Construindo um plugin de interface do software histplt do GSLIB para o AR2GeMS

Áttila Leães Rodrigues
attila.leaes@gmail.com
LPM-UFRGS 2016

25 de Agosto de 2016

Conteúdo

Capítulo 1

Introdução

O ambiente de desenvolvimento de plugins para o AR2GeMS é uma excelente ferramenta para o desenvolvimento de programas que estendem a funcionalidade do núcleo central de recursos do AR2GeMS (o que é a função de um plugin). Com o novo recurso da utilização da linguagem Python para a criação de plugins torna-se mais fácil a implementação de novos plugins.

Outra possibilidade que pode ser explorada com as ferramentas de plugins é o interfaceamento de programas no ambiente gráfico do AR2GeMS. Um exemplo são os programas que compõem o GSLIB. O interfaceamento de um programa do GSLIB na interface gráfica do AR2GeMS significa que o usuário poderá clicar no plugin correspondente a um programa do GSLIB no ambiente do AR2GeMS, digitar os parâmetros necessários, logo em seguida o plugin executará o programa e tratará os resultados.

Nesse documento apresento um exemplo com fins didáticos de um interfaceamento para o programa `histplt` do GSLIB. Além de apresentar a estratégia do planejamento do plugin de interfaceamento discuto boas práticas para a construção de plugins.

O código-fonte do plugin implementado encontra-se no endereço `https://github.com/LPM-UFRGS/lpm-python-course-projects/tree/master/plugin_examples/gslib-histplt-interface`.

Capítulo 2

Objetivo do trabalho

O objetivo do trabalho foi construir um plugin que faça a interface do programa `histplt` para o software AR2GeMS. Também é um objetivo o documentação deste processo de implementação e estabelecer boas práticas para outras implementações de plugins com interfaceamento de programas terceiros.

Capítulo 3

O programa histplt

3.1 Obtendo e compilando os programas do pacote GSLIB

A página do GSLIB se encontra no endereço:

`http://www.gslib.com.`

E o link para a seção de downloads se encontra em

`http://www.statios.com/Quick/gslib.html.`

Neste documento será comentada a instalação a partir dos códigos-fonte em um sistema GNU/Linux (como na máquina virtual do curso).

Os códigos-fonte baixados foram do Fortran 90 para Linux, link:

`http://www.statios.com/software/gslib90_ls.tar.gz.`

O principal requisito para a produção dos executáveis GSLIB é a presença de um compilador Fortran no sistema. Usualmente no Linux utilizamos o `gfortran`, que será usado como exemplo.

Depois de descompactar o arquivo `gslib90_ls.tar.gz` altere o arquivo `Makefile` dentro da pasta `gslib90` para determinar qual o compilador Fortran que será utilizado. Altere a linha do arquivo como no exemplo a seguir:

```
1 FC=gfortran
```

Depois execute o comando `make` para iniciar a compilação.

Depois da compilação, é importante o usuário anotar a pasta completa onde estão os executáveis para incluir no código-fonte do plugin. O comando (no Linux) para exibir o caminho completo de uma pasta corrente é o `pwd`. Como no exemplo:

```
attila@hydra:~/ar2-gslib/gslib90$ pwd
/home/attila/ar2-gslib/gslib90
```

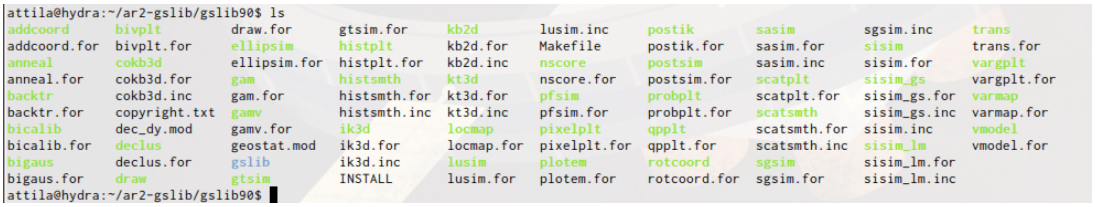

Figura 3.1: Figura com os arquivos-fonte e executáveis do GSLIB depois da compilação.

3.2 Parâmetros de execução do histplt

Os parâmetros de execução do `histplt` são determinados através de um arquivo que usualmente tem a terminação "`.par`".

Na listagem 3.1 é mostrado o conteúdo de um arquivo de parâmetros válido para uso durante e execução do programa `histplt`. Cada linha é identificada de acordo com a sua numeração. A função de cada linha será descrita depois da listagem.

Listing 3.1: Exemplo de arquivo de parâmetros para o programa `histplt`.

```
1  Aquivo de parametros para teste
2  START OF PARAMETERS:
3  data_1.dat
4  1 0
5  -1 1
6  saida.ps
7  -2 2
8  -1
9  25
10 0
11 1
```

```
-1
2
TITULO DA FIGURA
0.5
3.1415926535
```

1. Linha que pode conter qualquer informação, não é utilizada durante a execução;

2. Define o início dos parâmetros na linha subsequente;

3. Nome do arquivo de dados no formato GSLIB. É recomendado utilizar o caminho completo do arquivo;

4. Dois valores: a coluna que contém os dados e em seguida a coluna que contém os pesos;

5. Dois valores: os dados menores que o primeiro valor e maior que o segundo serão ignorados. Exemplo se esta linha for 12 logo todos os dados menores que 1 e maiores que 2 serão suprimidos;

6. Esta linha define o nome do arquivo ".ps" de saído com a figura do histograma. Pode ser utilizado um caminho completo para este arquivo, como /home/user/saida.ps (Linux) ou C:\dados\saida.ps (Windows);

7. Dois valores: xmin e xmax que serão utilizados na figura do histograma. Caso xmin for maior que xmax então o eixo x será estabelecido a partir dos dados;

8. Frequência máxima (eixo y) do histograma. Caso seja fornecido um valor menor que zero, essa frequência máxima será determinada automaticamente;

9. Número de classes;

10. Escala no eixo x: 0 para escala aritmética, 1 para escala logarítmica base 10;

11. Tipo de histograma: histograma de frequências (0) ou histograma acumulativo (1);

12. Número de quantis a serem plotados para o caso de histograma acumulativo;

13. Número de casas decimais para as informações na caixa de dados estatísticos, 0 para automático;

14. Título. Até 40 caracteres;

15. Posição da caixa de estatísticas. Valor de -1 a 1;

16. Valor de referência. A linha deve estar em branco caso não seja desejado um valor para referência.

3.3 Formato do arquivo de dados do gslib

O programa `histplt` lê um arquivo de dados no formato usual do GSLIB. Este formato especifica que:

1. A primeira linha seja uma descrição dos dados;

2. A segunda linha seja um número que explicita a quantidade de colunas;

3. Um "label" para cada coluna, da esquerda para a direita;

4. Nas linhas seguintes residem os dados, que podem ser separados por espaços em branco ou por um "tab".

Um exemplo de arquivo de dados é mostrado na figura 3.2.

```
Data Example
2
X variable
Y variable
0.0488566790561 0.76045507733
0.509783209681 0.230452348659
0.317285391041 0.861030891104
0.51656032276 0.562299990089
0.499199888524 0.353370838922
0.15624706761 0.771860545339
0.995119905767 0.0858321988321
0.189612453338 0.639003386556
0.423918310304 0.68658036078
0.89052353307 0.718779287713
0.914833276512 0.77587819044
0.684705431684 0.248479878867
0.18012260214 0.224019797649
0.432085593681 0.538592624705
0.947239994905 0.277421109744
```

Figura 3.2: Exemplo de arquivo de dados utilizado pelo GSLIB.

3.4 Execução do programa **histplt**

Um exemplo de comando para a execução do `histplt` no ambiente Linux é

```
1 /home/attila/ar2-gslib/gslib90/histplt /home/attila/file.par
```

No exemplo acima as informações do nome do arquivo de saída e sua localização estão inseridas no arquivo `.par`. É possível executar o `histplt` sem informar o caminho completo do programa e do arquivo `.par`, porém o comando deve ser executado a partir da pasta com o executável e o arquivo `.par`. Como no exemplo:

```
1 ./histplt file.par
```

, no Linux quando vamos executar um programa na mesma pasta precisamos informar ao sistema que o executável está na pasta corrente. Fazemos isso usando os caracteres `./`.

No caso de um script para o AR2GeMS é importante sempre trabalhar com o caminho completo dos arquivos. Isso porque nem sempre temos o poder de decidir como o programa será executado, ou seja, a partir de qual pasta.

Capítulo 4

Anatomia do plugin

A estratégia para a implementação de uma interface para o programa `histplt` foi criada a partir do entendimento da execução do programa `histplt`. Para a execução do programa deve existir um arquivo de dados e um arquivo de parâmetros. Logo o nosso plugin de interface deve criar esses arquivos necessários caso eles não existam. Foi desenvolvido então dois modos de execução do plugin: o primeiro modo serve para o caso do usuário ter os dados de entrada em um arquivo (modo "with file"). Nesse modo "with file" o plugin irá criar somente o arquivo de parâmetros e utilizará um arquivo com dados fornecido pelo usuário. A saída do plugin é na forma de um arquivo postscript gerado pelo `histplt`, o nome do arquivo de saída deve ser informado pelo usuário na interface gráfica do plugin.

O outro modo de execução é o "with property". Este modo aproveita realmente a interface gráfica oferecida pelo AR2GeMS. Desta vez o usuário escolhe uma propriedade e o plugin cria os dois arquivos necessários para a execução do `histplt`. Em seguida o plugin executa o programa `histplt` usando os arquivos criados e exibe os dados de saída.

De uma maneira geral, um plugin de interface para um programa externo usualmente é elaborado baseando-se nas necessidades da execução do programa em questão (no nosso caso, o `histplt`) e finalmente no tratamento dos resultados que o programa fornece como output. Então podemos estabelecer esse procedimento como um método geral para criar um plugin de interface.

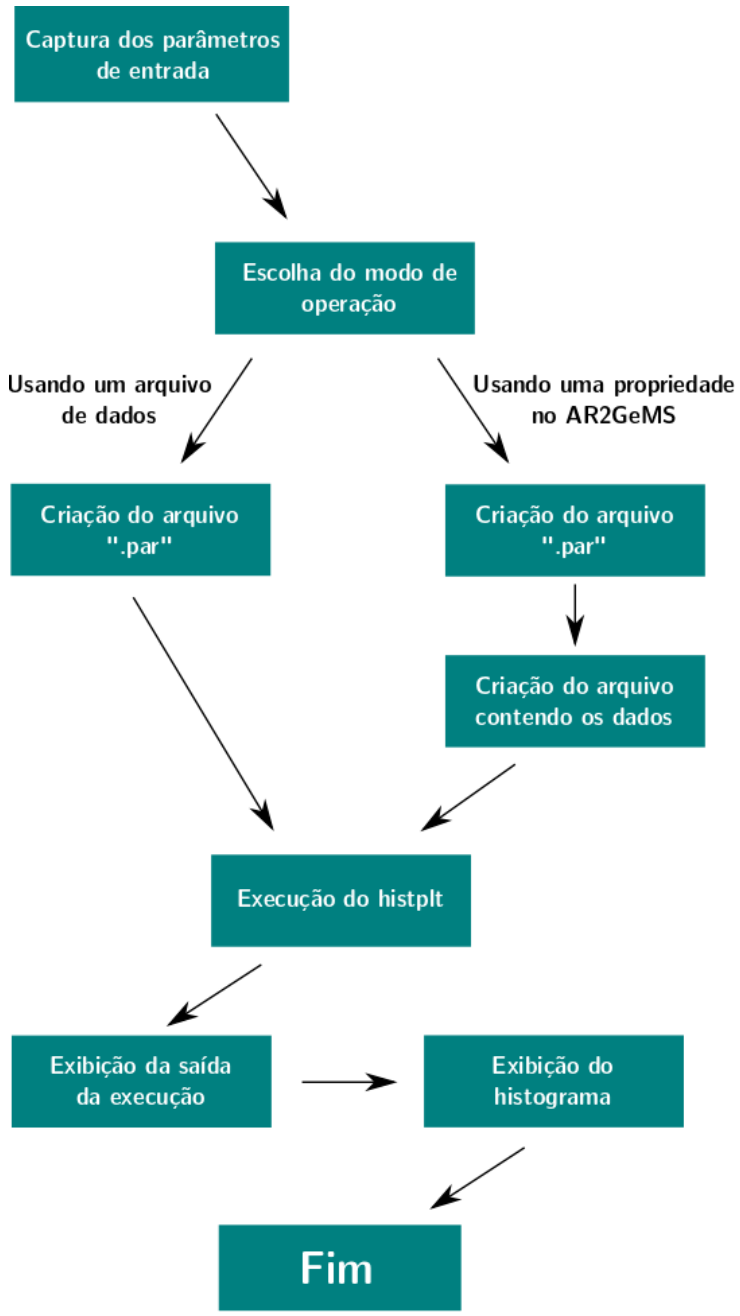

Figura 4.1: Fluxograma das operações internas do plugin de interface do programa `histplt` no AR2GeMS.

