Python no AR2GeMS

Scripts

Estrutura dos points sets

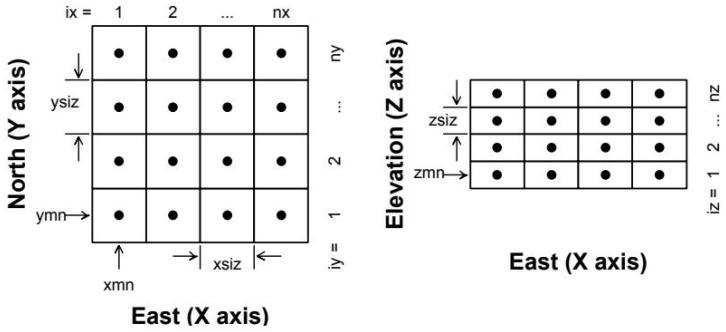
Os points sets são um vetor de coordenadas X, um vetor de coordenadas Y e um vetor de coordenadas Z. em uma determinada ordem. Os teores associados à esse point set são um vetor de valores, ordenados, conforme os vetores de coordenadas.

Estrutura dos grids

Um grid está associado apenas à um vetor de valores ordenados de acordo com o índice de cada nó, dessa forma as coordenadas não precisam ser guardadas.

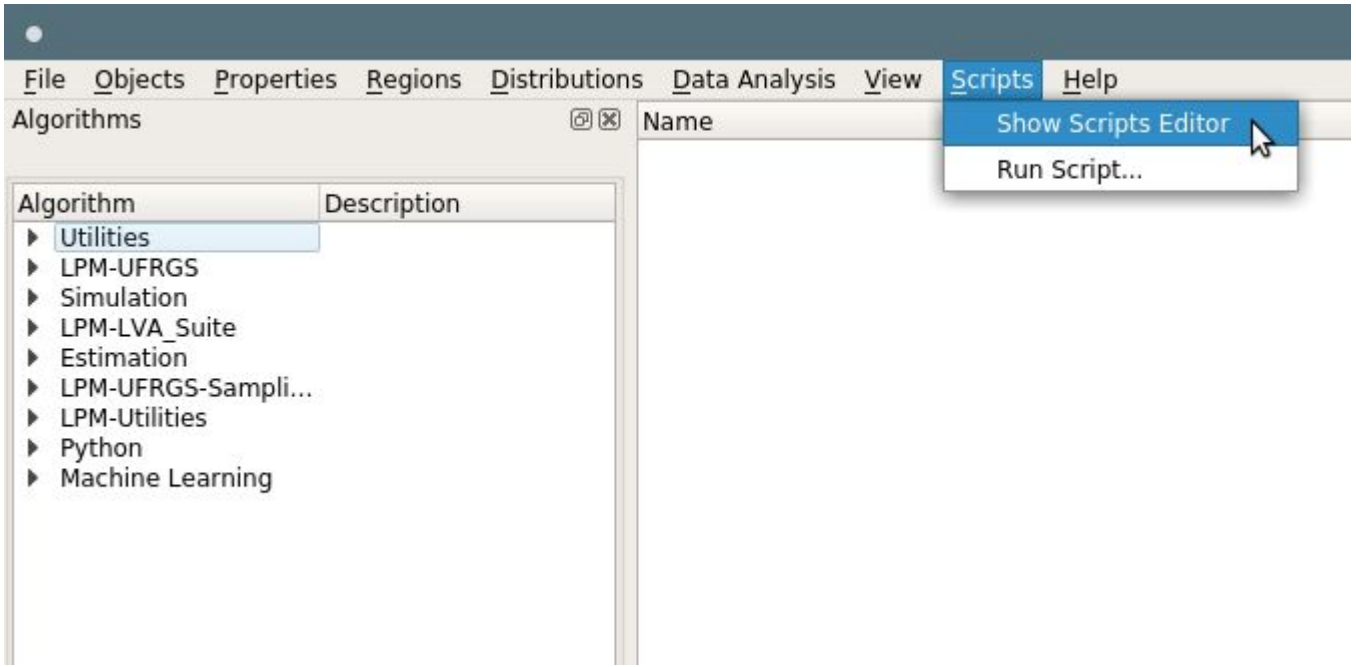
- Cresce, nó a nó, para leste;
- Então linha à linha para o norte;
- E finalmente nível por nível para cima.

X é percorrido mais rápido, depois Y e por fim Z.

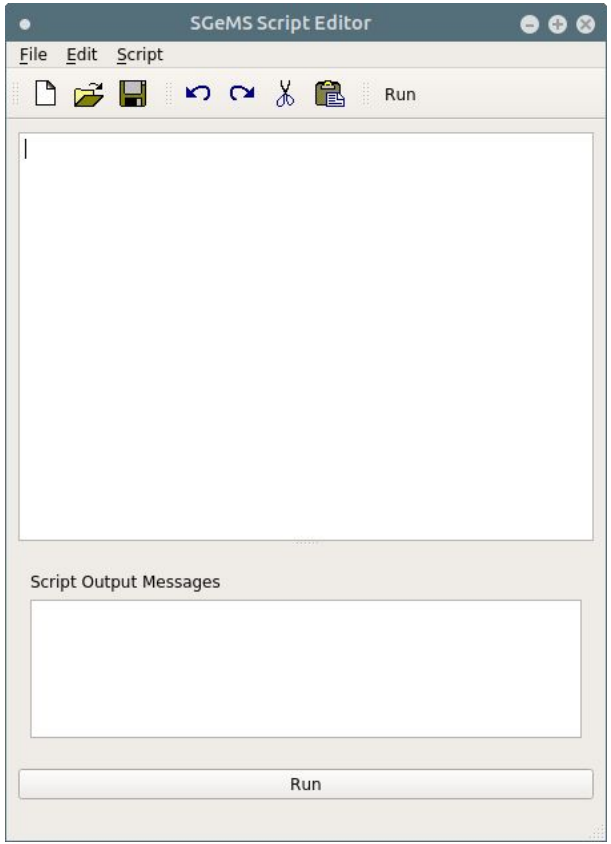


$$loc = (iz - 1) * nx * ny + (iy - 1) * nx + ix$$

Scripts em python

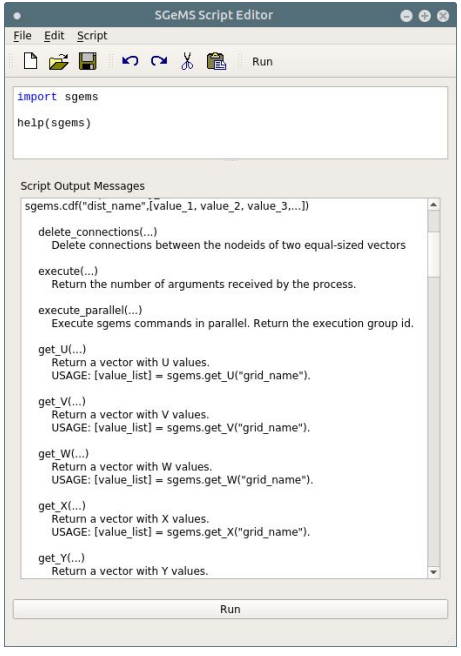


Editor

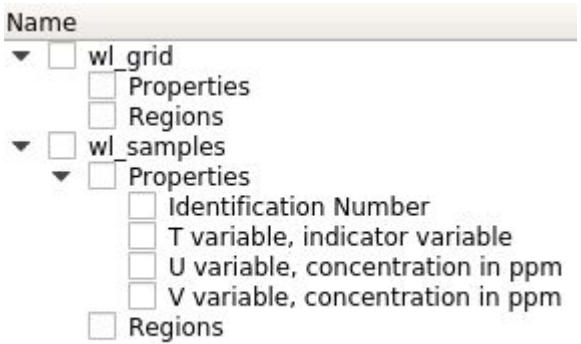

SGeMS Script Editor
File
Edit
Script
Run
Script Output Messages
Run

Módulo sgems

A integração sgems + python se dá através do módulo sgems que deve ser importado com o comando `import sgems`. O comando `help(sgems)` lista todas as funções disponíveis.

