Python in the VE

Genetic optimisation using PyGAD

What? Using PyGAD, a non-bound open source library, to carry out genetic optimisation in the VE

Why? Genetic optimisation is an alternative to a parametric uncertainty approach … rather than simulate all combinations of possible model changes a genetic algorithm runs samples of model changes in a series of 'generations' picking the best of each generation to 'seed' the choices in the next generation. What is 'best' is derived using a test of 'fitness'.

In this example we use the PyGAD library, installed via the paths facility in VE Scripts to create a Multiple objective Genetic optimisation script that utilises the scripts (modules) we created for TM54.

```
"""
=======================================
Genetic optimization - multiple objective
=======================================

Module description
------------------
Genetic optimisation tool for optimisation purposes.

For an introduction on genetic optimization see:
https://stackoverflow.com/questions/69544221/how-can-i-find-the-optimal-parameters-for-a-function

This script uses the PyGAD library; refer to https://pygad.readthedocs.io/en/latest/
PyGAD is an external library that needs adding in VE Scripts using the VE Scripts config tool.

In this script we define and solve a multi-objective optimisation problem.

```

Script background and instructions

```
In the script main_loop:

- Set the target metrics that will be minimized (try 2 or 3)
- Set the desired target values (pick design targets, however if you pick zero the sims
  will run all generations if it cannot converge)
- Set the required inputs for the lower / upper bounds (lower must be < upper) and step.
  The step value is important as it minimises the creation of models with genes that have
  close values. Note: shade changes are cumulative so use small values & -ve lower bound
- For ID based inputs setup the mapped_ids dict
- Set the simulation route
- Set if you want load sizing sims and makes sure boundaries['asp_file'] is set
- Set the model to edit (normally the real model)
- Set the optimisation parameters. Adjust these to suit your needs

It will be necessary to experiment with # of generations & population. If these parameters
are not set large enough an optimal minima will not be found. Smaller parameter sets should
show a minima but it is unlikely to be optimal; however this still may offer value. As noted
above this is a very good reason to consider a series of parametric analyses, coarse optimization
etc.

The script will delete each aps/asp file after each iteration.
The script will archive the model before making any changes.

The GA_output.csv will be overwritten before a new optimization task so move/rename the
file if you wish to keep it.
```

