamo' module staat #Nodig wanner deze niet bij
de python install directory staat of overruled moet worden
sys.path.insert(0, 'D:')  #in dit voorbeeld staat de dhydamo map direct op de
 D schijf.

from dhydamo import DFlowFMModel, HyDAMO, Rectangular, DFlowFMWriter  #delft3d
fmpy
from dhydamo.core.geometry import as_polygon_list
from dhydamo.core.dfm import ObservationPoints
from dhydamo.io.dflowfmwriter import write_fm_file

import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Importeer gebied dmv. een shapefile*

De roosterresolutie wordt bij voorkeur zo klein mogelijk gekozen om veel detail weer te geven. Een fijnere resolutie betekent meer cellen en daarmee een grotere rekentijd. Het modelgebied is daarom bij voorkeur zo klein mogelijk en bevat het overstroomde gebied met niet al teveel ruimte daaromheen.

De begrenzing van het modelgebied is gemaakt in QGIS en iteratief verkleind op basis van verkennende simulaties. Na het runnen van een model op 5 a 10 meter kan er een beeld gevormd worden van het overstroomde gebied. Aan de hand van deze resultaten kan het gebied iteratief verkleind worden en vervolgens met een kleinere resolutie opnieuw gerund worden. Hierbij is het belangrijk om goed te controleren of er met een kleinere resolutie en dus meer detail in onder andere het hoogtemodel het water nog binnen het gebied blijft of dat het modelgebied op sommige locaties weer iets vergroot moet worden.

*Definieer de locatie van het gebieds-extent als shapefile bij de variabele: 'mask_file'.

In [2]:
```
# modeldir = r"d:\PR\PR4139.10"
mask_file = 'GIS_Genbroek/test_extend.shp'

clipgeometry = gpd.read_file(mask_file).at[0, 'geometry']
```

1D deel

Importeer duikers*

Duikers kunnen nog niet als 2D object toegevoegd worden en zijn daarom als 1D objecten toegevoegd. De onderstaande code gebruikt als invoer een shapefile voor de duikers die niet conform HYDAMO is. De afwijkende kolomnamen zijn in de onderstaande code vervangen door de HYDAMO kolom namen.

*Laad 1D duikers in als shapefile (bij 'path_culverts') en pas de attributen/kolomnamen aan indien niet conform HYDAMO.

In [3]:
```
path_culverts = 'GIS_Genbroek/wl_duikers.shp'
culverts = gpd.read_file(path_culverts)
# print(culverts.columns)

mapping = columns={
    'CODE': 'code',
    'HOOGTEOPEN': 'hoogteopening',
    'BREEDTEOPE': 'breedteopening',
    'HOOGTEBINN': 'hoogtebinnenonderkantbenedenstrooms',
    'HOOGTEBI_1': 'hoogtebinnenonderkantbovenstrooms',
    'LENGTE': 'lengte',
    'VORMKOKER': 'vormcode'
}

culverts.rename(columns=mapping, inplace=True)
culverts = culverts[list(mapping.values())+['geometry']]
culverts['ruwheidswaarde'] = 75
culverts['ruwheidstypecode'] = 4
culverts['intreeverlies'] = 1.0
culverts['uittreeverlies'] = 1.0
culverts['lengte'] = culverts.length

culverts.at[0, 'breedteopening'] = 1.0
```

Clip 1d objecten uit de HYDAMO data op het gebied*

De duikers die binnen het modelgebied liggen worden als 1D object toegevoegd. In dit 2D model zijn geen waterlopen aanwezig. Echter als er 1D objecten gebruikt worden moeten deze aan een waterloop gekoppeld worden. De ingeladen duikerlijnen worden nu zowel gebruikt als kunstmatige waterloop en als duiker aangezien er verder geen waterlopen aanwezig zijn.

*Vul de lijst hieronder aan indien meer 1D objecten aanwezig zijn dan duikers.

In [4]:
```
# Parse HyDAMO model within extent file
hydamo = HyDAMO(extent_file=mask_file)
hydamo.branches.set_data(culverts, index_col='code')
hydamo.culverts.set_data(culverts, index_col='code')
hydamo.culverts.snap_to_branch(hydamo.branches, snap_method='ends')
```

Voeg de 1d schematizatie toe aan het model

In [5]:
```
# # Add to schematisation
dfmmodel = DFlowFMModel()
dfmmodel.structures.io.culverts_from_hydamo(hydamo.culverts)
```

Voeg cross-secties toe aan de takken*

De duikers zijn nu ingeladen op de 1D waterlopen, maar er is nog geen profiel aan de waterlopen toegekend. Onderstaande code voegt een standaardprofiel toe met een bodemhoogte van 80 meter. In de code achter # is ook een wijze opgenomen om standaardprofielen op een andere wijze op te nemen. Dit werkte echter niet met deze versie van de Interacter.