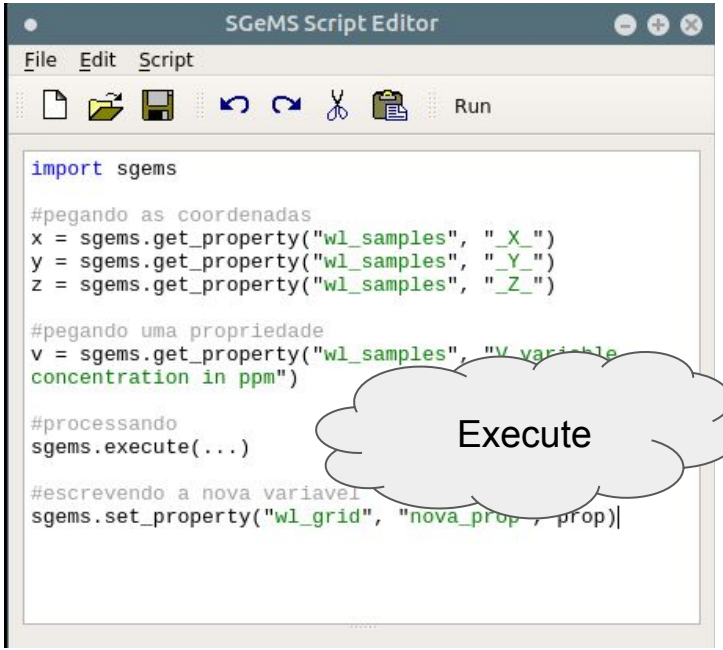


Workflow



Get functions

Set functions



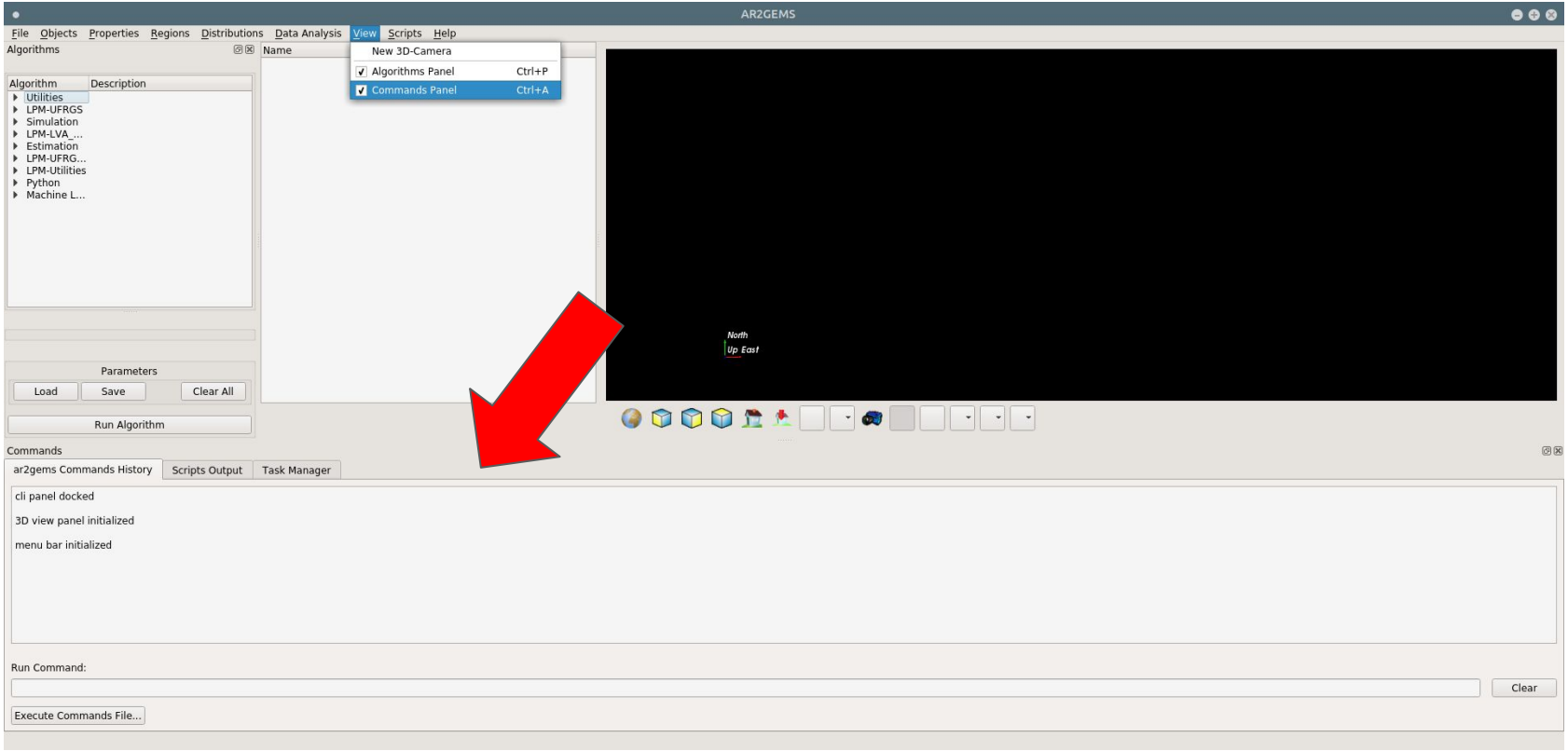
Get functions (lista)

