Table of Contents

The **GeneralGeneletModel.py** package (imported as GGM in accompanying scripts) includes a GeneletNetwork class that for initializing, simulating, and compiling sequences for genelet networks. The general genelet model functions at the beginning of **GeneralGeneletModel.py** are helper functions for the GeneletNetwork class. Below are the descriptions for the initialization of GeneletNetwork and how to use its associated functions.

1. Examples

Supplementary Section 5 of the paper describing the general genelet model describes the model in detail. Supplementary Figure 35 shows an example of how the vectors and matrices are defined for an incoherent feedforward pulse network.

The General Genelet model github also has a number of examples showing how to set up the model for a number of network topologies and run network simulations under a range of conditions:

Figure3_IFFL_simulations.py: simulates an incoherent feedforward pulse network with three different activator concentrations for the pulsing node
Figure3_IFFL_1_2_simulation.py: simulates a network with two incoherent feedforward pulse modules connected in series.
Figure4_TSN_stable_states.py: simulates a mutually repressive tri-stable network initialized in each of its three stable states
Figure4_TSN_single_state_changes.py: simulates a mutually repressive tri-stable network initialized in state 1 and then induced to
switch to another stable state by the addition of an inducer RNA. All state changes are identical in the model so only one is shown.
Figure5_I_IFFL12_FB1.py: simulates a mutually repressive bi-stable network with each state coupled to an incoherent feedforward loop. One of the feedforward loops feeds back to induce a state change in the bi-stable network.
supp_figure31_faster_sd_rates_vs_higher_genelet_con.py: simulates coactivation and repression with 10x and 100x higher strand displacement rate constants or with 5x and 10x higher genelet concentrations.
supp_figure44_IFFL1_2leak.py: simulates a network with two incoherent feedforward pulse modules connected in series with some of the nodes exhibiting a 5% transcription leak in their blocked state.
supp_figure56_TSN_repressor_state_changes.py: simulates a mutually repressive tri-stable network that is initialized in state 1 and then the repressor of state 1 is added in an attempt to switch states. This turns the state 1 nodes off but cannot direct the network to either other state.
supp_figure67_ring_oscillators.py: simulates 3-, 5-, and 7-node ring oscillators. Higher transcription and RNA degradation rate constants are used.
supp_figure68_feedback_oscillators.py: simulates 2-, 3-, 4- and 5-node feedback oscillators. Higher transcription and RNA degradation rate constants are used.
supp_figure69_or_and_persist.py: simulates OR and AND logic elements using coactivation with varied blocker concentrations. The AND logic element is used to build a persistence detector.
supp_figure70_not_nor_nand.py: simulates NOT, NOR, and NAND logic elements using repression with varied activator concentrations.
supp_figure71_not_nor_nand_xor.py: simulates NOT, NOR, NAND, and XOR logic elements using coactivation and coactivator inducer RNAs.

2. **Class initialization definition:**

```
CLASS: GeneletNetwork(act_vec,prod_vec,blk_vec,indc_vec=0)
-------------------------------------------------------------------------------------
Initializes the genelet network instance to be simulated
```

Input definitions:

```
For act_vec and blk_vec:
      List the length of ind_nodes (total nodes)

      Numbers represent which orthogonal activators/blockers correspond to which
      nodes
            0 indicates no activator/blocker for a given node

For prod_vec:
      List the length of ind_nodes (total nodes)

      Numbers represent which RNA repressors and/or RNA coactivators are produced by
      which nodes

            -ve values are repressors
            +ve values are coactivators
             0 = no production of an inducer RNA

For indc_vec:
       List the length of ind_nodes (total nodes)

      Numbers represent which inducer RNAs are produced by which nodes
            -ve values are inducers that bind repressors (thereby activating)
            +ve values are inducers that bind coactivators (thereby BLKing)
              0 = no production of an inducer RNA

       indc_vec is optional is default to zeros if not provided
```