Link para o repositório onde está o plugin de interface para o `histplt`: `https://github.com/LPM-UFRGS/lpm-python-course-projects/tree/master/plugin_examples/gslib-histplt-interface`.

Capítulo 5

Escrevendo o plugin

5.1 Uma classe em python para o plugin

Foi escolhida uma implementação de uma classe com métodos que realizam as operações necessárias para o interfaceamento do `histplt`. Essa classe chama-se `histplugin`.

5.2 Parâmetros que precisam ser inicializados

Na classe `histplugin` alguns parâmetros precisam ser inicializados para a correta execução do plugin, são eles:

- `self.dict_gen_params['gslib_executables_folder']`
  Esta variável precisa ser inicializada com uma string com o caminho completo da pasta onde se encontra o executável `histplt`.
- `self.dict_gen_params['gslib_hist_executable']`
  Nome do executável, no caso `histplt`.
- `self.dict_gen_params['work_folder']`
  Pasta de trabalho onde serão gravados os arquivos temporários.
- `self.dict_gen_params['par_file']`
  Nome do arquivo temporário `.par`.

- `self.dict_gen_params['display_command']`
  String com o comando para visualizar um arquivo postscript (saída do programa `histplt`).

- `self.dict_gen_params['path_separator']`
  Separador de pastas para um caminho completo. No ambiente Linux o separador é o caractere "/", no Windows é o caractere "\".

- `self.dict_gen_params['datafilename_with_property']`
  Nome do arquivo temporário com os dados gerados a partir de uma propriedade do AR2GeMS.

5.3 Capturando os parâmetros de execução

Na primeira etapa do plugin é realizada a captura dos parâmetros de entrada. De acordo com o diagrama na figura 4.1 (página 12) primeiramente é decidido o modo de execução do plugin.

O plugin é dividido em três abas. Na primeira aba o usuário escolhe o modo de operação. Uma imagem desta aba é mostrada na figura 5.1. O usuário então escolhe o modo de execução através de um grupo de dois widgets do tipo radio button. O primeiro modo é o "with file" e o segundo modo de operação é o "with parameter".

Caso o usuário escolha o modo "with file" ele vai escrever os parâmetros na aba com o mesmo nome ("with file"), uma imagem desta aba é mostrada na figura 5.2 (página 17). Caso o usuário escolha o modo "with property" ele vai escrever os parâmetros de entrada na aba "with-property", figura 5.3 na página 18.

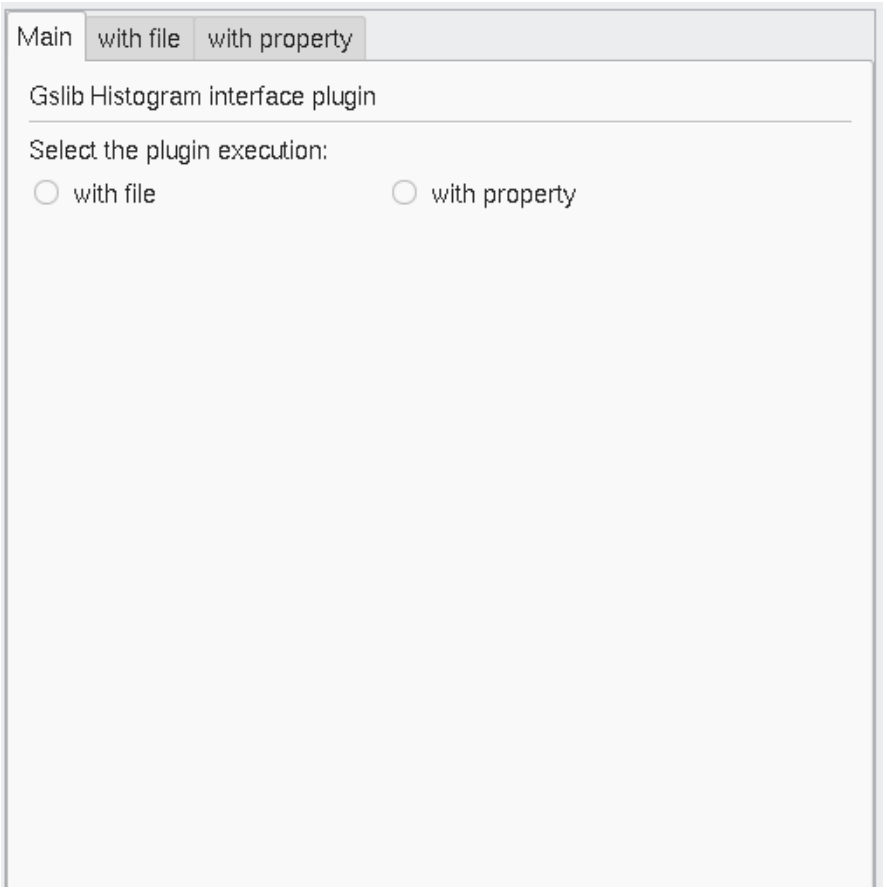

Figura 5.1: Seleção do modo de operação do plugin.