Function	Description	Example
`get_X(...)`	Return a vector with X values.	`[value_list] = sgems.get_X("grid_name")`
`get_Y(...)`	Return a vector with Y values.	`[value_list] = sgems.get_Y("grid_name")`
`get_Z(...)`	Return a vector with Z values.	`[value_list] = sgems.get_Z("grid_name")`
`get_property(...)`	Return a vector with property values if informed.	`[value_vector] = sgems.get_property("grid_name","property_name") - Use _X_ or _Y_ or _Z_ for x,y,z.`
`get_property_list(...)`	Return the list of property name in a grid.	`[prop_list ]= sgems.get_property_list("grid_name")`
`get_region(...)`	Export a region from a grid.	`[v1,v2,v3...] = sgems.get_region("grid_name","region_name")`
`get_closest_nodeid(...)`	Return the closest nodeid from a x,y,z location.	`[nodeid ]= sgems.get_closest_nodeid("grid_name", x,y,z)`
`get_grid_info(...)`	Get informations about the grid.	`{'z_rotation': z_rotation,'origin':[ox, oy, oz], 'dimension':[xsize, ysize, zsize], 'num_cells':[nx, ny, nz]} = sgems.get_grid_info("grid_name")`
`get_ijk(...)`	Return the i,j,k indexes of a regular grid based on the nodeid.	`[i,j,k] = sgems.get_ijk("grid_name", nodeid)`
`get_location(...)`	Return the x,y,z location of a grid based on the nodeid.	`[x,y,z] = sgems.get_location("grid_name", nodeid)`
`get_masked_grid_full_X(...)`	Return a vector with X values of the cartesian grid behind a masked grid.	`[value_list] = sgems.get_masked_grid_full_X("masked_grid_name")`
`get_masked_grid_full_Y(...)`	Return a vector with Y values of the cartesian grid behind a masked grid.	`[value_list] = sgems.get_masked_grid_full_Y("masked_grid_name")`
`get_masked_grid_full_Z(...)`	Return a vector with the Z values of the cartesian grid behind a masked grid.	`[value_list] = sgems.get_masked_grid_full_Z("masked_grid_name")`
`get_masked_grid_mask(...)`	Return a vector of the mask of a masked grid.	`[value_list] = sgems.get_masked_grid_mask("masked_grid_name")`
`get_neighbors(...)`	Get the neighbors id for a list of nodeid.	`[nodeids] = sgems.get_neighbors("grid_name","prop_name", range1, range2, range3, azimuth, dip, rake, max_neighbors, "grid_source_name","prop_source_name", list_of_nodeids)`
`get_nodeid(...)`	Return the nodeid from a x,y,z location.	`[nodeid] = sgems.get_nodeid("grid_name", x,y,z)`
`get_nodeid_from_ijk(...)`	Return the nodeid from a i,j,k indexes for a regular grid.	`[nodeid] = sgems.get_nodeid_from_ijk("grid_name", i,j,k)`

Exercício

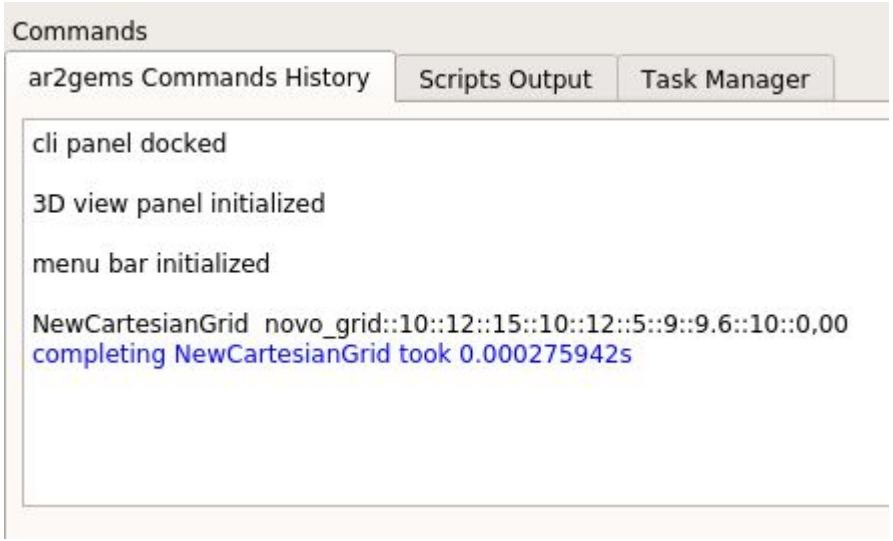
1. Carregue o banco de dados walker lake;
2. Atribua a uma variável os valores da coordenada X usando a função `get_X(...)` e imprima os 10 primeiros valores;
3. Faça o mesmo para Y e Z;
4. Atribua a uma variável os teores de V usando a função get_property(...) e imprima os 10 primeiros valores;
5. Crie uma região qualquer no ar2gems;
6. Atribua a uma variável os valores da região usando o comando get_region(...) e imprima todos os valores.

Command Panel



Criando um grid

```
NewCartesianGrid nome::nx::ny::nz::sx::sy::sz::ox::oy::oz::rotz
```



O método format

```
nome = “Andre”

idade = 25

string = “Ola! Meu nome eh {}, eu tenho {} anos”.format(nome, idade)

print(string)

Ola! Meu nome eh Andre, eu tenho 25 anos
```

Execute functions

```
sgems.execute("NewCartesianGrid novo_grid::10::15::25::10::12::5::9::9.5::5::0")
```

Um jeito mais prático:

```
nx, ny, nz, sx, sy, sz, ox, oy, oz, rotz = 10, 15, 25, 10, 12, 5, 9, 9.5, 5, 0

comando = "NewCartesianGrid novo_grid::{}::{}::{}::{}::{}::{}::{}::{}::{}::{}".format(nx, ny, nz,
sx, sy, sz, ox, oy, oz, rotz)

sgems.execute(comando)
```

Exercício

1. Crie um grid 1x1 (Objects > New cartesian grid);
2. Imprima as informações do grid usando a função get_grid_info(...);
3. Rode um NN para a variável V e observe o resultado no command panel (Estimation > nearest-neighbor);
4. Reproduza o mesmo resultado mas dessa vez a partir de um script, use a função sgems.execute(...) e o método format;
5. Faça o mesmo, mas dessa vez usando o controle de fluxo for, para as variáveis U e V.

Set functions (lista)

Function	Description	Example
`set_UVW(...)`	Set UVW coordinates on grid from fully-informed list.	`sgems.set_UVW_from_properties("grid_name","prop1","prop2","prop3")`
`set_UVW_from_properties(...)`	Set UVW coordinates on grid from fully-informed grid properties.	`sgems.set_UVW_from_properties("grid_name","prop1","prop2","prop3")`
`set_categorical_property_alpha(...)`	Set a categorical property from a list of alphanumeric entries (string).	`[void] = sgems.set_categorical_property_int("grid_name", property_name, strings)`
`set_categorical_property_int(...)`	Set a categorical property from a list of integer.	`[void] = sgems.set_categorical_property_int("grid_name", property_name, property_values)`
`set_cell_property(...)`	Change or create a property of a grid at specific nodeid	`sgems.set_cell_property("grid_name","property_name",[node1,node2,node3...],[val1,val2,val3...])`
`set_connections(...)`	Set connections between the nodeids of two equal-sized vectors.	`[void] = sgems.set_connections(list_of_N_nodeid_left, list_of_N_nodeid_right)`
`set_property(...)`	Change or create a property of a grid.	`sgems.set_property("grid_name","property_name",vector of values populated in nodeid order)`
`set_region(...)`	Import a region to a grid.	`sgems.set_region("grid_name","region_name",[v1,v2,v3...])`

Exercício

1. Atribua a uma variável os teores de U usando a função get_property(...);
2. Use a biblioteca numpy para calcular o logaritmo de todos os itens da lista variável U, atribua o novo vetor a uma variável; Dica: cuidado com os zeros! np.where(...);
3. Escreva a nova variável no point set usando a função sgems.set_property(...);
4. Visualize a nova variável;
5. Imprima todos os valores da variavel U;
6. Crie uma nova variável que vale 1 se U for NaN e zero caso contrário; Dica: Use np.isnan(...) para checar se um valor NaN.