Read the instructions for what to edit

```
import iesve
import importlib
import pygad
import pandas as pd
import utils_genetic
from datetime import datetime
from pathlib import Path

# Reload utils
importlib.reload(utils_genetic)

# Main loop
if __name__ == "__main__":

    # Get the current project
    project = iesve.VEProject.get_current_project()

    ### Archive the project
    print('Archiving project ...')
    project_folder = project.path
    time_stamp = datetime.now().strftime("%d%m%Y_%H%M%S")
    path = Path(project_folder, 'Backups', f'{project.name}_{time_stamp}.zip')
    project.archive_project(str(path), False)
    print('Project archived to project backups folder')
    print(' ')

    ### Define the optimization problem to minimise; enter 2 or 3 objectives
    # Example coded list entries:
    # 'Gas_MWh',

    target = ['CE_kgCO2/m2', 'EUI_kWh/m2']
    # Define target values
    desired_target_values = [0.0, 0.0]
    # Number of objectives
    no_objectives = len(target)

    ### Define the output metrics
    # For df column names use snake_case. Spaces are not permitted, example coded list
    # Summary entries:
    # 'Gas_MWh',
    # 'Elec_MWh',
    # 'Gas_kWh/m2',
    # 'Elec_kWh/m2'
    # 'Boilers_MWh',

    outputs = [
                'Gas_MWh',
                'Elec_MWh',
                'Gas_kWh/m2',
                'Elec_kWh/m2',
                'Boilers_MWh',
                'Chillers_MWh',
                'Boilers_kWh/m2',
                'Chillers_kWh/m2',
                'CE_kgCO2/m2',
                'UK_BER_kgCO2/m2',
                'EUI_kWh/m2',
                'Ta_max_degC',
                'Boiler_max_kW',
                'Chiller_max_kW',

                'Interior_lighting_kWh/m2',
                'Exterior_lighting_kWh/m2',
                'Space_heating_(gas)_kWh/m2',
                'Space_heating_(elec)_kWh/m2',
                'Space_cooling_kWh/m2',
                'Pumps_kWh/m2',
                'Fans_interior_kWh/m2',
                'DHW_heating_kWh/m2',
                'Receptacle_equipment_kWh/m2',
                'Elevators_escalators_kWh/m2',
                'Data_center_equipment_kWh/m2',
                'Cooking_(gas)_kWh/m2',
                'Cooking_(elec)_kWh/m2',
                'Refrigeration_kWh/m2',
                'Wind_PV_kWh/m2'
                ]

    ### Define the optimization problem inputs (genes or boundaries) in a dict
    # Example coded dict entries [lower, upper, step]; comment-out inputs not required:
    boundaries = {}

    # ... numeric inputs
    # Step use int values when possible
    boundaries['building_orientation'] = [90, 180, 15]
    boundaries['room_heating_setpoint'] = [19.0, 22.0, 0.5]
    #boundaries['room_cooling_setpoint'] = [25.0, 28.0, 1.0]
    #boundaries['apsys_scop'] = [0.7, 0.9, 0.1]
    #boundaries['apsys_sseer'] = [2.0, 3.0, 0.1]
    #boundaries['sys_free_cooling'] = [4.0, 5.0, 0.5]
    #boundaries['ncm_terminal_sfp'] = [0.1, 0.5, 0.1]            # NCM only
    #boundaries['ncm_localexhaust_sfp'] = [0.1, 0.5, 0.1]        # NCM only
    #boundaries['ncm_light_pho_parasit'] = [0.01, 0.05, 0.01]     # NCM only
    #boundaries['ncm_light_occ_parasit'] = [0.01, 0.05, 0.01]     # NCM only
    #boundaries['window_openable_area'] = [25.0, 30.0, 1.0]
    #boundaries['ext_wall_glazing'] = [5.0, 50.0, 5.0]
    #boundaries['wall_const_u_value'] = [0.1, 0.3, 0.1]
    #boundaries['window_const_u_value'] = [1.0, 1.5, 0.25]
    #boundaries['roof_const_u_value'] = [0.1, 0.3, 0.1]
    #boundaries['floor_const_u_value'] = [0.1, 0.3, 0.1]
    #boundaries['outer_pane_transmittance'] = [0.2, 0.5, 0.1]
    #boundaries['outer_pane_reflectance'] = [0.6, 0.8, 0.1]
    #boundaries['local_shade_overhang'] = [-0.1, 0.1, 0.1]    # NB. cumulative
    #boundaries['local_shade_depth'] = [-0.1, 0.1, 0.1]       # NB. cumulative
    #boundaries['pv_area'] = [10.0, 50.0, 10.0]

    # ... string id by integer index (maps to id, name or filename)
    # Index low value must be set to 0 (zero indexed)
    # Index high value must be set to the number of items as PyGAD appears to
    # pick values within the range excluding the high limit (so zero index range + 1)
    # Step must be set as 1
    #boundaries['weather_file'] = [0,3, 1]
    boundaries['ap_system'] = [0, 3, 1]
    #boundaries['infiltration_rate'] = [0, 2, 1]
    #boundaries['gen_lighting_gain'] = [0, 2, 1]
    #boundaries['computer_gain'] = [0, 2, 1]
    boundaries['wall_construction'] = [0, 3, 1]
    #boundaries['window_construction'] = [0, 2, 1]
    #boundaries['roof_construction'] = [0, 2, 1]
    #boundaries['floor_construction'] = [0, 2, 1]
    #boundaries['asp_file'] = [0, 3, 1]

    ### For string/id based boundaries define the mappings between index and ids
    mapped_ids = {}
    # Example coded list entries; comment-out inputs not required:
    #mapped_ids['weather_file'] = ['LondonDSY2020H.fwt', 'LondonDSY2050H.fwt', 'LondonDSY2080H.fwt']
    mapped_ids['ap_system'] = ['SYST0001', 'SYST0002', 'SYST0003']
    #mapped_ids['infiltration_rate'] = ['Infiltration 0.5', 'Infiltration 0.75']
    #mapped_ids['gen_lighting_gain'] = ['General Lighting 5','General Lighting 10']
    #mapped_ids['computer_gain'] = ['Computers 3','Computers 5']
    mapped_ids['wall_construction'] = ['STD_WAL1', 'STD_WAL2','STD_WAL3']
    #mapped_ids['window_construction'] = ['STD_EXT1','STD_EXT2']
    #mapped_ids['roof_construction'] = ['STD_ROOF1', 'STD_ROOF2']
    #mapped_ids['floor_construction'] = ['STD_FLO2', 'STD_FLO3']
    #mapped_ids['asp_file'] = ['CAV.asp', 'VAV.asp', 'UFAD.asp']

    ### Define the simulation route
    #   0 is run_simulation() i.e. Apache
    #   1 is run_compliance_simulation () i.e. UK Part L
    route = 0

    ### Define loads sims (asp_file must be defined)
    # True for ON, False for off. Set to False for UK Compliance
    loads_on = False

    ### Set the model to edit
    #   The Real model is index 0 in the project.models list
    #   The Real, Actual & Proposed models are the same model; actual/proposed room model
    #   data is updated following edits in accordance with relevant compliance rules
    model_index = 0

    ### Create an instance of the model_utils class
    # This initialises the simulation output and model modifications
    problem = utils_genetic.model_utils(target,
                                        outputs,
                                        boundaries,
                                        mapped_ids,
                                        route,
                                        loads_on,
                                        model_index,
                                        'GA_MO_output.csv')

    ### Create the fitness function for PyGAD
    def fitness_func(ga_instance, solution, solution_idx):
        # Calculates the fitness for each solution
        # This is used by PyGAD to assess if a gene solution is better or worse
        # Create empty fitness output list
        fitness = []
        # Call model_utils simulation method, this returns a list of values, one per objective
        output = problem.simulation(solution)
        # PyGAD uses a maximization method so that a high fitness value returned
        # is selected compared to a solution with a low value. All our target metrics
        # are better when lower so the calculated fitness values need to be 'inverted'
        # Create a fitness value for each objective
        for n in range (0, no_objectives):
            value = 1.0 / (abs(output[n] - desired_target_values[n]) + 0.000001)
            fitness.append(value)
        return fitness

    ### Create a gene space list from boundaries dict
    # This is the boundaries data rearranged in a format suitable for PyGAD
    # The format is a dict of low, high & optional step values for each gene
    # Including step
    gene_space = []
    for n in boundaries:
        if len(boundaries[n]) == 3:
            # Step included
            gene_space.append({'low': boundaries[n][0], 'high':  boundaries[n][1], 'step': boundaries[n][2]})
        else:
            # Step excluded
            gene_space.append({'low': boundaries[n][0], 'high':  boundaries[n][1]})

    ### PyGAD gene type list
    # It appears that PyGAD gene_type only accepts float, not int or list [class] despite what is
    # documented, so this parameter is not usable. According to the following reported issue if all
    # gene space values are int then the population will be int, otherwise it will be float. This
    # reinforces the importance of using step as this will operate as a workaround
    # https://github.com/ahmedfgad/GeneticAlgorithmPython/issues/27

    ### Set the optimisation parameters
    num_generations = 3           # The number of generations
    sol_per_pop = 5               # The population size per generation
    num_parents_mating = 2        # The number of solutions to be selected from the population as parents
    num_genes = len(gene_space)
    parent_selection_type='nsga2'

    ### Create a function to be called each generation
    def on_generation(ga_instance):
        print("-------")
        print("Generations completed : ", ga_instance.generations_completed)
        #print(" ")
        #print("Fitness of the best solution this generation :", ga_instance.best_solution()[1])
        print("-------")

    ### Create an instance of the pygad.GA class
    # The constructor is responsible for creating the initial population
    ga_instance = pygad.GA(num_generations=num_generations,
                           num_parents_mating=num_parents_mating,
                           sol_per_pop=sol_per_pop,
                           num_genes=num_genes,
                           mutation_percent_genes=100,
                           fitness_func=fitness_func,
                           parent_selection_type='nsga2',
                           gene_space=gene_space,
                           save_solutions=True,
                           on_generation=on_generation,
                           save_best_solutions=True,
                           suppress_warnings=True)

    ### Run the genetic optimisation
    ga_instance.run()

    # Plot optimisation data
    ga_instance.plot_genes()
    # Close the plot dialog to allow script to proceed

    # Output the details of the best solution for each generation
    best_solns = ga_instance.best_solutions
    print(' ')
    print('Best solution per generation ...')
    print(best_solns)
```