Output definitions:

```
After initializing the GeneletNetwork class the user has access to the following
Attributes [ where: model = GeneletNetwork( ) ]

model.ortho_nodes     - number of orthogonal genelet nodes
model.ind_nodes       - number of total (individual) genelet nodes
model.act_vec         - activator vector mapping ind_nodes to orthogonal activators
model.prod_vec        - RNA production vector mapping ind_nodes to which RNAs they
                        produce (repressors/coactivators)
model.blk_vec         - blocker vector mapping ind_nodes to orthogonal blockers
model.indc_vec        - RNA inducer production vector mapping ind_nodes to which
                        inducers they produce
model.topology_mat    - topology matrix (an alternative representation of
                        act_vec/prod_vec notation)
model.I_topology_mat  - inducer topology matrix (an alternative representation of
                        act_vec/indc_vec notation)
model.act_mat         - activator connection matrix used in ODEs to map activators to
                        ind_nodes
model.rep_mat         - repressor connection matrix used in ODEs to map repressors to
                        ind_nodes
model.blk_mat         - blocker connection matrix used in ODEs to map blockers to
                        ind_nodes
```

```
model.ca_mat          - coactivator connection matrix used in ODEs to map coactivators
                        to ind_nodes
model.Rprod_mat       - repressor production matrix used in ODEs to map which
                        ind_nodes produce which repressors
model.Cprod_mat       - coactivator production matrix used in ODEs to map which
                        ind_nodes produce which coactivators
model.Rindc_mat       - repressor inducer production matrix used in ODEs to map which
                        ind_nodes produce which repressor inducers
model.Cindc_mat       - coactivator inducer production matrix used in ODEs to map
                        which ind_nodes produce which coactivator inducers
```

3. **Function: initial conditions( ) definition:**

```
initial_conditions(dA_tot,G_tot,G_int_vec,dB_added=0,rRin=0,rCin=0,rIrin=0,rIcin=0)
-------------------------------------------------------------------------------------
Sets the initial conditions for the network. All concentrations should be input in nM
```

Input definitions:

For dA_tot and G_tot:

```
       numpy arrays the length of ortho_nodes and ind_nodes, respectively

      Represent total concentrations (nM) of activators and genelets, respectively
```

For G_int_vec:

```
      List the length of ind_nodes (total nodes)

      Represents the initial state of a node
            1 for ON
           -1 for BLK
            0 for OFF
```

For dB_added:

```
      All BLK genelets are assumed to be annealed with 50% excess dB
      Thus for 25 nM of a BLK there will be 32.5 nM total dB and 12.5 nM free dB to
      as the initial concentration

      To specify additional free dB modify the dB_added input (in nM)

      EX: model.initial_conditions(dA_tot,G_tot,G_int_vec,dB_added=[0,150,150])
```

For all other species the default values are 0:

```
      To modify these species input a list of concentrations for any other
      species by name (in nM):

      EX: model.initial_conditions(dA_tot,G_tot,G_int_vec,rCin =[0,150,150])
      EX: model.initial_conditions(dA_tot,G_tot,G_int_vec,rRin=[1000,0,0])
```

Output definitions:

After calling initial_conditions the user will have access to the following attributes:

```
model.G_int_vec  -the list of initial genelet states
model.ind_cond   -the vector holding the initial conditions for each species
model.(species)  -all of the individual species concentrations that make up int_cond
                  are also accessible as attributes by name (dB_added, rRin, etc)
```

4. **Function: simulate( ) definition:**

```
simulate(t_vec,iteration,rnase='RnH',leak=0,rate_constants=[],rR=[],dA=[],rC=[],dB=[],
rIr=[],rIc=[],dR=[])
--------------------------------------------------------------------------------
Simulates the network with the above initial conditions

Simulate can be called numerous times and specific conditions can be updated by name
       Use 'NA' for values that should not be updated for a given species

       For updating a species conditions, the list has to be the length of the total
       number of that species and input in nM

       dR is a special option where DNA repressors can be added to the reaction to
      permanently remove DNA activators
          dR species are currently not included in the symbolic equations that can be
          output as they are not really part of the networks

      t_vec2 should start at the last timepoint from t_vec1 and go to a later
      timepoint
```