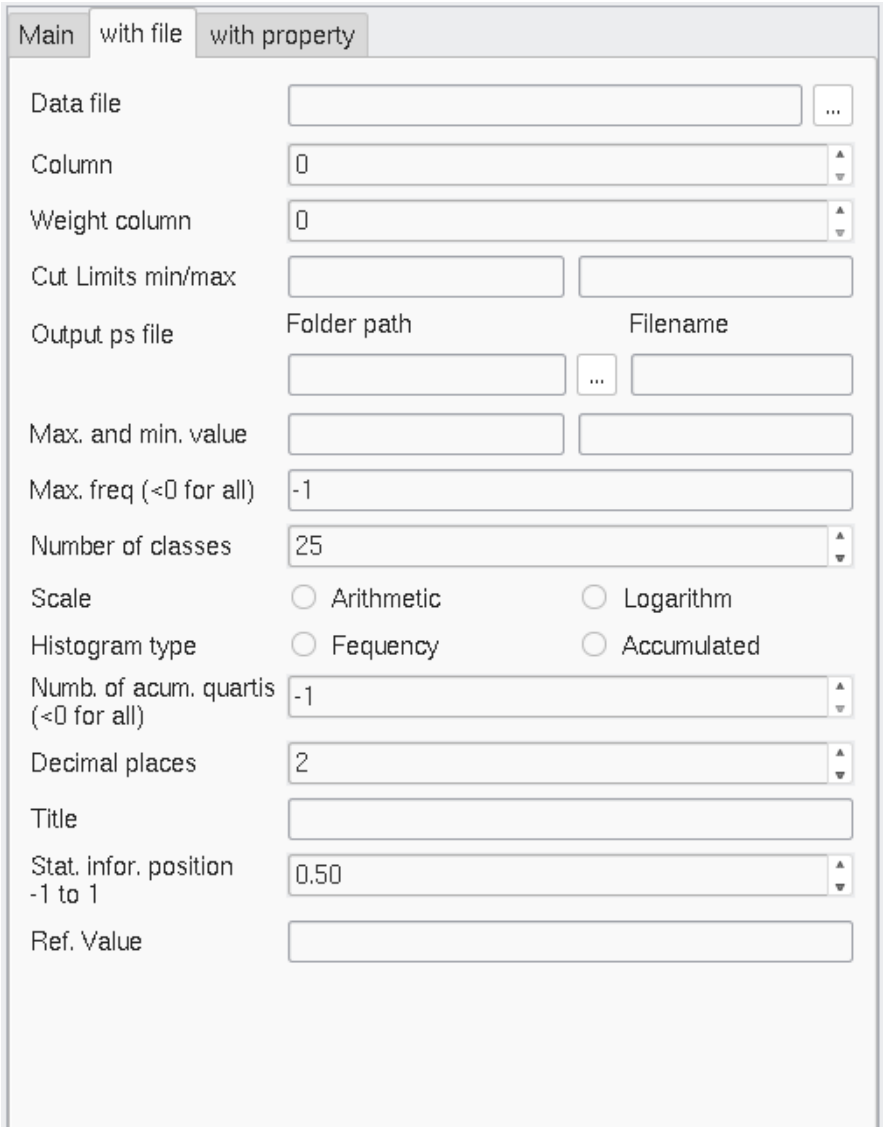
Main
with file
with property
Data file
...
Column
0
Weight column
0
Cut Limits min/max
Output ps file
Folder path
Filename
...
Max. and min. value
Max. freq (<0 for all)
-1
Number of classes
25
Scale
Arithmetic
Logarithm
Histogram type
Fequency
Accumulated
Numb. of acum. quartis (<0 for all)
-1
Decimal places
2
Title
Stat. infor. position -1 to 1
0.50
Ref. Value

Figura 5.2: .

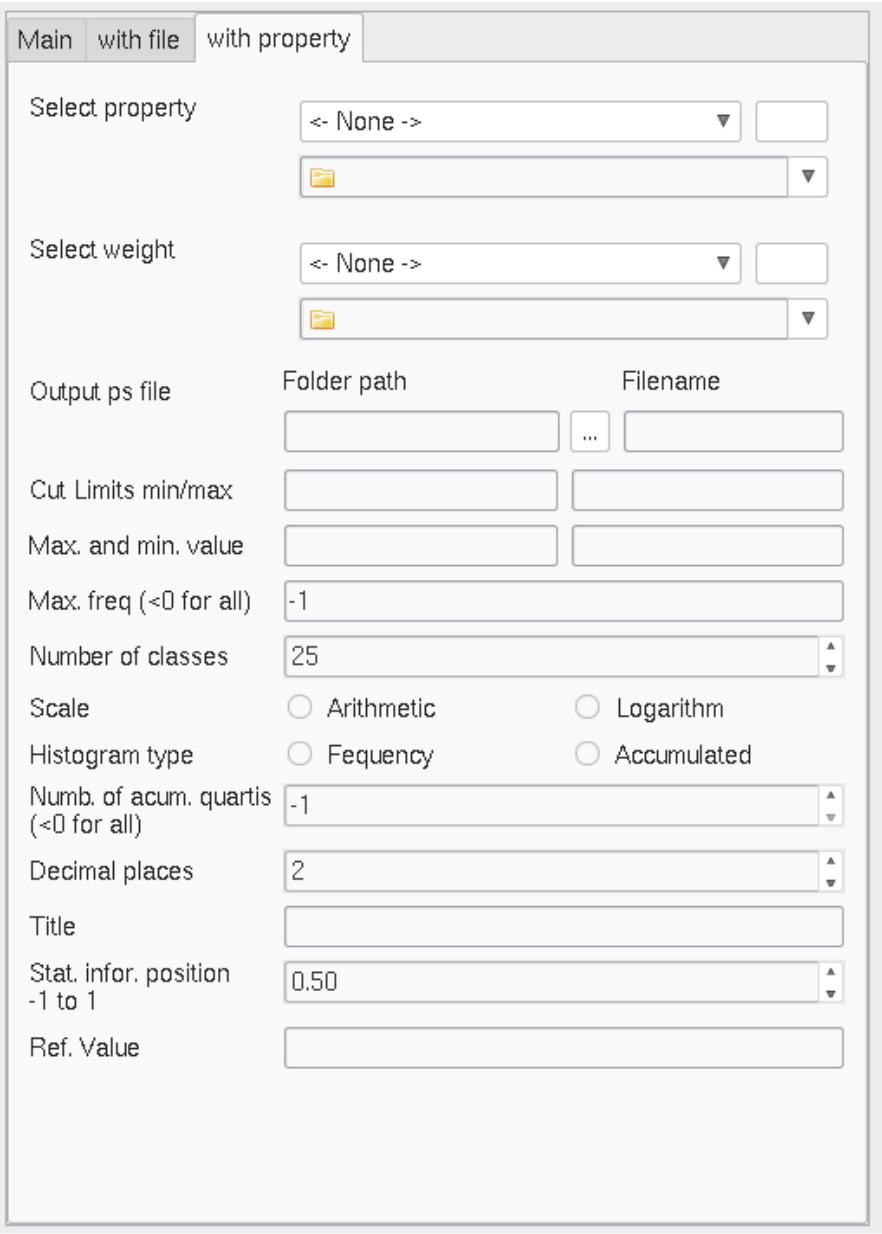
Main
with file
with property
Select property
<- None ->
Select weight
<- None ->
Output ps file
Folder path
Filename
...
Cut Limits min/max
Max. and min. value
Max. freq (<0 for all)
-1
Number of classes
25
Scale
Arithmetic
Logarithm
Histogram type
Fequency
Accumulated
Numb. of acum. quartis (<0 for all)
-1
Decimal places
2
Title
Stat. infor. position -1 to 1
0.50
Ref. Value

Figura 5.3: .