Import modules, note the PyGAD module requires separate installation, then is utilised as normal

Reload() refreshes the referenced utils module to ensure any changes are picked up during the sesssion

Archive the project

Comments listing the available objectives

Set the targets for multiple optimisation in a list

Set the target values in a list. Try to set realistic values rather than just zeros

Comments listing the available model outputs

Set the outputs in a list. There is little reason to edit this list as having all the outputs in the output is useful

Create an empty dict to define the model changes

Comment-out unwanted lines
For wanted inputs set range low, range high and step inputs

Where possible use integers

Step value is important as it defines the sensitivity you want for the optimisation

Comment-out unwanted lines

PyGAD has to use numbers so for string based references we will substitute a dict of indices

For wanted inputs set range low, range high and step inputs … for indexed inputs set range low to zero and step to 1. Range high is set to zero index end +1 e.g. if there are three items the index is 0,3,1

For string based references we will create a matched dict of string references to which the indices point

Note that we are using IDs for systems, constructions and file names for weather and HVAC network files

Set the simulation route

Set if loads is being utilised as a pre-simulation. If True make sure boundaries['asp_file'] is also set up

Set which model to simulate

The utils_genetic module contains a class that will allow us to pass data to the simulation by using a class instance outside of the parameter set allowed by PyGAD. This is much tidier than using global variables. Here we create an instance of this class. See the comments in utils_genetic module for more information.

A key part of PyGAD is creating a fitness function which is used to test the output of each simulation. The parameters are fixed by PyGAD

The solution parameter passes-in the combination of model changes (genes) that PyGAD has determined

In turn we pass the solution parameter to the function method of our instance of the utils_genetic class. This method makes changes to the model, simulates and returns values for the output metrics

We process the output metrics to fitness values that will be tested by a *maximise* function in PyGAD. Because our metrics are by nature best when minimums we need to invert them for the purposes of PyGAD

The *boundaries* dict is user readable, but not in the format that PyGAD requires, so we process it in a format that PyGAD can consume

The PyGAD documentation appears at odds with the API. We have used a work around in the boundaries dict to get around the issue this causes

Set the main parameters for the genetic optimisation

Create a function that PyGAD calls after each generation e.g. to output progress information

Create an instance of the PyGAD GA class and set the parameters

For ga_instance call the run method to start the genetic optimsation

After optimisation is complete show plots of the genes through the optimisation process. **Close the dialog to let the script continue**

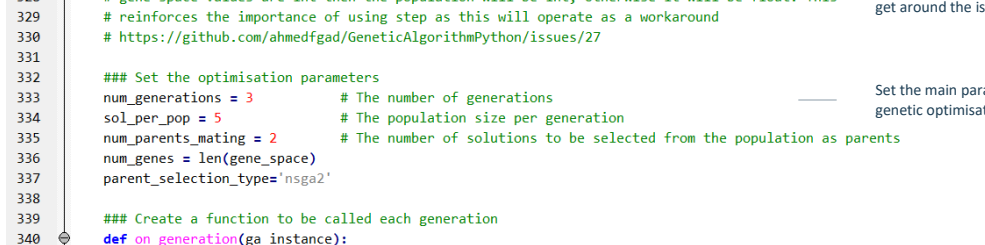
Output the optimum model changes for each generation

The script also outputs a CSV file that can be plotted with *chart_parallel.py*

Output …

```
IES Virtual Environment Python 3.7.2 (default, Jan 12 2022, 16:57:37) [MSC v.1916 64 bit (AMD64)]

  >>> Run start, Mon Oct 21 10:09:11 2024
Archiving project ...
Project archived to project backups folder

Applying gene set model changes ...  {'building_orientation': 105.0, 'room_heating_setpoint': 20.0, 'ap_system': 'SYST0003', 'wall_construction': 'STD_WAL3'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [132.98, 359.68]

Applying gene set model changes ...  {'building_orientation': 165.0, 'room_heating_setpoint': 20.5, 'ap_system': 'SYST0001', 'wall_construction': 'STD_WAL2'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [129.38, 343.01]

Applying gene set model changes ...  {'building_orientation': 90.0, 'room_heating_setpoint': 19.5, 'ap_system': 'SYST0002', 'wall_construction': 'STD_WAL2'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [122.63, 311.77]

Applying gene set model changes ...  {'building_orientation': 150.0, 'room_heating_setpoint': 19.0, 'ap_system': 'SYST0003', 'wall_construction': 'STD_WAL2'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [127.11, 332.5]

Applying gene set model changes ...  {'building_orientation': 165.0, 'room_heating_setpoint': 21.5, 'ap_system': 'SYST0001', 'wall_construction': 'STD_WAL2'}
```