Outras funções

Function	Description	Example
new		
`new_masked_grid(...)`	Create a new masked grid, given a set of cartesian grid parameters and a mask.	`sgems.new_masked_grid("grid_name",[nx, ny, nz], [ox, oy, oz], [sx, sy, sz], [rot_ox, rot_oy, rot_oz, angle], mask)`
`new_point_set(...)`	Create a new point set or append to existing point set, given a set of x,y,z coordinates.	`sgems.new_point_set("point_set_name",x,y,z)`
distributions		
`cdf(...)`	Get the cdf values (probability below) from a list of quantiles.	`[probability_below values list (doubles between 0 and 1)] = sgems.cdf("dist_name",[value_1, value_2, value_3,...])`
`pdf(...)`	Get the pdf values (probability) from a list of quantiles.	`[probability values list (doubles between 0 and 1)] = sgems.pdf("dist_name",[value_1, value_2, value_3,...])`
`quantile(...)`	Get the inverse cdf values (quantiles) from a list of probabilities.	`[quantile values list] = sgems.quantile("dist_name",[.1, .2, .5,...])`
execution		
`execute(...)`	Return the number of arguments received by the process.	
`execute_parallel(...)`	Execute sgems commands in parallel. Return the execution group id.	
`wait(...)`	Wait sgems commands running in parallel.	
misc		
`delete_connections(...)`	Delete connections between the nodeids of two equal-sized vectors	
`reset_connections(...)`	Delete all connections on the grid, and restore default connections using face contacts.	
`nan(...)`	Return the ar2gems value for NAN.	`[nanval] = sgems.nan()`

Python no AR2GeMS

Plugins

Os plugins

Um plugin python para o ar2gems é formado de dois arquivos:

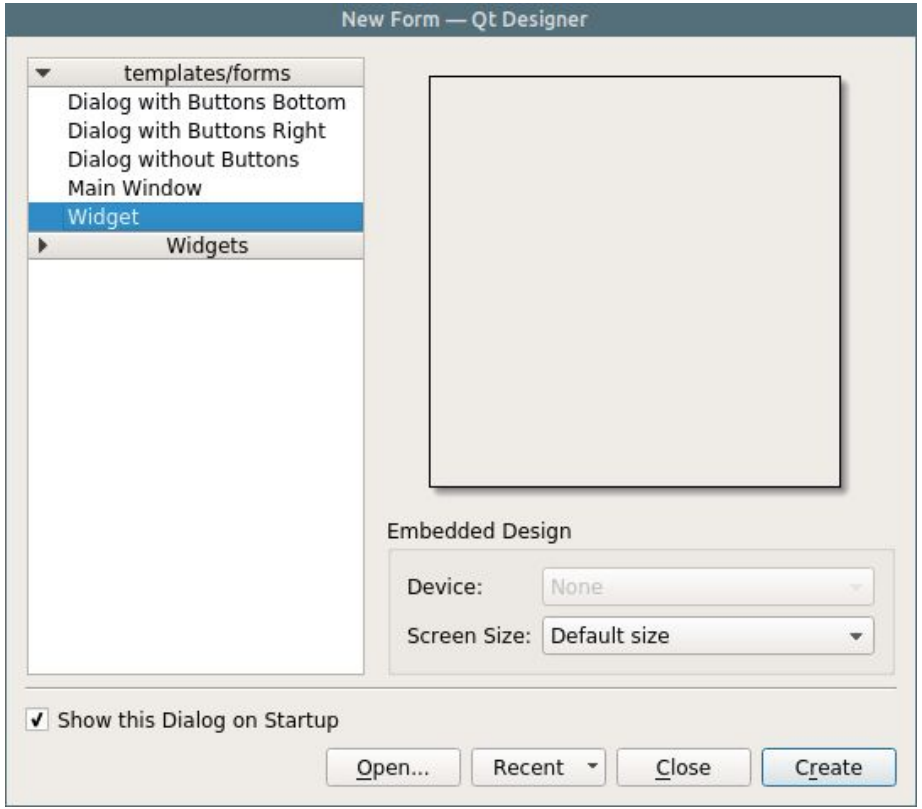
1. Arquivo .py: Código python;
2. Arquivo .ui: Código xml que determina a interface do plugin;

Instalação

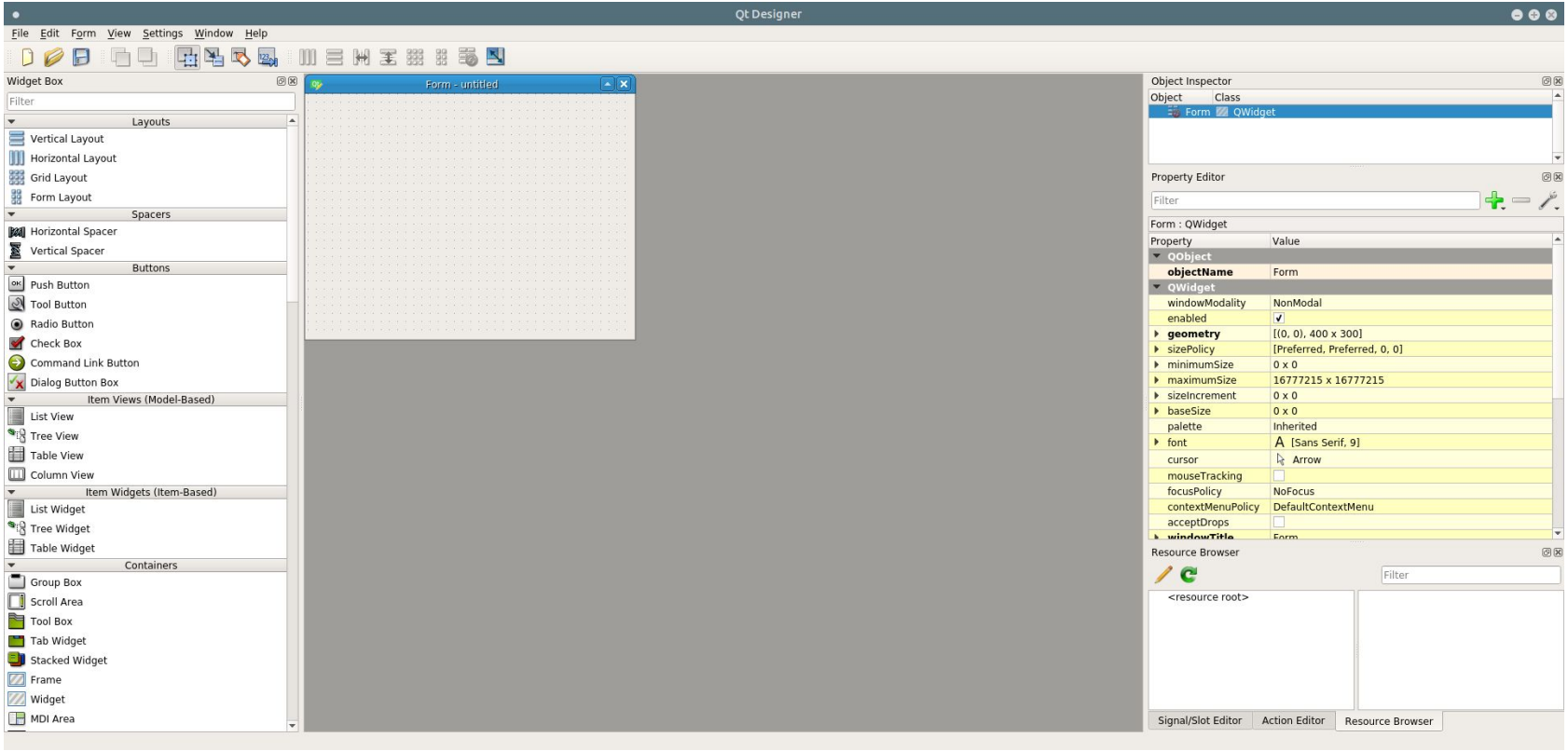
Os arquivos devem ser colocados em suas respectivas pastas:

1. Arquivo .py:
    a. No Linux: opt/ar2gems/bin/plugins/Geostat/python/
    b. No Windows: C:\SGeMS-ar2tech-x64\plugins\Geostat\python
2. Arquivo .ui: Código xml que determina a interface do plugin;
    a. No Linux: opt/ar2gems/bin/plugins/Geostat/
    b. No Windows: C:\SGeMS-ar2tech-x64\plugins\Geostat

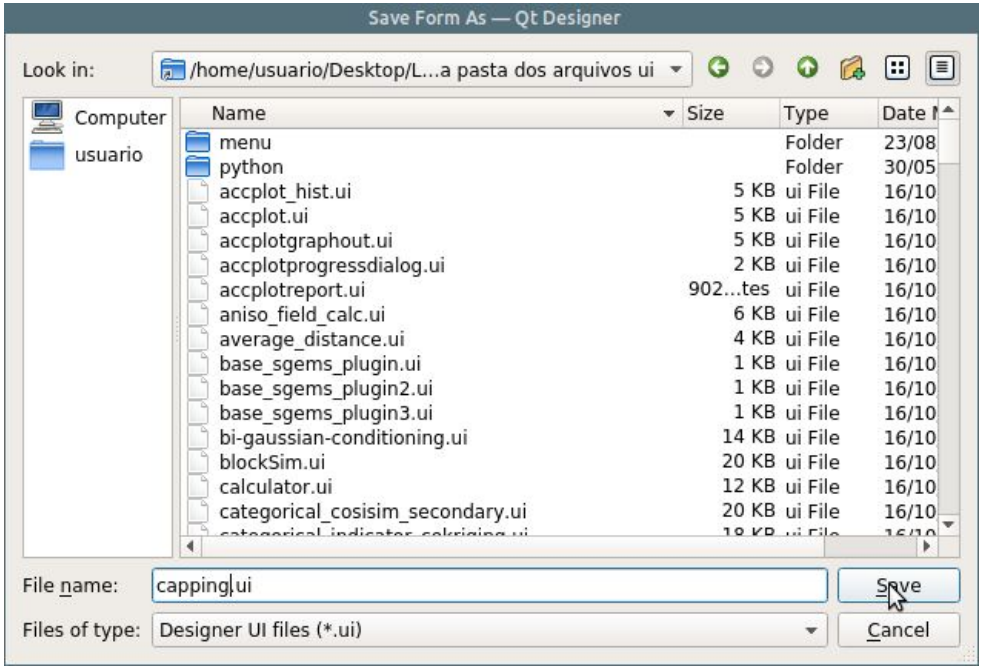
Criando a interface gráfica



Área de trabalho do Qt



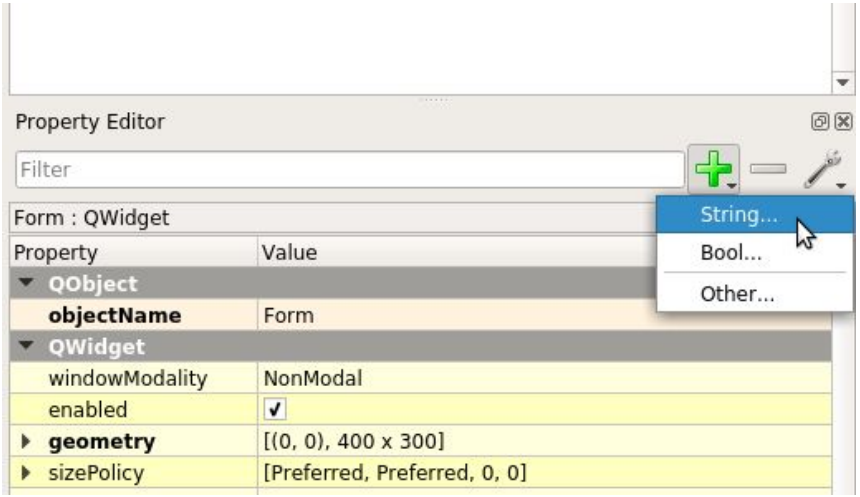
Salvando seu arquivo .ui

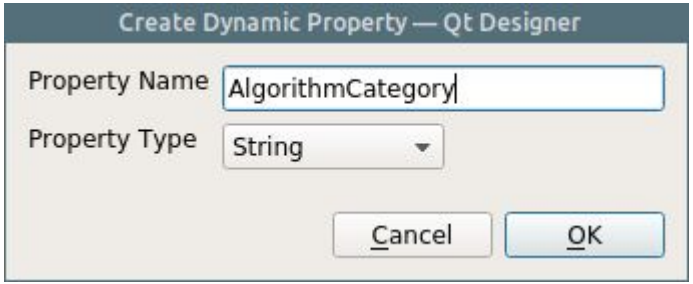


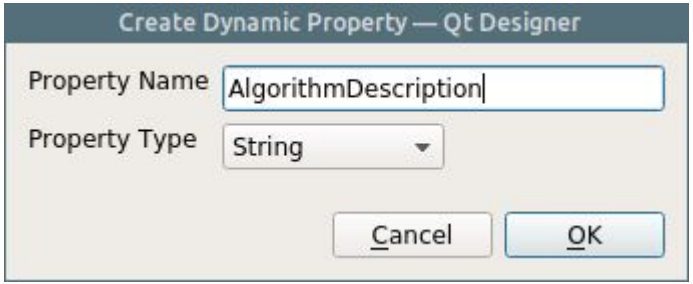
O arquivo deve ser salvo na pasta dos arquivos .ui.

O arquivo .py deve ter o mesmo nome.