- Definição do modo de operação

O programa foi idealizado com a utilização de um objeto instanciado a partir de uma classe `histplugin`. Um objeto desta classe tem um dicionário `dict_gen_params` e a chave desse dicionário que guarda a informação de qual modo de operação será executado é a chave 'execution_type'. Logo no código da classe `histplugin` o modo de operação é atribuído ou consultado usando `self.dict_gen_params['execution_type']`.

Na listagem de código a seguir é mostrado o primeiro bloco de linhas do método `get_gui_input` da classe `histplugin`. Este método é responsável por colocar as informações digitadas pelo usuário na gui ("graphical user interface") nos dicionários do objeto. No caso da próxima listagem de código, o programa verifica qual modo de operação o usuário deseja executar e guarda essa informação.

```
1  def get_gui_input(self):
2    if (self.params['radioButton_5']['value']=='1' and self.params['↩
         ↪ radioButton_6']['value']=='0'):
3      self.dict_gen_params['execution_type']='with_file'
4    elif (self.params['radioButton_5']['value']=='0' and self.params['↩
         ↪ radioButton_6']['value']=='1'):
5      self.dict_gen_params['execution_type']='with_property'
6    elif (self.params['radioButton_5']['value']=='0' and self.params['↩
         ↪ radioButton_6']['value']=='0'):
7      print "Plugin execution type not set.."
8      self.dict_gen_params['execution_status']="ERROR"
9    else:
10     print "Error in execution type parameters."
11     self.dict_gen_params['execution_status']="ERROR"
```

É importante mencionar que é realizada uma verificação dos parâmetros utilizados pelo usuário. Se forem parâmetros inválidos uma chave do dicionário `self.dict_gen_params['execution_status']` é atribuída com valor `'ERROR'`.

- Definição coletando as entradas para o modo de operação `'with_file'`

Depois de definido o modo de operação, o programa coletará os dados inseridos pelo usuário na aba relacionada ao referido modo de operação. Se

o modo de operação escolhido for o `with_file` o programa executará (ainda dentro do método `get_gui_input`) o seguinte bloco de código:

```
    (ainda dentro do get\_gui\_input)
    ### 'WITH FILE' INPUTS ###
    if self.dict_gen_params['execution_type']=='with_file':

      #input defined by the widget: a spinbox with -1 to 99 integers, default -1
      self.dict_with_file['number_of_quantiles']=self.params['spinBox_3']['value']
      # filechooser widget, must input a valid folder
      self.dict_gen_params['outputfile_folder']=self.params['filechooser_2']['value']
      # any name
      self.dict_gen_params['outputfile_name']=self.params['lineEdit_8']['value']
      # data file defined by the filechooser widget
      self.dict_with_file['datafile']=self.params['filechooser']['value']
      #
      self.dict_with_file['number_of_classes']=self.params['spinBox_2']['value']
      self.dict_with_file['pos_stat']=self.params['doubleSpinBox']['value']
      self.dict_with_file['decimal_places']=self.params['spinBox_4']['value']
      self.dict_with_file['column_data']=self.params['spinBox']['value']
      self.dict_with_file['weight']=self.params['spinBox_5']['value']
      self.dict_with_file['lim_cut_min']=self.params['lineEdit']['value']
      self.dict_with_file['lim_cut_max']=self.params['lineEdit_2']['value']
      self.dict_with_file['value_min']=self.params['lineEdit_4']['value']
      self.dict_with_file['value_max']=self.params['lineEdit_3']['value']
      self.dict_with_file['title']=self.params['lineEdit_5']['value']
      self.dict_with_file['max_frequency']=self.params['lineEdit_7']['value']
      self.dict_with_file['reference_value']=self.params['lineEdit_6']['value']

      if (self.params['radioButton']['value']=='0' and self.params['radioButton_2']['value']=='1'):
```

```
28        self.dict_with_file['scale']='0'
29      elif (self.params['radioButton']['value']=='1' and self.params['↩
            ↪ radioButton_2']['value']=='0'):
30        self.dict_with_file['scale']='1'
31      elif (self.params['radioButton']['value']=='0' and self.params['↩
            ↪ radioButton_2']['value']=='0'):
32        print "Scale parameter not set."
33        self.dict_gen_params['execution_status']="ERROR"
34      else:
35        print "Error in scale parameters."
36        self.dict_gen_params['execution_status']="ERROR"
37
38      if (self.params['radioButton_4']['value']=='1' and self.params['↩
            ↪ radioButton_3']['value']=='0'):
39        self.dict_with_file['histogram_type']='0' #frequency rb4
40      elif (self.params['radioButton_4']['value']=='0' and self.params↩
            ↪ ['radioButton_3']['value']=='1'):
41        self.dict_with_file['histogram_type']='1' #acum rbutton_3
42      elif (self.params['radioButton_4']['value']=='0' and self.params↩
            ↪ ['radioButton_3']['value']=='0'):
43        print "Histogram type not set."
44        self.dict_gen_params['execution_status']="ERROR"
45      else:
46        print "Error in histogram type parameters."
47        self.dict_gen_params['execution_status']="ERROR"
```