VEScripts output pane … the optimise script running

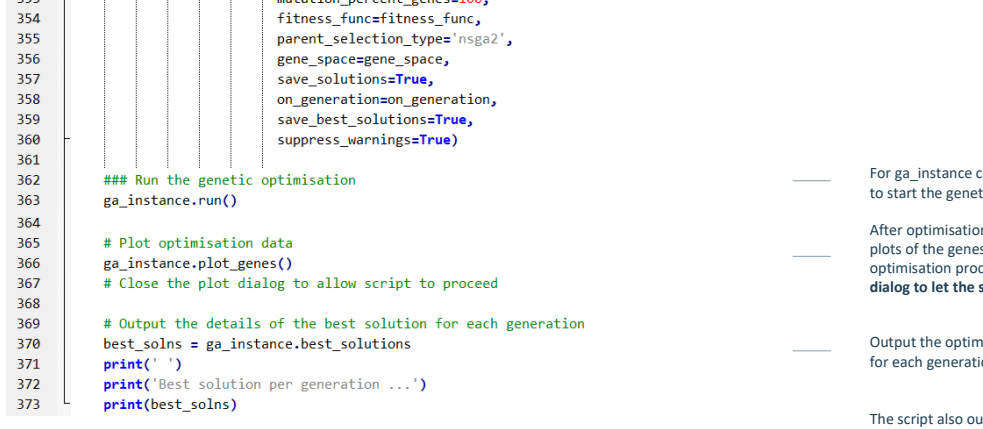


The Genes plot

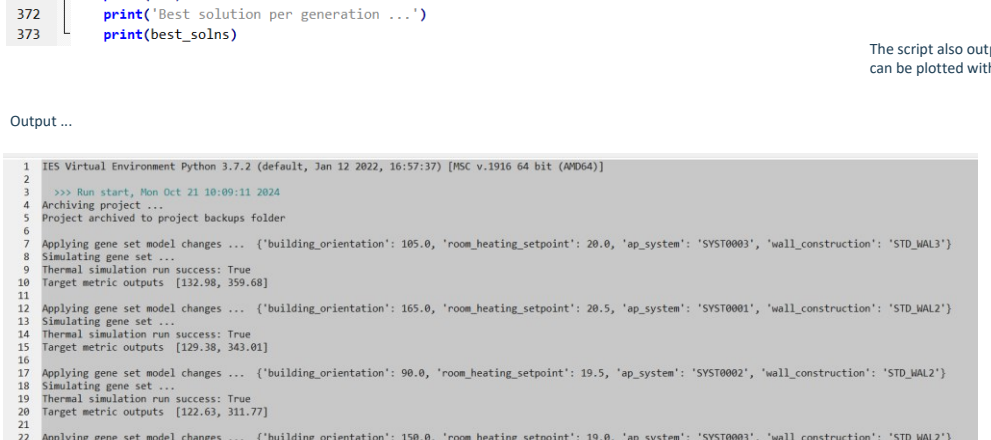
```
Applying gene set model changes ...  {'building_orientation': 90.0, 'room_heating_setpoint': 21.0, 'ap_system': 'SYST0001', 'wall_construction': 'STD_WAL1'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [132.5, 357.45]

Applying gene set model changes ...  {'building_orientation': 90.0, 'room_heating_setpoint': 19.5, 'ap_system': 'SYST0003', 'wall_construction': 'STD_WAL1'}
Simulating gene set ...
Thermal simulation run success: True
Target metric outputs  [130.43, 347.87]

-------
Generations completed :  3
-------
Best solution per generation ...
[[150.  19.   1.   2.]
 [150.  19.   1.   2.]
 [150.  19.   1.   2.]
 [150.  19.   1.   2.]]
  >>> Runtime: 26 minutes 55.3 seconds
```

VEScripts output pane … completion

The best solution was found in the first generation in this instance



Output CSV file with all inputs and outputs



Loading the output CSV file into script … chart_parallel.py