*De 'one_d_mesh_distance' kan worden aangepast om de afstand tussen de 1D/2D koppelingen aan te passen. In dit geval zijn de culverts de enige 1D objecten, en hoeft dit niet veranderd te worden.

In [6]:
```
dfmmodel.mdu_parameters['BedlevUni'] = 80

# Also add cross sections to branches

# name = dfmmodel.crosssections.add_rectangle_definition(
#     height=5, width=5, closed=False, roughnesstype='Strickler', roughnessval
ue=75)

# for culvert in hydamo.culverts.itertuples():
#     for dist in [0.1 * culvert.lengte, 0.9 * culvert.lengte]:
#         dfmmodel.crosssections.add_crosssection_location(
#             branchid=culvert.branch_id,
#             chainage=dist,
#             definition=name,
#             shift=min(culvert.hoogtebinnenonderkantbenedenstrooms, culvert.h
oogtebinnenonderkantbovenstrooms)-1.0
#         )

dfmmodel.network.set_branches(hydamo.branches)
dfmmodel.network.generate_1dnetwork(one_d_mesh_distance=40.0, seperate_structu
res=False)
```

2D deel

Genereer een vierkant 2D rooster binnen het modelgebied.

Celgrootte*

Op dit moment wordt geadviseerd het aantal cellen in een rooster te beperken tot maximaal 1 miljoen. Dit kan hieronder worden gecontroleerd als de celgrootte wordt ingevoerd.

*Geef de cellsize op en krijg als feedback het aantal gridcellen terug voor de combinatie modelgebied en celgrootte.

In [7]:
```
#Verander cellsize naar het gewenste detailniveau
cellsize = 2
print('Het ingelezen gebied heeft een oppervlakte van: ',int(clipgeometry.area
), 'm2') #
print('Het aantal gridcellen met het huidig extent en cellsize is: ', int(clip
geometry.area/(cellsize*cellsize)), 'cellen') #
```

```
Het ingelezen gebied heeft een oppervlakte van:  557276 m2
Het aantal gridcellen met het huidig extent en cellsize is:  139319 cellen
```

Grid generatie en rotatie*

Het kan voor het model gunstig zijn om het grid te draaien in de hoofdstroomrichting.

*Als een grid rotatie is gewenst kan dit hieronder bij 'rotation' opgegeven worden in graden. LET OP: gridrotatie en lokaal verfijnen in de volgende stap kan nog niet gecombineerd worden.

In [8]:
```
# Genereer grid binnen gebiedsgrenzen
mesh = Rectangular()
rotation = 0 #in graden
mesh.generate_within_polygon(clipgeometry, cellsize=cellsize, rotation=rotatio
n)
```

Lokaal verfijnen*

Voor lokaal verfijnen naar een kleinere resolutie kan het script hieronder worden gebruikt. Op de eerste regel kan een shapefile met vlakken worden ingelezen waarvoor de verfijning wordt toegepast. Op de onderste regel wordt deze shape aangeroepen en kan een niveau ('level') van verfijning worden ingevoerd. Level = [1] betekent dat de cellsize 1 keer gehalveerd wordt.

*Om dit script te gebruiken kunnen de hashtags (#) op regel 1 en 4 weggehaald worden-> Selecteer alles in het blok en gebruik CTRL ?

In [9]:
```
# verfijning_shape = gpd.read_file('../GIS/Verfijning_shape_straat.shp').at[0,
'geometry']

# # Refine
# mesh.refine(polygon=[verfijning_shape], level=[1], cellsize=cellsize)
```

Hoogtemodel*

Het hoogtemodel wordt opgebouwd op basis van het AHN3. Het opbouwen van het hoogtemodel is gedaan in een apart python script. De stappen kunnen ook in QGIS worden doorlopen:

```
1) Merge eventuele losse AHN3 rasters tot 1 raster bestand.

2) Vul gaten (bomen, watergangen en huizen) a.d.h.v. de omliggende cellen.

3) Verhoog de huizen en schuren met 3 meter. Dit kan gedaan worden door de BAG en d
e bgt_overigebebouwing als maskfile te gebruiken voor de ophoging.

4) Verwijder de regenwaterbuffer dam die verwacht wordt door te breken uit de DEM e
n vul op met de hoogtewaarde op de voet van de dam. Hiermee maken we de bresgroei m
ogelijk via een lijnelement.
```

De hoogtekaart wordt ingeladen aan de hand van het met python gegenereerde of in Qgis bewerkte DEM raster bestand met de functie "altitude_from_raster".