Input definitions:

```
For rnase:
      Selects the rnase to use in the simulations
             'RnH' = RNase H (degrade RNA in RNA:DNA duplex)
             'RnA' = RNase A (degrade ssRNA and RNA in RNA:DNA duplex (likely with
                     slower rate than RNase H))
             'Both' = RNase H and A

      This input is optional and 'RnH' is the default

For leak:
      Represents the leak transcription rate of BLK genelets as a fraction of the ON
      transcription rate between 0 and 1

      Can be entered as a single value applied to all BLK genelets or as a list for
      different leak rates for each RNA production:
            leak = [0.05,0.1,0.075,0.06]
             There should be ind_node # of leak inputs as in this model each
             individual genelet has its own production rate even if two genelets
             produce the same RNA

For rate_constants:
      Use different rate constants than the default values

      Import as a list of single values (1/M-s for 2nd order reactions and 1/s for 1st
      order reactions):
      [kpr,kpc,kpi,kd_H,kd_A,kga,kgar,kar,kgb,kgab,kgbc,kbc,kir]


      Each individual rate can be a list of length ind_nodes for production rates
             Or a list of length ortho_nodes for other rates
             Or a single value which will be assumed to be the same for all nodes

EX: model.simulate(t_vec1,1)
EX: model.simulate(t_vec2,2,rIr=['NA','NA',10000])
EX: model.simulate(t_vec3,3,rR=[1000,'NA','NA'])
EX: model.simulate(t_vec1,1,rnase='RnA')  # use RNase A
```

```
EX: model.simulate(t_vec1,1,rnase='both',leak=0.1) # use both RNase A and H
                                                   # 10% leak transcription

EX: model.simulate(t_vec1,1,rnase='both',leak=0.1,rate_constants=rate_list)
                                                 # use both RNase A and H
                                                 # 10% leak transcription for all RNAs
                                                 # user input rate constants

EX: model.simulate(t_vec1,1,leak=[0.05,0.1,0.075]) # different leak rates for 3 RNAs
```

Output definitions:

After running simulate the user has access to the following useful attributes:

```
model.sol    -sol.y contains all of the concentration date and sol.t contains the time
              in seconds

model.(rate_constants)   -all of the individual rate constants can be accessed by name

model.output_concentration    -the dictionary containing all of the concentrations
                               with specific name keys

EX: G1 = model.output_concentration['GdA1']
EX: A1 = model.output_concentration['A1']
EX: R4 = model.output_concentration['rR4']
```

5. Function: export sym eqs( ) definition:

```
export_sym_eqs(file_name='sym_eqs',path_name='')
-------------------------------------------------------------------------------
```

Exports the symbolic ODE equations of the model to a text file

Both of the inputs are optional and have default values

Input definitions:

For file_name:

    User defined name of the .txt file (do not include .txt in the name)

For path name:

    User defined path to where the file should be saved (include \ at the end)

    If user does not supply a path the file is saved in current folder

```
EX: model.export_sym_eqs(file_name='TSN_eqs',path_name='C:\\Desktop\\')
```

Output definitions:

The output from this is the text file with the symbolic equations in it
with the specified or default filename in the specified or current (default) folder

6. <u>Function: plot topology( ) definition:</u>

```
plot_topology(pos=[],layout='spring',plot_title='',show_rnas=1)
-------------------------------------------------------------------------------
Plots the network topology with initial node states

This considers both genelet nodes and RNA coactivators/repressors as topological nodes
which allows inducer RNA connections to be represented

If this is called right after initializing the network the initial states will be
inferred

If this is called after the initial_conditions function has been called the user
defined initial states will be displayed
```

<u>Input definitions:</u>

<u>For pos:</u>

```
      This is an optional input of each nodes position in the plot
            It has to be a dictionary with keys that are the names of the topological
            nodes and the desired [x,y] position values
```

<u>For layout:</u>

```
      This is an optional input that selects the algorithm to organize the nodes in
      the plot

            Default is 'spring'
            Other options:
                 'spectral'
                 'circular'
                 'shell'

      See the networkX package layout options for further details
```

<u>For plot_title:</u>

```
      This is an optional title for the topology plot
```

<u>For show_rnas:</u>

```
      This is an option to show the RNA repressors/coactivators in the topology as
      their own nodes

      This is necessary if you want to plot topologies where there are nodes that
      produce inducers
            If this is set to 0 then the RNAs will not be plotted and any inducer
            nodes will appear free standing
```

```
EX: model.plot_topology(layout='shell',plot_title='IFFL network')
```

<u>Output definitions:</u>

```
The output from this is the topology plot

model.net_edges gives the user access to all the network edges used in the plot
```