Para o caso do modo de operação ser `'with_property'` outro bloco de código é executado:

```
1     (ainda dentro do get\_gui\_input)
2    ############ WITH PROPERTY INPUTS ##########
3     if self.dict_gen_params['execution_type']=='with_property':
4
5       self.dict_with_property['weight_region']=self.params['↩
            ↪ propertyselector_2']['region']
6       self.dict_with_property['weight_property']=self.params['↩
            ↪ propertyselector_2']['property']
7       self.dict_with_property['weight_grid']=self.params['↩
            ↪ propertyselector_2']['grid']
8       self.dict_with_property['data_region']=self.params['↩
            ↪ propertyselector']['region']
9       self.dict_with_property['data_property']=self.params['↩
            ↪ propertyselector']['property']
10      self.dict_with_property['data_grid']=self.params['↩
            ↪ propertyselector']['grid']
11      self.dict_gen_params['outputfile_folder']=self.params['↩
```

```
            ↪ filechooser_3']['value']
        self.dict_gen_params['outputfile_name']=self.params['lineEdit_9↩
            ↪ ']['value']
        self.dict_with_property['lim_cut_min']=self.params['lineEdit_10↩
            ↪ ']['value']
        self.dict_with_property['lim_cut_max']=self.params['lineEdit_11↩
            ↪ ']['value']
        self.dict_with_property['value_min']=self.params['lineEdit_12↩
            ↪ ']['value']
        self.dict_with_property['value_max']=self.params['lineEdit_13↩
            ↪ ']['value']
        self.dict_with_property['max_frequency']=self.params['↩
            ↪ lineEdit_14']['value']
        self.dict_with_property['reference_value']=self.params['↩
            ↪ lineEdit_15']['value']
        self.dict_with_property['number_of_quantiles']=self.params['↩
            ↪ spinBox_8']['value']
        self.dict_with_property['number_of_classes']=self.params['↩
            ↪ spinBox_6']['value']
        self.dict_with_property['pos_stat']=self.params['doubleSpinBox_2↩
            ↪ ']['value']
        self.dict_with_property['decimal_places']=self.params['spinBox_7↩
            ↪ ']['value']
        self.dict_with_property['title']=self.params['lineEdit_16']['↩
            ↪ value']

        if (self.params['radioButton_7']['value']=='1' and self.params['↩
            ↪ radioButton_8']['value']=='0'):
          self.dict_with_property['scale']='0'
        elif (self.params['radioButton_7']['value']=='0' and self.params↩
            ↪ ['radioButton_8']['value']=='1'):
          self.dict_with_property['scale']='1'
        elif (self.params['radioButton_7']['value']=='0' and self.params↩
            ↪ ['radioButton_8']['value']=='0'):
          print "Scale parameter not set."
          self.dict_gen_params['execution_status']="ERROR"
        else:
          print "Error in scale parameters."
          self.dict_gen_params['execution_status']="ERROR"

        if (self.params['radioButton_9']['value']=='1' and self.params['↩
            ↪ radioButton_10']['value']=='0'):
          self.dict_with_property['histogram_type']='0'
        elif (self.params['radioButton_9']['value']=='0' and self.params↩
            ↪ ['radioButton_10']['value']=='1'):
```

```
39          self.dict_with_property['histogram_type']='1'
40        elif (self.params['radioButton_9']['value']=='0' and self.params↩
              ↪ ['radioButton_10']['value']=='0'):
41          print "Histogram type not set."
42          self.dict_gen_params['execution_status']="ERROR"
43        else:
44          print "Error in histogram type parameters."
45          self.dict_gen_params['execution_status']="ERROR"
```

5.4 Segunda parte: rotinas para a gravação dos arquivos de parâmetros e de dados

O método `write_par_file` é utilizado para gravar os arquivos necessários para a execução do programa `histplt`. Caso o mode de operação seja o `with_file` será necessário somente a criação do arquivo `.par`. O código para esse procedimento é listado a seguir:

```
1   def write_par_file(self):
2
3     ## first try to open .par file for writing
4     try:
5       file=open(self.dict_gen_params['work_folder']+self.↩
            ↪ dict_gen_params['path_separator']+self.dict_gen_params['↩
            ↪ par_file'],'w')
6     except:
7       print 'Problem opening requested par file for writing.'
8       self.dict_gen_params['execution_status']='ERROR'
9
10    if self.dict_gen_params['execution_status'] != 'ERROR' and self.↩
          ↪ dict_gen_params['execution_type']=='with_file':
11      writebuff="Parameters file generated by histgslib Ar2GeMS plugin↩
            ↪ , "+time.strftime("%c")+".\nPlugin execution type: "+self↩
            ↪ .dict_gen_params['execution_type']+'\n'
12      file.write(writebuff)
13      writebuff="START OF PARAMETERS:\n"
14      file.write(writebuff)
15      writebuff=self.dict_with_file['datafile']+"\n"
16      file.write(writebuff)
17      writebuff=self.dict_with_file['column_data']+" "+self.↩
            ↪ dict_with_file['weight']+"\n"
18      file.write(writebuff)
```

```
19      writebuff=self.dict_with_file['lim_cut_min']+" "+self.↩
            ↪ dict_with_file['lim_cut_max']+"\n"
20      file.write(writebuff)
21      if (self.dict_gen_params['outputfile_folder'] != ''):
22        writebuff=self.dict_gen_params['outputfile_folder']+self.↩
              ↪ dict_gen_params['path_separator']+self.dict_gen_params↩
              ↪ ['outputfile_name']+"\n"
23      else:
24        writebuff=self.dict_gen_params['outputfile_name']+"\n"
25      file.write(writebuff)
26      writebuff=self.dict_with_file['value_min']+" "+self.↩
            ↪ dict_with_file['value_max']+"\n"
27      file.write(writebuff)
28      writebuff=self.dict_with_file['max_frequency']+"\n"
29      file.write(writebuff)
30      writebuff=self.dict_with_file['number_of_classes']+"\n"
31      file.write(writebuff)
32      writebuff=self.dict_with_file['scale']+"\n"
33      file.write(writebuff)
34      writebuff=self.dict_with_file['histogram_type']+"\n"
35      file.write(writebuff)
36      writebuff=self.dict_with_file['number_of_quantiles']+"\n"
37      file.write(writebuff)
38      writebuff=self.dict_with_file['decimal_places']+"\n"
39      file.write(writebuff)
40      writebuff=self.dict_with_file['title']+"\n"
41      file.write(writebuff)
42      writebuff=self.dict_with_file['pos_stat']+"\n"
43      file.write(writebuff)
44      writebuff=self.dict_with_file['reference_value']+"\n"
45      file.write(writebuff)
46      file.close()
```

Caso o modo de operação seja `with_property` será necessária a criação de dois arquivos para a execução do `histplt`: o arquivo `.par` e o arquivo de dados com os pesos (caso houverem). O arquivo de dados com os pesos devem ser gerados a partir de uma propriedade selecionada no AR2GeMS.

5.5 Terceira parte: execução do programa

Depois de serem criados os arquivos necessários é hora da execução do `histplt`. Esse procedimento é invocado a partir do método `execute_gslib_hist` apresentado a seguir:

```
1  def execute_gslib_hist(self):
2    cmd=self.dict_gen_params['gslib_executables_folder']+self.↩
         ↪ dict_gen_params['path_separator']+self.dict_gen_params['↩
         ↪ gslib_hist_executable']+" "+self.dict_gen_params['↩
         ↪ work_folder']+self.dict_gen_params['path_separator']+self.↩
         ↪ dict_gen_params['par_file']
3    os.system(cmd+" > "+self.dict_gen_params['work_folder']+self.↩
         ↪ dict_gen_params['path_separator']+"out.log 2>&1")
4    ### print gslib message
5    gslib_msg = os.popen('cat '+self.dict_gen_params['work_folder']+↩
         ↪ self.dict_gen_params['path_separator']+"out.log").read()
6    print gslib_msg
7    ### check if gslib hist finishes correctly
8    if ('HISTPLT Version' not in gslib_msg and 'Finished' not in ↩
         ↪ gslib_msg):
9      print "Gslib Histogram did not finished."
10     self.dict_gen_params['execution_status']='ERROR'
```

O método `execute_gslib_hist` utiliza uma série de chaves já preparadas dentro do dicionário `dict_gen_params` (gen_params significa general parameters), como:

- `self.dict_gen_params['gslib_executables_folder']`
  Esta chave guarda a pasta onde estão os executáveis do GSLIB, mais especificamente o `histplt`.

- `self.dict_gen_params['path_separator']`
  Esta chave guarda o separador entre subpastas do sistema. Como estamos utilizando Linux primeiramente, o separador atribuído no início do script é o "/". Foi utilizada uma chave para o separador para facilitar uma futura implementação em Windows, que utiliza o separador "\".

- `self.dict_gen_params['gslib_hist_executable']`
  Esta chave guarda o nome do executável, no caso é `histplt`.

- `self.dict_gen_params['work_folder']`
  Esta chave guarda a pasta de trabalho, onde serão gravados os arquivos temporários: o arquivo `.par` e se necessário o arquivo de dados.

- `self.dict_gen_params['par_file']`
  Nome do arquivo temporário `.par`.

O método `execute_gslib_hist` captura ainda a string de saída do programa `histplt`. Com isso podemos verificar se o programa finalizou corretamente ou com algum erro. A string de saída do `histplt` é capturada na linha 5, impressa na tela na linha 6 e a verificação do sucesso do comando é realizada na linha 8.

5.6 Quarta parte: saída dos dados

O método `visualize_result` possibilita a visualização do arquivo de saída do programa `histplt` (o histograma no formato ps). Para ser visualizado o histograma deve-se inserir no código do plugin o nome de um programa instalado no sistema que possa ler e exibir arquivos postscript. Um exemplo para o caso do Linux é o uso do `evince`.

```
1 def visualize_result(self):
2    os.system(self.dict_gen_params['display_command']+' '+self.↩
       ↪ dict_gen_params['outputfile_folder']+self.dict_gen_params['↩
       ↪ path_separator']+self.dict_gen_params['outputfile_name'])
```

5.7 A sequência de chamada dos métodos

A sequência de chamadas dos métodos é mostrada na listagem 5.1 a descrição das operações é mostrada a seguir:

1. Linha 2: Inicializa uma instância da classe histplugin;
2. Linha 3: Coloca os parâmetros inseridos pelo usuário no dicionário correspondente dentro do objeto `instance`;
3. Linhas 4 e 5: Teste condicional que verifica se a execução ocorre sem problemas. Se a execução está correta o script executa o método `write_par_file` para criar os arquivos temporários necessários para a execução do `histplt`;
4. Linhas 6 e 7: Teste condicional que verifica se a execução ocorre sem problemas. Se a execução está correta o script executa o método `execute_gslib_hist` que realiza a execução do `histplt` no sistema;

5. Linhas 8 e 9: Teste condicional que verifica se a execução ocorre sem problemas. Se a execução está correta o script executa o método `visualize_result` que executa uma visualização do histograma com os resultados.

Listing 5.1: Código fonte do método execute.

```
1    def execute(self):
2      instance=histplugin(self.params)
3      instance.get_gui_input()
4      if instance.dict_gen_params['execution_status'] != 'ERROR':
5          instance.write_par_file()
6      if instance.dict_gen_params['execution_status'] != 'ERROR':
7          instance.execute_gslib_hist()
8      if instance.dict_gen_params['execution_status'] != 'ERROR':
9          instance.visualize_result()
10       else:
11         print "ERROR in execution."
```

Capítulo 6

Boas práticas para a implementação de um plugin

6.1 Opções exclusivas demandam grupos de radio button exclusivos

Muitas vezes o plugin espera um input do usuário na forma de uma escolha exclusiva, ou seja, o usuário deve escolher uma opção 1 ou uma opção 2. Neste caso um bom método é utilizar um radio button para cada opção. Depois de definir os widgets com as opções deve-se seguir os passos:

1. Selecionar todos os widgets radio buttons que formam o conjunto de escolhas. Isso pode ser feito clicando no primeiro widget e, segurando o botão CTRL selecionando os outros widgets radio button;

2. Aperte com o botão direito em cima de um dos radio buttons. Vá na opção "Assign to button group" e clique em "New button group" (ver a figura 6.1). A seleção exclusiva está pronta!

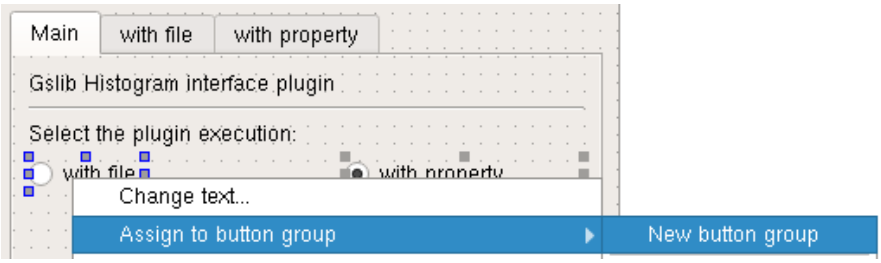

Figura 6.1: Criando um grupo de botões tipo radio para seleção exclusiva. Menu de criação de grupo de botões quando é apertado o botão direito do mouse em um dos elementos.

6.2 Tratamento dos parâmetros de entrada

O tratamento dos parâmetros de entrada é uma fase necessária para qualquer tipo de plugin, não só para plugins de interfaceamento de programas de terceiros. Por exemplo, o campo da coluna no arquivo de dados (figura 5.2, página 17)

Uso do método `.isdigit()` para aceitar números inteiros maiores ou iguais a zero:

```
>>> a='1024'
>>> a.isdigit()
True
>>> a='-1024'
>>> a.isdigit()
False
>>> a='0'
>>> a.isdigit()
True
>>> a='3.1415'
>>> a.isdigit()
False
```

A partir do método `isdigit()` podemos criar uma função que verifica se uma string representa um inteiro negativo, positivo ou neutro:

```
>>> def is_int(string):
...   if string[0]=='-':
...     return string[1:].isdigit()
...   else:
...     return string.isdigit()
```

```
...
>>> is_int('-1')
True
```

Se quisermos verificar que a entrada é um número do tipo ponto-flutuante podemos usar uma rotina com `try/except`:

```
>>> def is_floating_point(string):
...   if '.' not in string:
...     return False
...   try:
...     float(string)
...     return True
...   except:
...     return False
...
>>> is_floating_point('-3.1415')
True
>>> is_floating_point('-31415')
False
```

É importante enfatizar que no exemplo anterior os números inteiros recebem retorno `False`.

Outro método é o `isalnum()`, que verifica se a string contêm pelo menos um caractere e se é formado por letras e números. Exemplos do uso do método são mostrados na listagem 6.1.

Listing 6.1: Exemplos do uso do método isalnum().

```
>>> a='asdjdlk213'
>>> a.isalnum()
True
>>> a='asdjdlk21&3'
>>> a.isalnum()
False
>>> a='asdjdlk21/3'
>>> a.isalnum()
False
>>> a='abcdefg'
>>> a.isalnum()
True
>>> a='abcdefg1234'
>>> a.isalnum()
True
>>> a='99abcdefg1234'
>>> a.isalnum()
```

```
18 True
19 >>> a='991234'
20 >>> a.isalnum()
21 True
22 >>> a=''
23 >>> a.isalnum()
24 False
25 >>> a='asdf$'
26 >>> a.isalnum()
27 False
28 >>> a='/home/usuario'
29 >>> a.isalnum()
30 False
```

Análise do tratamento do campo do nome do arquivo postscript de saída, caracteres de espaço por exemplo.

6.3 Tratamento de valores de entrada usando widgets

No tratamentos dos valores de entrada utilizando uma interface gráfica podemos utilizar também os widgets que compôem a interface como um elemento que trata os dados de entrada

Alguns exemplos são:

1. QSpinBox: Este widget permite que um usuário utilize somente um número inteiro como entrada (figura 6.2). O programador pode determinar ainda o valor padrão, o valor mínimo e o valor máximo que poderá ser utilizado;

2. QDoubleSpinBox: Este widget permite que um usuário utilize somente um número ponto flutuante como entrada (figura 6.3). O programador pode determinar ainda o número de casas decimais ,o valor padrão, o valor mínimo e o valor máximo que poderá ser utilizado;

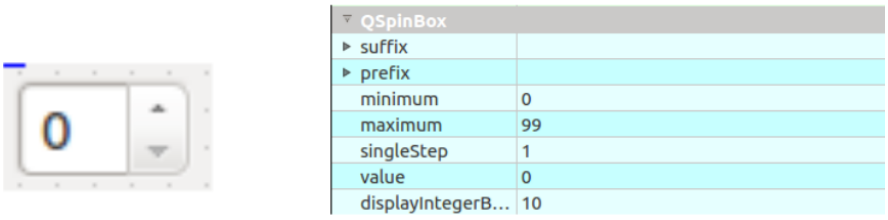

Figura 6.2: Imagem do widget QSpinBox juntamente com um quadro de parâmetros do widget.

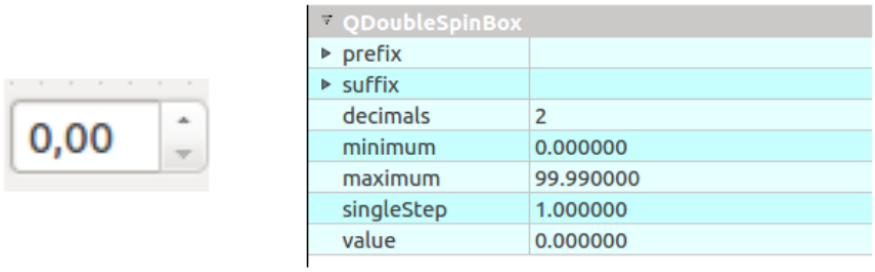

Figura 6.3: Imagem do widget QDoubleSpinBox juntamente com um quadro de parâmetros do widget. O valor padrão do widget poder ser definido alterando o campo "value".

6.4 Campos com entradas padrão

Outra técnica para facilitar o uso de um plugin é utilizar valores padrão em alguns campos de entrada. Por exemplo, o campo "Stat. Information position" pode ter um valor pré-definido, como $0,75$ por exemplo. Um valor padrão pode ser usado como uma dica caso o usuário não entenda perfeitamente o que o campo significa ou não saiba um valor razoável para o campo.

Todos os widgets de entrada permitem que o programador defina um valor padrão. Os exemplos dos widgets QSpinBox e QDoubleSpinBox foram mostrados nas figuras 6.2 e 6.3 (página 32).

Capítulo 7

Portando o plugin para Windows

O código do plugin desenvolvido pode ser utilizado no sistema operacional Windows obervando os seguinte items:

- Instalação dos programas GSLIB. Deve ser realizada a instalação dos programas do GSLIB, mais especificamente o `histplt`. A pasta de localização dos executáveis deve ser informada no corpo do programa, na chave `self.dict_gen_params['gslib_executables_folder']` e o nome `histplt` informado na chave

  `self.dict_gen_params['gslib_hist_executable']`;

- O separador de pastas deve ser alterado para o caractere "\". Chave `self.dict_gen_params['path_separator']`;

- Visualização do arquivo de saída. Para funcionar deve-se utilizar algum comando que exiba na tela o arquivo postscript gerado pelo `histplt`. É possível que o programa GSview realize a tarefa.

Se inicializados corretamente os parâmetros do plugin ele deve rodar no ambiente Windows. Porém isso ainda não foi testado.