De hoogtekaart wordt ingeladen aan de hand van het met python gegenereerde of in QGIS bewerkte DEM raster bestand met de functie "altitude_from_raster".

Bij de huidige versie kwamen er problemen bij het verfijnen naar 0.5 meter met een DEM-bestand op 0.5 meter resolutie. In dit script is de DEM daarom naar 0.25 m geresampled. Dit kan op regel 10 door een ander hoogtemodel op te geven.

*Hieronder moet het hoogtemodel dat gebruikt wordt om hoogtes aan elke cel toe te kennen opgegeven (vul path in bij regel 12).

In [10]:
```
BedlevType = 1 # gebruik setting 3 voor analytisch grid.
if BedlevType == 1:
    facenode =  'face'
    Conveyance2D = -1
if BedlevType == 3:
    facenode =  'node'
    Conveyance2D = 3

# #for 0.5m refinement -> file is resampled to 0.25m res
# mesh.altitude_from_raster('../DEM/ahn_merged_genbroek_interpolated_panden+3m
_noDam_interpol_025m.tif', where= facenode, stat='mean')# 'node' voor optie he
rman

mesh.altitude_from_raster('GIS_Genbroek/ahn_merged_genbroek_interpolated_pande
n+3m_noDam_interpol.tif', where= facenode, stat='mean')# 'node' voor optie her
man
dfmmodel.mdu_parameters['BedlevType'] = BedlevType
dfmmodel.mdu_parameters['Conveyance2D'] = Conveyance2D

# Voor analytisch subgrid, kies where='node' en bedlevtype=3
# Voor een duidelijker hoogteverschil bij huizen, kies stat='median'
print(facenode)
```

```
face
```

Voeg het 2d mesh toe aan de schematisatie.

In [11]:
```
dfmmodel.network.add_mesh2d(mesh)
```

1D en 2D

Koppel elk 1d punt aan het dichtstbijzijnde 2d punt.

Gebruik de optie 2D naar 1D. Eerst wordt voor elke 2D punt het dichtstbijzijnde 1D punt binnen een bepaalde afstand gezocht (in dit geval cellsize + 1 m), waarna alle linkjes die een andere cel kruisen worden verwijderd.

In [12]:
```
del dfmmodel.network.links1d2d.faces2d[:]
del dfmmodel.network.links1d2d.nodes1d[:]

dfmmodel.network.links1d2d.generate_2d_to_1d(max_distance=cellsize+1, intersec
ting=False)
```

```
Removing links crossing mult. cells: 100%|██████████| 30/30 [00:00<00:00, 556.05it/s]
```

Inladen regenwaterbuffer*

Code voor het inladen van de regenwaterbuffer polygon dat met een gekozen waterstand gevuld kan worden.

*Geef hier de shapefile locatie op en het waterlevel tov. NAP.

In [13]:
```
rainbuffer_pol = gpd.read_file('GIS_Genbroek/buffer_genbroek.shp').at[0, 'geom
etry']
dfmmodel.external_forcings.set_initial_waterlevel(83.3, rainbuffer_pol)
```

Toevoegen van observatiepunten*

Observatiepunten kunnen hieronder worden toegevoegd via een punt-shapefile met een kolomnaam 'id'. Als de kolomnaam anders is dan 'id', kan dit hieronder bij obs_points.id worden veranderd.

*Als geen observatiepunten gewenst zijn, maak dan commentaar van de code hieronder (selecteer alles, vervolgens CTRL ?)

In [14]:
```
obs_points = gpd.read_file('GIS_Genbroek/observatie_punten.shp')
# print(list(obs_points.geometry)[:])
# print(list(obs_points.id))


dfmmodel.observation_points.add_points(
    crds = list(obs_points.geometry),
    names = list(obs_points.id),
    snap_to_1d = False
)
```

Model wegschrijven*

Creëer een writer en schrijf het model naar bestanden in D-HYDRO formaat. Het grootste deel wordt weggeschreven door de functie "write_all". De functie "objects_to_ldb" schrijft de symbolen voor de duikers, overlaten, pompen en doorsnedes zodat deze in de Interacter kunnen worden gevisualiseerd.