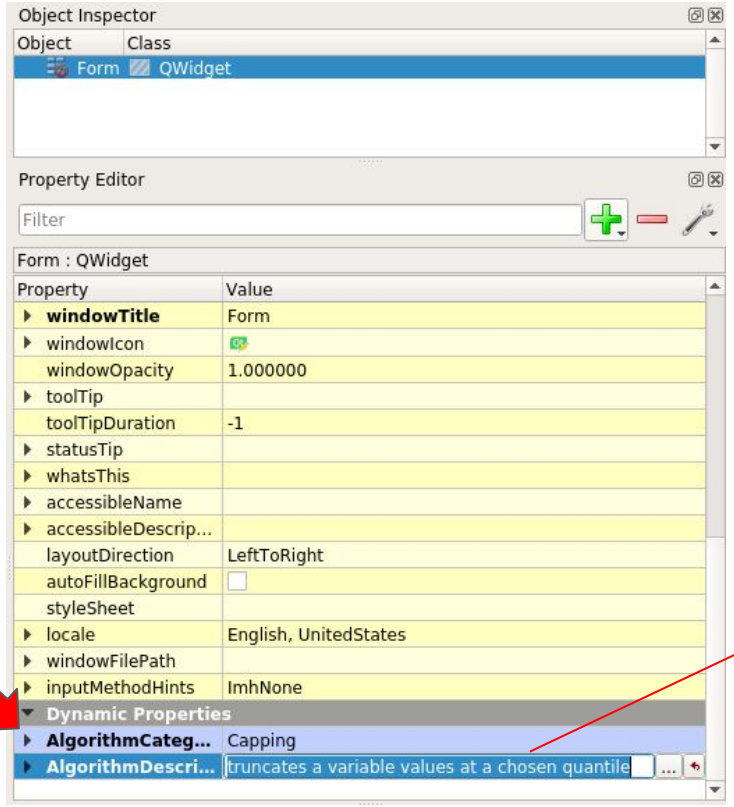
Variáveis AlgorithmCategory e AlgorithmDescription

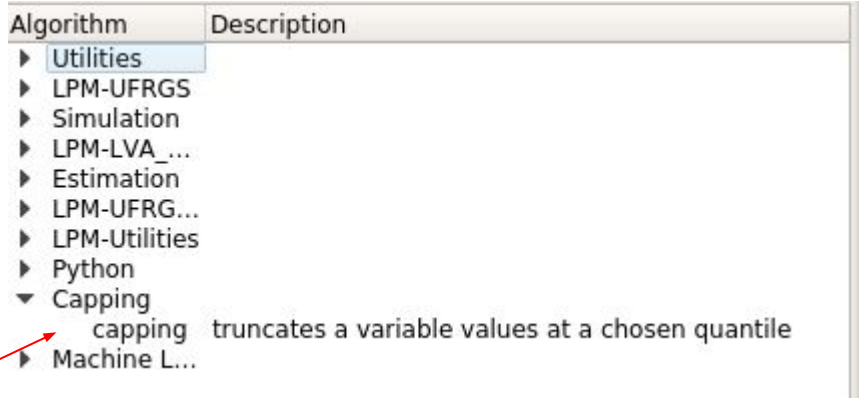






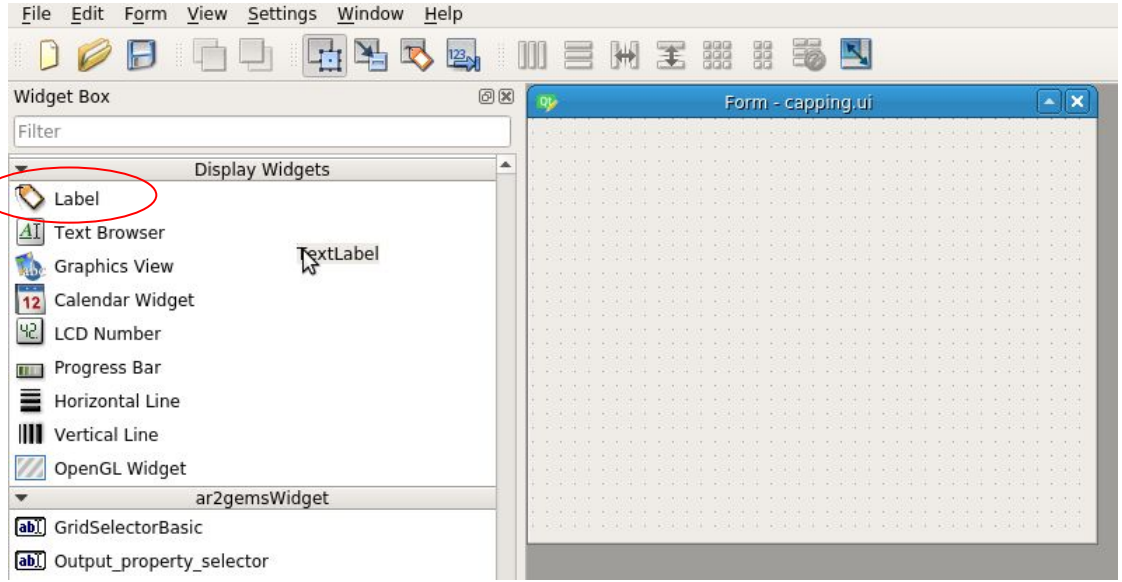
Editando as variáveis criadas



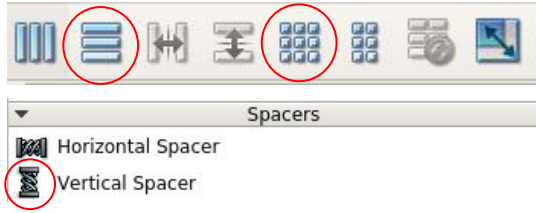


Inserindo widgets

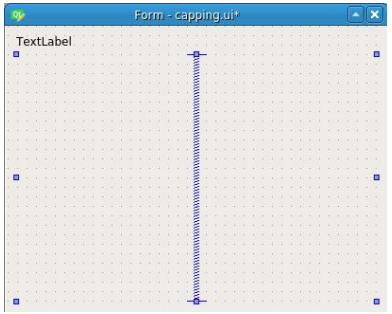
O primeiro wiget a ser inserido e um label, que vai dar uma instrução ao usuário. Arraste o widget escolhido para o plugin.



