Extending ICM+ with Python scripts with examples using CENTER-TBI data sets

Michał M. Placek
mp963@cam.ac.uk
8th September 2019
Python for ICM+ installation

This installer will start Python 3.7.4 installation.

Please make sure you tick 'Add Python to PATH'.

Once python is installed a little batch script will run to download 3 indispensable maths packages: numpy, scipy and stattools.

Please note that if you change the default installation location of Python you will need to run the script manually after the setup is done.
Please make sure you tick 'Add Python to PATH'! If you change the default installation location of Python, you will need to run the script manually after the setup is done.
Python for ICM+ installation

Please make sure you tick 'Add Python to PATH'.

Then, click 'Install Now.'

Tick these options!
Python for ICM+ installation: cont’d
Python for ICM+ installation: cont’d
Python for ICM+ installation: cont’d

Microsoft Visual C++ 2015 Redistributable (x86) - 14.0.23026

MICROSOFT SOFTWARE LICENSE TERMS
MICROSOFT VISUAL STUDIO 2015 ADD-ONS, VISUAL STUDIO SHELLS and C++ REDISTRIBUTABLE

- Agree to the license terms and conditions
- Install
- Close

Setup - ICM+ Python Installer

Information
Please read the following important information before continuing.

- Thank you for installing Python environment.
- You should now be able to use python scripts to extend ICM+ stats functions in the calculation engine.
- Please refer to the ICM+ Python plugin SOP for details.

Press any key to continue . . . .
Python Plugin important directories

The ICM+ plugins folder

The Python plugin
Python Plugin important directories

The ICM+ plugins folder

The ICM+ Python plugin folder

The Python script

The corresponding config file
The ICM+ Tool for a Python template and its config file creation
The ICM+ Tool for a Python template and its config file creation
The ICM+ Tool for a Python template and its config file creation
The ICM+ Tool for a Python template and its config file creation
Generated Configuration File

The XML config file generated by ICM+

```xml
<xml version = "1.0">
<PyToICMPlusConfig>
  <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
    <GUID>{7A097741-6044-4828-8371-B4BC3E6A1BFE}</GUID>
    <Description>Calculates Shannon entropy for a given data sequence.</Description>
  </Function>
</PyToICMPlusConfig>
```
The generated Python script

```python
# import ...
import scipy as sp

class ShannonEntropy:
    # DO NOT MODIFY THIS METHOD. It is a part of the ICM++-Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # config file because ICM++ will do it for you.
    # You will have to add your own code, only if you need to initialize some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size correponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 = input variable/signal 1
    # ts_time in the data_time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    # self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my own code here
        result = 0.0
        return result
```
import scipy as sp

class ShannonEntropy:
    # DO NOT MODIFY THIS METHOD. It is a part of the ICM++-Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM++ will do it for you.
    # You will have to add your own code, only if you need to initialise some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

    # You can append your own code to the destructor but most likely
    # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the params:
    # sig1 - input signal
    # ts_time - time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    # self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my own code here
        result = 0.0
        return result
The generated Python script

```python
# The generated Python script

class ShannonEntropy:
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    def __init__(self):
        self.sampling_freq = None

    def calculate(self, sig1, ts_time, ts_date):
        result = -0.0
        return result
```

One input signal

Add your own code to the `calculate` method
The generated Python script

```python
# import ...
import scipy as sp

class ShannonEntropy:
    # DO NOT MODIFY THIS METHOD. It is a part of the ICM++-Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # You can append your own code to the constructor, if needed.
    # You should not set here values of parameters declared in your XML
    # configuration file because ICM++ will do it for you.
    # You will have to add your own code, only if you need to initialize some
    # extra data structures which were not declared in the XML config file.
    def __init__(self):
        self.sampling_freq = None

        # You can append your own code to the destructor but most likely
        # you will not need it.
    def __del__(self):
        pass

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It takes the following inputs:
    # sig1 - input signal
    # ts_time - Time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    # self.sampling_freq
    def calculate(self, sig1, ts_time, ts_date):
        # my own code here
        result = -0.0
        return result
```

from scipy import stats

def calculate(self, sig1, ts_time, ts_date):
    result = stats.entropy(sig1)
    return result

One input signal

Add your own code to the `calculate` method
Adding options to the user-defined Python function
Adding options to the user-defined Python function
Adding options to the user-defined Python function
Adding an option of the type 'flag'

Data Field Definition Form

- **Name**: zeroOffset
- **Caption**: zero offset
- **Description**: Subtract current minimal value to zero offset
- **Type**: Flag (Y/N)
- **Is Mandatory**: 
- **Categories**:
- **Min - Max**:
  - Min: 0
  - Max: 0
- **Default**: False

 ![Image of Data Field Definition Form]
Adding an option of the type 'flag'

Data Field Definition Form

Name: zeroOffset
Caption: Zero offset
Description: Subtract current minimal value to zero offset
Type: Flag (Y/N)

Categories:

Min - Max: 0 0
Default: False

OK Cancel Keyboard
Adding an option of the type 'flag'

Data Field Definition Form
- Name: zeroOffset
- Description: Subtract current minimal value to zero offset
- Type: Flag (Y/N)
- Default: False

Python Plugin Script Configuration Dialog
- Function Name: ShannonEntropy
- Arguments: 1
- Description: Calculates Shannon entropy for a given data sequence.
- Import modules: NumPy, SciPy, StatTools
- Option Definitions:
  - Name: zeroOffset
  - Caption: Zero offset
  - Type: Flag
  - Description: Subtract current minimal value
Adding an option of the type 'category'
Adding an option of the type 'category'
Adding an option of the type 'category'
<?xml version = "1.0"?>

<PyToICMPlusConfig>
  <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">
    <GUID>{3DE49F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>
    <Description>Calculates Shannon entropy for a given data sequence.</Description>
    <Parameter ShortName="logBase" IsMandatory="False">
      <Caption>The base of the logarithm</Caption>
      <Description>The base of the logarithm used to calculate the entropy</Description>
      <Type Name="StringList">
        <Item Value="BIN" Caption="binary" IsDefault="True"/>
        <Item Value="NAT" Caption="natural"/>
        <Item Value="DEC" Caption="decimal"/>
      </Type>
    </Parameter>
    <Parameter ShortName="zeroOffset" IsMandatory="False">
      <Caption>Zero offset</Caption>
      <Description>Subtract current minimal value to zero offset</Description>
      <Type Name="Bool" DefaultValue="False"/>
    </Parameter>
  </Function>
</PyToICMPlusConfig>
<xml version="1.0"?>
<PyToICMPlusConfig>
  <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">  
    <GUID>{3DE497F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>  
    <Description>Calculates Shannon entropy for a given data sequence.</Description>  
    <Parameter ShortName="logBase" IsMandatory="False">  
      <Caption>The base of the logarithm</Caption>  
      <Description>The base of the logarithm used to calculate the entropy</Description>  
      <Type Name="StringList">  
        <Item Value="BIN" Caption="binary" IsDefault="True"/>  
        <Item Value="NAT" Caption="natural"/>  
        <Item Value="DEC" Caption="decimal"/>  
      </Type>  
    </Parameter>  
    <Parameter ShortName="zeroOffset" IsMandatory="False">  
      <Caption>Zero offset</Caption>  
      <Description>Subtract current minimal value to zero</Description>  
      <Type Name="Bool" DefaultValue="False"/>  
    </Parameter>  
  </Function>
</PyToICMPlusConfig>
<xml version = "1.0"?>
<PyToICMPlusConfig>
  <Function Name="ShannonEntropy" Type="Stats" SignalsCount="1">  
  <GUID>{3DE497F8-5AF6-40D5-907E-02B2CCDF19C5}</GUID>
  <Description>Calculates Shannon entropy for a given data sequence.</Description>
  <Parameter ShortName="logBase" IsMandatory="False">
    <Caption>The base of the logarithm</Caption>
    <Description>The base of the logarithm used to calculate the entropy</Description>
    <Type Name="StringList">
      <Item Value="BIN" Caption="binary" IsDefault="True"/>
      <Item Value="NAT" Caption="natural"/>
      <Item Value="DEC" Caption="decimal"/>
    </Type>
  </Parameter>
  <Parameter ShortName="zeroOffset" IsMandatory="False">
    <Caption>Zero offset</Caption>
    <Description>Subtract current minimal value to zero</Description>
    <Type Name="Bool" DefaultValue="False"/>
  </Parameter>
</Function>
</PyToICMPlusConfig>
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation Window
# It must return one floating-point number
# It take the following parameters:
# sigl - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
#   self.sampling_freq
# and the following variables already set at the initialisation time (via function options):
#   self.logBase - The base of the logarithm
#   self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sigl, ts_time, ts_date):
    if self.zeroOffset == True:
        sigl = np.array(sigl) - min(sigl)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sigl, None, base)
    return result
How to use the configured options in the Python script

```python
# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation Window
# It must return one floating-point number
# It take the following parameters:
# sigl - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
# self.sampling_freq
# and the following variables already set at the initialisation time (via function options):
# self.logBase - The base of the logarithm
# self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sigl, ts_time, ts_date):
    if self.zeroOffset == True:
        sigl = np.array(sigl) - min(sigl)

    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10

    result = stats.entropy(sigl, None, base)
    return result
```
How to use the configured options in the Python script

`'calculate' is the main work-horse function.
It is called with a data buffer (one or more) of size corresponding to the Calculation
It must return one floating-point number
It take the following parameters:
  sigl - input variable/signal 1
  ts_time - part of the data time stamp - number of milliseconds since midnight
  ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
It can also use the data sampling frequency:
  self.sampling_freq
and the following variables already set at the initialisation time (via function options)
  self.logBase - The base of the logarthim
  self.zeroOffset - Subtract current minimal value to zero offset

def calculate(self, sigl, ts_time, ts_date):
    if self.zeroOffset == True:
        sigl = np.array(sigl) - min(sigl)
    if self.logBase == 'BIN':
        base = 2
    elif self.logBase == 'NAT':
        base = math.e
    elif self.logBase == 'DEC':
        base = 10
    result = stats.entropy(sigl, None, base)
    return result
How to use the configured options in the Python script

# 'calculate' is the main work-horse function.
# It is called with a data buffer (one or more) of size corresponding to the Calculation
# It must return one floating-point number
# It take the following parameters:
# sig1 - input variable/signal 1
# ts_time - part of the data time stamp - number of milliseconds since midnight
# ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
# It can also use the data sampling frequency:
# self.sampling_frep
# and the following variables already set at the initialisation time (via function optio
# self.logBase - The base of the logarthim
# self.zeroOffset - Substract current minimal value to zero offset

def calculate(self, sig1, ts_time, ts_date):
  if self.zeroOffset == True:
    sig1 = np.array(sig1) - min(sig1)
  if self.logBase == 'BIN':
    base = 2
  elif self.logBase == 'NAT':
    base = math.e
  elif self.logBase == 'DEC':
    base = 10
  result = stats.entropy(sig1, None, base)
  return result
Using user-defined Python function in ICM+
Using user-defined Python function in ICM+
Using user-defined Python function in ICM+
Using user-defined Python function in ICM+
Using user-defined Python function in ICM+
Using user-defined Python function in ICM+

1. Open the On Line Analysis Configuration Dialog.
2. Add the necessary virtual signals and define them in the Virtual Signals tab.
3. Select the primary analysis and add the defined Python function in the Primary Analysis tab.
4. Configure the calculation window specification and enabled status in the Primary Analysis Configuration Editor.
5. Insert the function and specify its arguments in the Formula section.
6. Check the Options for the selected function.
7. Set the valid values range and input parameters accordingly.

Function options:
- Function: PyShannonEntropy
- Base of the logarithm: natural
- Zero offset: checked

Brain Physics Lab
Using user-defined Python function in ICM+

- Function: PyShannonEntropy
- The base of the logarithm: natural
- Zero offset

Function definition:
- Function Name: ShannonEntropy
- Arguments: 1

Description:
Calculates Shannon entropy for a given data sequence.

Import modules: NumPy, SciPy, StatTools

Option Definitions:
- logData: The base of the logarithm
- Category: The base of the logarithm
- zeroOffset: Zero offset
- Flag: Substract current minimal value
Using user-defined Python function in ICM+

- **Function options**: PyShannonEntropy
  - The base of the logarithm: natural
  - Zero offset: checked

- **Primary Analysis Configuration Editor**
  - **Name**: ABP
  - Calculation Window Specification:
    - Calculation Period: 300 s
    - Update Period: 60 s
  - Valid values range:
    - Max Value: 0
    - Min Value: 0

**Formula**:

Options:
- PyPartialCorrel
- PyShannonEntropy
- PySpearmanCorrel
- Range
- RankCorrel
- ReinCount

**Inputs**:
- ABP
- etCO2
- FV
Using user-defined Python function in ICM+

Formulas:
- **PyShannonEntropy**: Calculates Shannon entropy for a given data sequence.
Using user-defined Python function in ICM+

- **On Line Analysis Configuration Dialog**
  - Virtual Signals: ABP, FV, etCO2
  - Primary Analysis: Formulas: Mean(ABP), Mean(FV), Mean(etCO2)
  - Final Analysis: Formula: `PyShannonEntropy(ABP, logBase=NAT&zeroOffset)`

- **Primary Analysis Configuration Editor**
  - Name: Entr
  - Calculation Window Specification: Calculation Period: 300 s
  - Valid values range: Max Value: 0, Min Value: 0

- **Function Options**
  - Function: PyShannonEntropy
  - Function description: Calculates Shannon entropy for a given data sequence.
Using user-defined Python function in ICM+
Partial correlation – example of a function with three inputs
Partial correlation – example of a function with three inputs
import numpy as np

class PartialCorrel:
    # DO NOT MODIFY THIS METHOD. It is a part of the ICM++-Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # ...
    # ...

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # sig2 - input variable/signal 2
    # sig3 - input variable/signal 3
    # ts_time - part of the data time stamp - number of milliseconds since midnight
    # ts_date - Part of the data time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    #   self.sampling_freq
    def calculate(self, sig1, sig2, sig3, ts_time, ts_date):
        # my own code here
        result = 0.0
        return result
import numpy as np

class PartialCorrel:
    # DO NOT MODIFY THIS METHOD. It is a part of the ICM++-Python interface.
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    # ...
    # ...

    # 'calculate' is the main work-horse function.
    # It is called with a data buffer (one or more) of size corresponding to the Calculation Window
    # It must return one floating-point number
    # It take the following parameters:
    # sig1 - input variable/signal 1
    # sig2 - input variable/signal 2
    # sig3 - input variable/signal 3
    # ts_time - part of the date-time stamp - number of milliseconds since midnight
    # ts_date - Part of the date-time stamp - One plus number of days since 1/1/0001
    # It can also use the data sampling frequency:
    # self.sampling_freq
    def calculate(self, sig1, sig2, sig3, ts_time, ts_date):
        # my_own_code_here
        result = 0.0
        return result
import numpy as np
from par_corr_module import partial_corr

class PartialCorrel:
    
    def set_parameter(self, param_name, param_value):
        setattr(self, param_name, param_value)

    
    def calculate(self, sig1, sig2, sig3, ts_time, ts_date):
        A = np.array([sig1, sig2, sig3]).transpose()
        R_coefficients = partial_corr(A)
        return R_coefficients[0, 1]
Using user-defined Python function in ICM+

**Primary Analysis Configuration Editor**

- **Name:**
  - parCorr
  - PyPartialCorrel (ABP, FV, etCO2)

- **Calculation Window Specification**
  - Calculation Period: 300 s
  - Update Period: 60 s

- **Valid values range**
  - Max Value: 0
  - Min Value: 0

**Formula:**

```
PyPartialCorrel(ABP, FV, etCO2)
```

**Function description:**
Calculates linear partial correlation between two input data sequences, controlling for the third input data sequence.
Using user-defined Python function in ICM+

Three input signals

Formula: 

PyPartialCorrel(ABP, FV, etCO2)

Description:
Calculates linear partial correlation between two input data sequences, controlling for the third input data sequence.
Using user-defined Python function in ICM+
Happy pythoning!

Based on image by Jeffrey Elkner
Happy pythoning!

I invite you to see my poster (#321, Monday, 12:00–13:00)