Via de MapInterval wordt opgegeven met welke interval (s) de rasters moeten worden weggeschreven. Via de hisInterval wordt opgegeven met welke interval (s) resultaten voor observatiepunten worden weggeschreven.
*Autostart 0= start niet vanzelf, 1= model start maar sluit niet vanzelf of 2=tsart en sluit wanneer klaar.

*Wanneer lijnelementen zijn toegevoegd om wegen, spoorwegen of verhogingen beter te modelleren moet regel 13 FixedWeirFile blijven staan, anders een # ervoor.

*Modelmapname geeft de output map naam aan voor het model.

*Bij regel 28 kan de modelnaam worden opgegeven.

In [15]:

```
dfmmodel.mdu_parameters['refdate'] = 20000101
dfmmodel.mdu_parameters['tstart'] = 0.0 * 3600
dfmmodel.mdu_parameters['tstop'] = 24.0 * 1 * 3600
dfmmodel.mdu_parameters['hisinterval'] = '120. 0. 0.'
dfmmodel.mdu_parameters['MapInterval'] = '600. 0. 0.'
dfmmodel.mdu_parameters['RstInterval'] = '0. 0. 0.'

dfmmodel.mdu_parameters['AutoStart'] = '1'

dfmmodel.mdu_parameters['UnifFrictCoef'] = 0.5  #(aanname fritiecoef van 0.5,
 bij missing data)
dfmmodel.mdu_parameters['UnifFrictType'] = 2 #(0: Chezy, 1: Manning, 2: White-
Colebrook, 3: idem, WAQUA style)

dfmmodel.mdu_parameters['FixedWeirFile'] = 'lijnelementen.pliz' #Verwijder wan
neer niet van toepassing

dfmmodel.mdu_parameters['CFLMax'] = 4.0  # Maximum Courant number, standaard o
p 0.7

# Only write relevant data
for ext in [
    'numlimdt', 'taucurrent', 'chezy', 'turbulence', 'tidal_potential', 'spira
l_flow',
    'wind', 'temperature', 'salinity', 'waterlevel_s0', 'velocity_component_u
0', 'velocity_component_u1',
    'upward_velocity_component', 'density_rho', 'horizontal_viscosity_viu', 'h
orizontal_diffusivity_diu'
]:
    dfmmodel.mdu_parameters['Wrimap_'+ext] = 0


modelmapname = "model_"+str(cellsize)+"m_rota"+str(rotation)+'bedlev'+str(Bedl
evType)+'_l'
#print(modelmapname)
writer = DFlowFMWriter(dfmmodel, output_dir=f'../{modelmapname}', name='buffe
r'+str(cellsize)+"m_r"+str(rotation)+'_b'+str(BedlevType)+'refinedv4_025')

# Write as model
writer.objects_to_ldb()
writer.write_all()
```

Lijnelementen vaste stuwen

Schrijf lijnelementen bestand voor fixedweirs* Hier kunnen lijnelementen worden toegevoegd die als vaste stuw zullen werken.

*Geef locatie op van shapefile die de lijnelementen bevat. Er moet een 'id' en een 'hoogte' kolom mee worden gegeven.

*vergeet niet bij het blok hierboven om de volgende regel aan te hebben (de regel zonder #):

```
dfmmodel.mdu_parameters['FixedWeirFile'] = 'lijnelementen.pliz'
```

In [16]:
```
#fixedweri shapefile is een lijn bestand met een id en een hoogte kolom
lijnelementen = gpd.read_file('GIS_Genbroek/lijnelementen3.shp')

write_fm_file(
    file=os.path.join(writer.output_dir,  'lijnelementen.pliz'),
    data=[np.c_[np.vstack(row.geometry.coords[:])[:, :2], np.ones(len(row.geom
etry.coords)) * row.hoogte] for row in lijnelementen.itertuples()],
    names = lijnelementen.index.tolist(),
    mode='a'
)
```

Bresgroei

Schrijf lijnelement voor bresgroei. Zelfde verhaal als voor de lijnelementen vaste stuwen.

*Uncomment als bresgroei nodig is (selecteer alles en dan ctrl ?).