7. Function: compile_network_sequences( ) definition:

```
compile_network_sequences(input_file_loc,desired_nodes=[],min_design=0,bth=1,\
save_file_name='genelet_seqs',save_path_name='')
-------------------------------------------------------------------------------
Exports genelet sequences for a specified topology to a text file

Algorithm for domain selection for unspecified nodes:

      First, nodes that must both be coactivated AND repressed are populated with
      domains that meet this criteria selected in order of their OCRW score in Supp.
      Section 12

      Second, nodes that do not have ANY coactivator or repressor inputs are
      Populated
            Since these nodes are the likely place a network will be expanded they
            are:
                  First populated with domains that can be coactivated AND repressed
                  selected in order of their OCRW score in Supp. Section 12

                  Second populated with domains that can be coactivated selected in
                  order of their OCRW score in Supp. Section 12

                  Third populated with domains that can be coactivated selected in
                  order of their OCRW score in Supp. Section 12

      Third, nodes that have ONLY coactivator inputs are populated
            If min_design = 0 the first available domain that can be coactivated will
            be selected in order of their OCW score in Supp. Section 12

            If min_design = 1 the first available domain that can ONLY be coactivated
            will be selected in order of their OCW score in Supp. Section 12
                  If there are not enough min_design domains then the remaining
                  domains will be selected as if min_design = 0

      Fourth, nodes that have ONLY coactivator inputs are populated
            If min_design = 0 the first available domain that can be repressed will
            be selected in order of their OCW score in Supp. Section 12

            If min_design = 1 the first available domain that can ONLY be repressed
            will be selected in order of their ORW score in Supp. Section 12
                  If there are not enough min_design domains then the remaining
                  domains will be selected as if min_design = 0
```

Input definitions:

For input_file_loc:

The file path to the Excel file holding all of the viable genelet node Sequences (all_genelet_sequences.xlsx)

For desired_nodes:

An optional input that specifies which input domains should be used for each TN in the network

By default, this is empty, algorithm will select its own nodes if not provided

```
      Provide as a list of length ortho_nodes with the names of nodes to be used to
      compile the sequences (['G1','G5','G8'])
                              TN1  TN2  TN3
      The order of the list of nodes is in the order of the topological nodes as
```

shown above

An error will occur if you select a domain that cannot conduct the regulation required at that position

i.e. if the TN needs to be coactivated and repressed and you select a domain that only represses

If only some nodes are specified and others are to be selected, leave the unspecified positions blank:
```
([ 'G1',' ','G8'])
   TN1  TN2  TN3
```

The unspecified positions will be filled with other domains using the same algorithm as when no nodes are specified

If there is an unspecified position and min_design = 1 then the unspecified node will attempt to use a min_design domain if possible

For min design:

An optional input to specified if sequences selected should adopt a minimum design

By default this is 0 so the first nodes that meet the needs of the network will be used as described for the algorithm

If min_design = 1 the algorithm will select nodes with the minimal possible function in the network

For example, if a node is only coactivated in the network then the algorithm will use the first node that can only be coactivated at this position

If there are not enough minimal nodes to fill the network other nodes will be selected as described for the algorithm

For bth:

Optional input that specifies whether or not BTH domains should be included

This is 1 by default which keeps the 8 base 5' blocker toehold (BTH) on all the genelets in the network

This can be set to 0 and then the 8 base 5' blocker toehold will be removed from all genelets that are only repressed in a given network as these nodes do not need a blocker

This makes it possible to order shorter -nt sequences without having to truncate their 3' end (as for the TSN)

All such -nt sequences will be denoted with a * in the output file

For save file name:

User defined name of the .txt file to be saved (do not include .txt in the name)

If user does not supply a name it will be saved as 'genelet_seqs.txt'

For save path name:

User defined path to where the file should be saved (include \ at the end)

If user does not supply a path the file is saved in current folder

```
EX:
model.compile_network_sequences(input_file_loc,save_file_name='TSN_sequences',save_pat
h_name='C:\\Desktop\\')
```

Output definitions:

The output from this is the text file with the genelet sequences in it

Each node is comprised of G-nt / G-t / activator / blocker sequences as needed for the simulated network

All -nt sequences that have had the BTH removed will be denoted with a * in the output file

Currently, any dummy reporter nodes (redundant nodes that do not produce any RNAs) will not be included as exported sequences

8. TROUBLESHOOTING

When doing iterative model.simulate calls make sure the t_vec units are correct, if t_vec1 = linspace(0,5,1000)*3600 then t_vec2 = linspace(t_vec1[-1]/3600,10,1000)*3600

The package currently does not populate sequences for any redundant dummy nodes in the system.