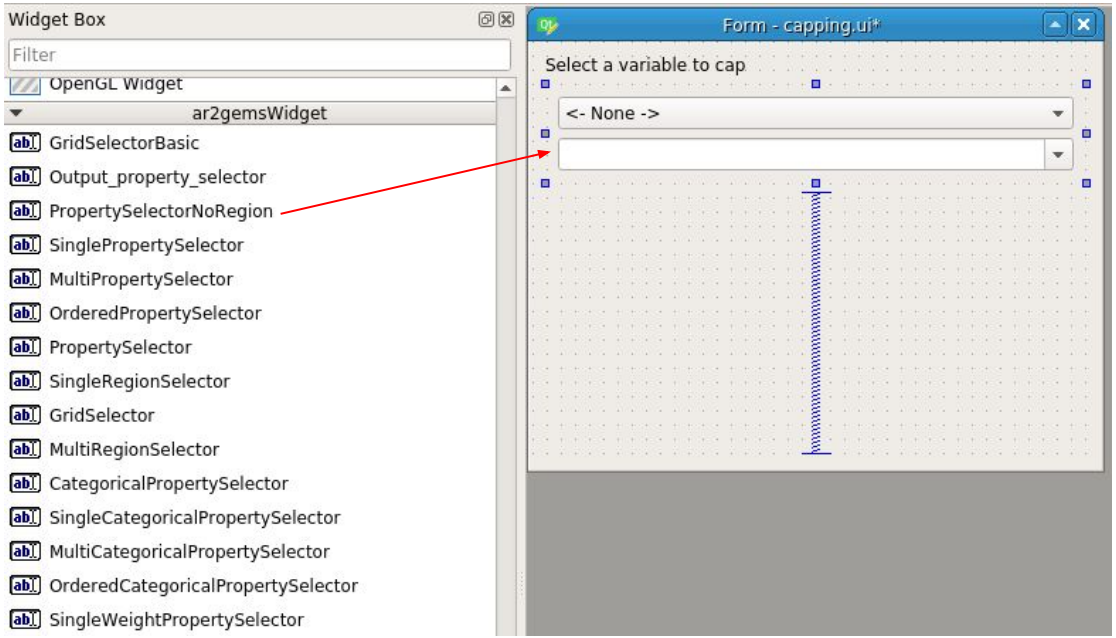
Layouts

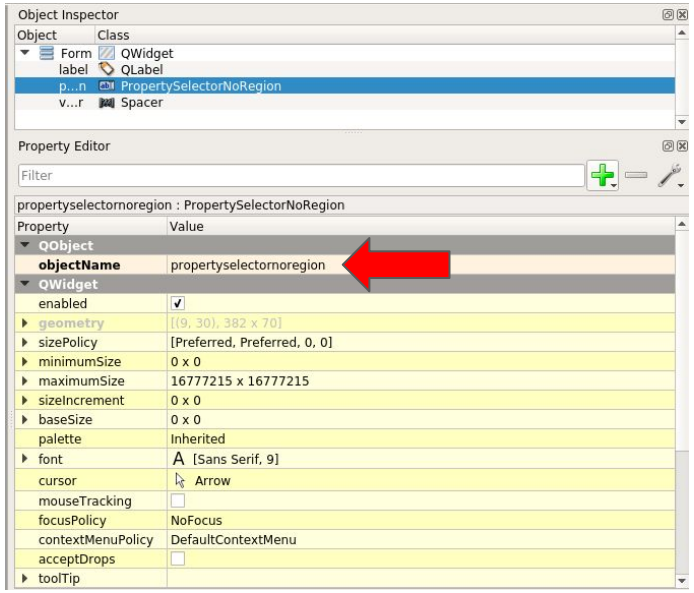


Os layouts organizam os widgets no plugin.

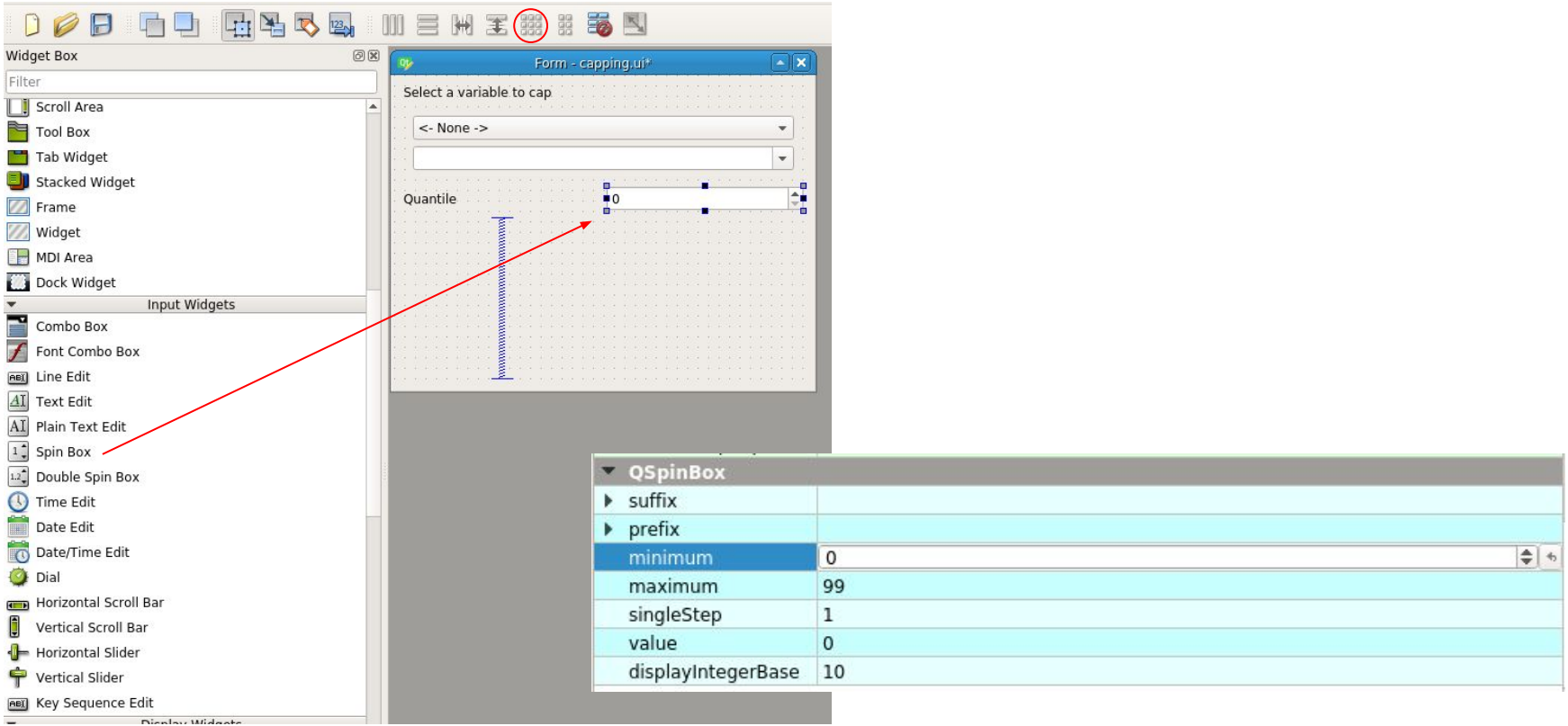


Adicionando mais widgets





Adicionando mais widgets



O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Cabeçalho: aqui eu importo bibliotecas, defino funções, escrevo uma descrição sobre o programa.

O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Classe: aqui vai o nome do plugin que deve ser o mesmo dos arquivos .py e .ui bem como desses campos.

O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Método initialize: o método initialize é invocado quando o usuário clica em Run Algorithm no ar2gems.

Neste método é recebido um dicionário `params` com todos os parâmetros fornecidos na interface

A linha de comando `self.params = params` copia os parâmetros de entrada para usar no `execute()`

O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Método execute: é invocado após o initialize. Aqui deve ser feito todo o processamento do plugin.

O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Esse bloco não é utilizado mas não pode ser removido.

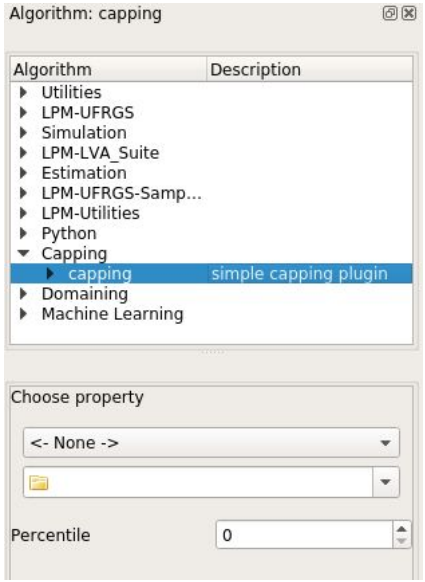
O arquivo .py

```
#!/bin/python

#####################################

class plugin:
      def __init__(self):
            pass

#####################################

      def initialize(self, params):
            Self.params = params
            return True

#####################################

      def execute(self):

            return True

#####################################

      def finalize(self):
            return True

      def name(self):
            return "plugin"

#####################################

def get_plugins():
      return ["plugin"]
```

Essa função registra o plugin na lista de plugins ativos do ar2gems.

Editando a arquivo .py

1. Abra o template e altere o nome do plugin nos campos necessários;
2. Salve na pasta correta alterando o nome do arquivo .py;
3. Abra o ar2gems (`sh run_ch_ar2gems.sh` no diretório home)

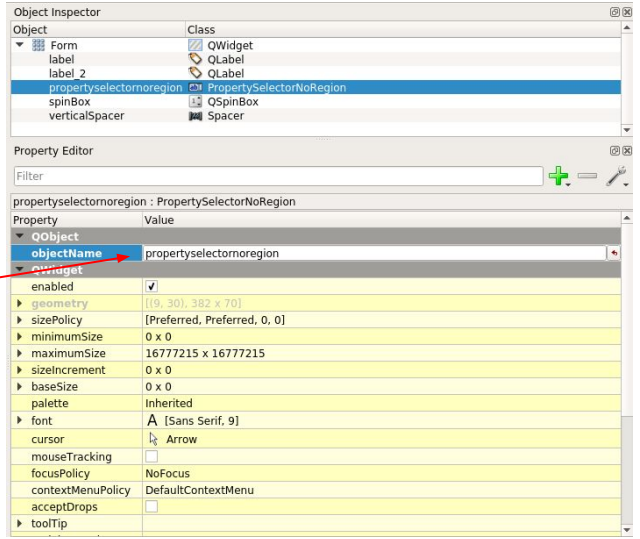


Se você fez tudo certo vai visualizar o plugin no painel de algoritmos.

Dicionário de parâmetros

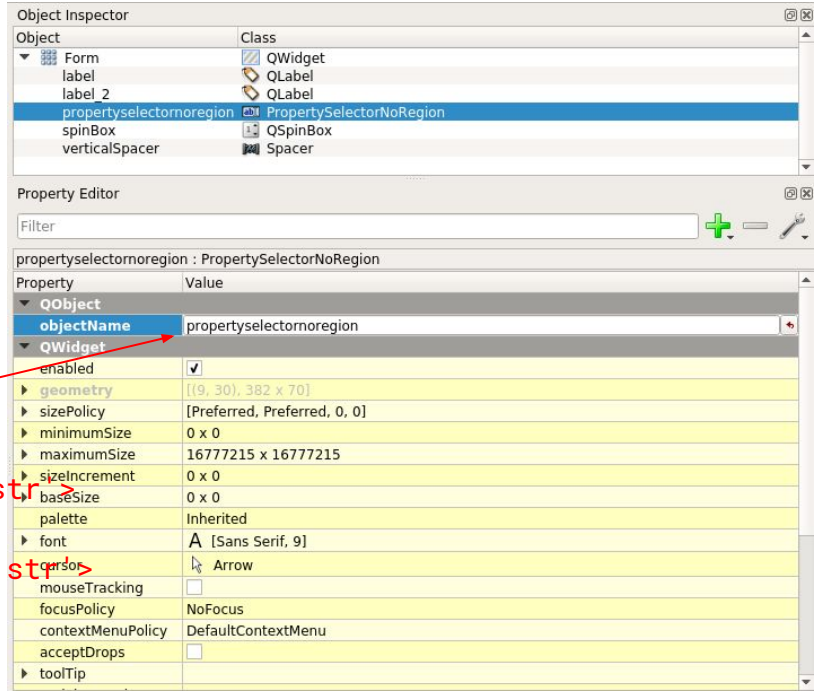
A variável de classe `self.params` é um dicionário dos parâmetros imputados na interface gráfica.

{'xml': '<parameters>\n <algorithm name="capping"/>\n <propertyselectornoregion property="V" grid="walker"/>\n <spinBox value="45"/>\n</parameters>\n',
'spinBox': {'value': '45'},
'propertyselectornoregion': {'property': 'V', 'grid': 'walker'},
'algorithm': {'name': 'capping'}}



Função read_params

```
### Printing GUI parameters ###

['xml']=<parameters>
<algorithm name="capping"/>
<propertyselectornoregion grid="" property=""/>
<spinBox value="0"/>
</parameters>
type: <type 'str'>

['spinBox']['value']=45 type: <type 'str'>

['propertyselectornoregion']['property']=V type: <type 'str'>

['propertyselectornoregion']['grid']=walker type: <type 'str'>

['algorithm']['name']=capping type: <type 'str'>
```



Programando seu plugin

1. “Pegando” a variável:
    a. sgems.get_property(“walker”, “V”)
    b. “walker” <- self.params['propertyselectornoregion']['grid']
    c. “V” <- self.params['propertyselectornoregion']['property']

```
grid_string = self.params['propertyselectornoregion']['grid']

prop_sting = self.params['propertyselectornoregion']['grid']

prop = sgems.get_property(grid_string, prop_string)
```

É mais eficiente trabalhar com numpy arrays, então transforme sua lista, mas lembre-se de reconverter para lista antes de devolver para o grid.

Programando seu plugin

1. “Pegando” o percentil:
    a. “45” <- self.params['spinBox']['value']

```
percentil = int(self.params['spinBox']['value'])
```

Atualizando o plugin

Rode o comando `sgems.execute("ReloadPythonPlugins")` no script editor.

Numpy

Vamos à documentação:

1. `numpy.percentile()`
2. `numpy.where()`

Terminando o seu plugin

1. “Pegue” as variáveis a partir da GUI;
2. Use as funções numpy para criar uma nova variável;
3. Devolva essa nova variável para o grid (como uma lista).

Profiling

Quando enfrentamos problemas de performance

Biblioteca time

```
Import time
```

`time.time()` retorna o tempo naquele instante

```
t1 = time.time()

...bloco de código

t2 = time.time()

print('O tempo de execucao do bloco e {}'.format(t2-t1)
```

Exercício

1. Usando o script editor do ar2gems veja quanto demora pra criar uma nova variável com capping;
2. Faça o mesmo usando o `np.where()`.