*Geef breslijn via shapefile op

In [17]:
```
# write pli and pliz file
# #bres shapefile is een lijn bestand met een id en een hoogte kolom
# breslijnelementen = gpd.read_file('../GIS/bres_lijn.shp')
# name_dambreakpli = 'bres.pli'

# #schrijf .pli file waarover de bres kan groeien (slaat alleen x,y coords op)
# write_fm_file(
#     file=os.path.join(writer.output_dir, name_dambreakpli),
#     data=[np.c_[np.vstack(row.geometry.coords[:])[:, :2]] for row in breslij
nelementen.itertuples()],
#     names = breslijnelementen.index.tolist(),
#     mode='a'
# )

# # Voeg lijn waarover de bres kan groeien toe als weir, dit wordt gebruikt vo
ordat de bres gaat groeien
# # Gebruik hoogte in de regel data ipv row.hoogte wanneer een andere damhoogt
e gebruikt wordt dan in de bres_lijn shapefile
# # Indien handmatig hoogte gebruikt wordt uncomment regel hieronder met [Ctrl
?] of door de # weg te halen
# #hoogte = 83.3

# #schrijf/append .pliz waarde bres voor de doorbraak als weir fungeerd (slaat
alleen x,y coords en hoogte op)
# write_fm_file(
#     file=os.path.join(writer.output_dir,  'lijnelementen.pliz'),
#     data=[np.c_[np.vstack(row.geometry.coords[:])[:, :2], np.ones(len(row.ge
ometry.coords)) * row.hoogte] for row in breslijnelementen.itertuples()],
#     names = breslijnelementen.index.tolist(),
#     mode='a'
# )
```

Ruwheden toevoegen*

In python (maar dit kan ook via QGIS) zijn diverse bronnen gebruikt om een landclasses map te maken. Aan de hand van de landclasses zijn de ruwheden bepaald. De brondata bestaat uit de BAG, de BGT en de BRP. Als de brondata overlappen, zijn prioriteiten voor aangehouden, voor het al dan niet meenemen van de lagen. Dit is beschreven in de ruwheden_per_class.xlsx

*Geef shapefile op met ruwheden in het gebied

In [18]:
```
# Toevoegen ruwheid
# Laad ruwheden in
file = "GIS_Genbroek/landclass_merged_met_ruwheden.shp"
gdf = gpd.read_file(file)
# Clip
idx = gdf.intersects(hydamo.clipgeo)
gdf = gdf.loc[idx]
```

Ruwheden groeperen

Voeg polygonen met dezelfde ruwheid samen om inlaadtijd te besparen. Dit creëert per ruwheidswaarde een apart .pol bestand dat door het model wordt ingelezen.

In [19]:
```
polygonen = {}
for ruwheid, group in gdf.groupby('Nikuradse'):
    polygonen[ruwheid] = as_polygon_list(group.unary_union)
    #polygonen[ruwheid*2] = as_polygon_list(group.unary_union) #Dit is een voo
rbeeld regel om de ruwheid aan te passen met een factor
template = """
QUANTITY=frictioncoefficient
FILENAME={filename}
FILETYPE=10
METHOD=4
OPERAND=O
VALUE={value}
"""

i = 1
with open(writer.extfile, 'a') as f:
    for ruwheid, polygonlist in polygonen.items():
        file = f'roughness/ruwheid_nikuradse_{ruwheid:05.2f}.pol'
        f.write(template.format(filename=file, value=ruwheid))
        write_fm_file(
            file=os.path.join(writer.output_dir, file),
            data=[np.vstack(polygon.simplify(0.25).exterior.coords[:])[:, :2]
for polygon in polygonlist],
            names=list(range(1, len(polygonlist)+1))
        )
```

Visualisatie stap

Voeg de polygonen met ruwheid 10 (bebouwing) en rijbanen toe als achtergrondelement in het model. Dit maakt het overstromingspatroon wat inzichtelijker. Deze stap is puur visueel.

In [20]:
```
selectie = gdf.loc[(gdf['Nikuradse'].eq(10.0) | gdf['class'].eq('rijbaan lokal
e weg') | gdf['class'].eq('rijbaan autosnelweg'))].copy()
selectie['geometry'] = selectie['geometry'].simplify(1.0)

write_fm_file(
    file=os.path.join(writer.output_dir,  'structures.ldb'),
    data=[np.vstack(row.geometry.exterior.coords[:])[:, :2] for row in selecti
e.itertuples()],
    names = selectie.index.tolist(),
    mode='a'
)
```

In [ ]: