S-Grid is implemented in Python and C++. Most of the Python functionality is in the sgrid.py module, with further utilities in fileIO.py, bresenhamalgorithm.py and reservoirs.py; these should never need to be edited by the user. The main processing engine is in the sgridHydraulics.dll library (sgridHydraulics.so for Linux systems).

The buildModel.py, runModel.py and postProcessModel.py scripts perform model build, run and post process and should be the only scripts that the user needs to edit.

4 Test models are provided with the installation, modelling flow down a planar slope, tidal flow on a planar beach, flow in a meandering channel, and rainfall-runoff modelling in the Thames catchment. These can be used as a starting point for trying S-Grid before building other models.

**Model build**

This step takes a DTM and generates a set of conveyance and storage parameters on a square grid for use in the model run stage. This can be quite computationally intensive, especially for high resolution DTMs, but should only need to be run once for each DTM. It may need to be rerun if the base data changes (e.g. different DTM, different Manning’s *n*).

All the most commonly user edited variables are in the section marked # CONTROL PANEL at the start of the file. These are:

|  |  |  |
| --- | --- | --- |
| sgridPath | Path to where the sgrid.py, sgridHydraulics.dll etc files are stored. This allows you to have multiple folders using the same sgrid.py etc files. Use “./” if they are in the same folder as buildModel.py. | Required |
| dtmFileName | Path to the DTM file, can be in any GDAL recognised format (e.g. \*.tif, \*.asc) | Required |
| clipPolyFileName | Path to a polygon shapefile which can be used to remove DTM areas outside the polygons; this is useful to clip a DTM to a catchment outline and remove other areas | Set to None if not required |
| useTempTopoFile | Set to True to create a temporary tiled, uncompressed topography file foe buildModel.py to use; this can help to speed up the process for large files (especially if they’re compressed). | Set to False if not required |
| noDataValue and noDataReplacement | Used to recognise a no data value in the DTM, and change to -9999 (which is recognised by S-Grid) | Set both to None if not required |
| addNullEdges | This adds a set of NULL cells around the edge of the model domain; these allow water to flow out (using a critical depth boundary condition) | Set to False if not required |
| xll and yll | Coordinates of lower left corner of grid | Required |
| cellSize | Size of grid cells | Required |
| xsz and ysz | Number of cells in grid in x and y directions | Required |
| nFloodplain | Uniform roughness value | Required, but ignored if nFloodplainFile present |
| nFloodplainFile | Raster of Manning’s *n* values with same extent and resolution as DTM | Set to None if not required |
| outputFile | Path to file for storing the conveyance and storage parameters (this uses a pickle format) | Required |
| gridFileName | Path to file which outputs the grid of cells in CSV/WKT format; this isn’t actually used by the model, but is useful for visualising the grid and checking extents | Not required |

To build a model, edit the buildModel.py (it can be saved with a different name) script and run. The outputs will be:

* grid.csv/grid.csvt (or as specified by the gridFileName parameter), which can be loaded into GIS to show the location of model grid cells
* params.pck (or as specified by the outputFile parameter), the file storing all the conveyance and storage parameters for each cell, to be used by runModel.py
* conveyanceParams.csv/storageParams.csv, storing information about conveyance between and storage within cells, which can be loaded into GIS for user inspection

**Model run**

This step runs the model, based on the parameters calculated in the model build stage. All the most commonly user edited variables are in the section marked # CONTROL PANEL. These are:

|  |  |  |
| --- | --- | --- |
| sgridPath | As for buildModel.py. | Required |
| parametersFile | Parameter file generated in the build model stage | Required |
| outputDirectory | Path to a folder where results will be stored | Required |
| outputPrefix | Prefix used to generate output file names (change this to store results in the same folder without overwriting other models) | Required |
| rainfallDuration | Duration of direct rainfall in hours | Set to 0 if no rainfall required |
| rainfallStart | Start of rainfall in hours; can be used to provide a run in period for the model to settle down before rainfall starts | Required |
| duration | Duration of model simulation in hours | Required |
| rainfallDepth | Depth of rainfall in mm; this is spread uniformly over the rainfall duration | Set to 0 if no rainfall required |
| initialWlFile | Initial water level CSV file generated by a previous S-Grid run; can be used as initial conditions or to continue a previous run | Set to None if not required |
| pcRunoff | Percentage runoff used to calculate runoff from rainfall | Required |
| saveMax | Saves maximum water levels from run if True | Required (True/False) |
| SaveEnd | Saves final water levels from run if True | Required (True/False) |
| flowPointShp | Path to point shapefile where flows are to be introduced | Set to None if not required |
| flowAttr | Attribute in shapefile which holds the flow values | Set to None if not required |
| flowMultiplier | Multiplies all flow point values; easy way to adjust all flows | Set to None if not required |
| baseFlow | Flow introduced into all cells in m3s-1 per cell; useful for representing groundwater flows | Set to zero if not required |
| wlShp | Path to polyline shapefile of water level boundary conditions | Set to None if not required |
| wlAttr | Attribute in shapefile which holds the water level values | Set to None if not required |
| initialWL | Initial water level applied to all cells; cells are dry if lowest point is above this level | Set to None if not required |
| initialTimeStep | Starting time step in seconds; time step is adaptive throughout rest of the simulation | Required |
| minTimeStep | Minimum time step in seconds | Required |
| maxTimeStep | Maximum time step in seconds | Required |
| reservoirShpFileName | Polyline shapefile with location of dam crests; the direction of these lines is important, the reservoir is to the right of the line. Attributes *cl* gives the crest elevation, *wc* the weir coefficient. | Set to None if not required |
| spillwayShpFileName | Point shapefile with locations of spillways draining through the dam crest. Attributes are *cl* for crest elevation, *width* for width and *wc* for weir coefficient. | Set to None if not required |
| conveyanceParameterFileName | Name of CSV file for storing modified conveyance parameters, for diagnostics only. | Set to None if not required |

Outside of the control panel, there are other more advanced edits that can be made in the locations marked by the following comments:

|  |  |
| --- | --- |
| # Add rainfall | Code to modify the rainfall and runoff spatially or temporally can be added here |
| # Apply water level boundary | Add code to modify wlGrid[wlPt[0],wlPt[1]] e.g. to represent hydrograph |
| # Modify flow boundary | Add code to modify flowPointsQ array values e.g. to represent hydrograph; flowPointsQ is a numpy array with a value for each model cell where a flow is applied. |

To run a model, edit the runModel.py script and run.

On screen outputs while running are: simulated time, timestep, volume in domain, active volume (i.e. in cells with depth>0.1m), rainfall volume, number of drying corrections (smaller number is better), number of active cells (depth>01.m), flow in, flow out, and projected simulation end time.

Outputs from runModel.py are in the form of CSV files of water level for each cell, and flow between cells.

**Post process**

This step takes the CSV files output by runModel.py and generates a set of rasters of depths and water levels at the same resolution as the input DTM.

All the most commonly user edited variables are in the section marked # CONTROL PANEL. These are:

|  |  |  |
| --- | --- | --- |
| parametersFile | Parameter file generated in the build model stage | Required |
| dtmFileName | Path to the DTM file, can be in any GDAL recognised format; does not have to be same resolution as used in buildModel | Required |
| useTempTopoFile | Set to True to create a temporary tiled, uncompressed topography file foe buildModel.py to use; this can help to speed up the process for large files (especially if they’re compressed). | Set to False if not required |
| noDataValue and noDataReplacement | Used to recognise a no data value in the DTM, and change to -9999 (which is recognised by S-Grid) | Set both to None if not required |
| resultsDirectory | Folder where CSV outputs from runModel are stored | Required |
| resultsPrefix | Prefix of runModel outputs | Required |
| processMax | Process maximum water levels from run if True | Required (True/False) |
| processEnd | Process final water levels from run if True | Required (True/False) |
| defaultDepth | Depth burned into flowpaths | Required |
| flowThreshold | Value of flow between cells in m3s-1 above which a flow is significant and water levels are interpolated between cells, otherwise a horizontal surface is used; this can be adjusted to represent water levels and depths in small catchments better | Required |
| reservoirsPresent | Set to True to indicate reservoirs are present and represent these in the post processed outputs | Required (True/False) |

To make processing of the outputs manageable they are split into tiles, all tiles are joined in a vrt. The \_depth and \_wl are the raw depth and water level outputs; \_lfp is the output of a lazy flow path algorithm which aims to link up disconnected flood areas based on the most likely flow path between them; and \_merge is a combination of the \_depth and \_lfp outputs.