引数が可変長の関数の設計

昨日に引き続き、関数を別ファイルで保存する方法を試行する。

昨日作成した関数は、引数が一つだけであったが Matlab の関数の中には多数の引数を持つ関数も存在する。また、引数の名前と値をセットで渡すことにより振る舞いを変化させることができる関数も存在する。

python では関数の引数にデフォルト値を与えて置くことで、全ての引数が与えられない場合でも処理を続けることができるのは、先日述べたが、Matlab でもこの方法を実装使用とすると、昨日の方法では不可能だったため、今日はそれについて新しく調べる。

今回は以下のサイトを参考にした。

- varargin
- inputParser

デフォルト引数の問題点

昨日実装したデフォルト引数は、以下のようなものだった。

```
function result = imrecognition(app, camera, SnapShot, Binalized, ParamPreview)

    arguments
        app
        camera
        SnapShot {mustBeInteger(SnapShot)} = true
        % スナップショットした画像を使用するか
        % True: スナップショットを撮影
        % False: 画像をファイルから読み込む。
        Binalized {mustBeInteger(Binalized)} = false
        % 読み込んだ画像に対して二値化を行うか
        % True: 二値化を行う。
        % False: 二値化を行わない。
        ParamPreview {mustBeInteger(ParamPreview)} = false
        % 画像サイズなどのパラメータをダイアログの適当な場所に表示するか。
        % True: 表示する
        % False: 表示しない。
    end
```

Matlab では、デフォルト引数を渡すことで引数を渡す位置は自由に変更可能になるが、その個数自体はできないという問題が存在する。すなわち、以下のようなことは許されない。

```
result = imrecognition(app, camera)
```

可変長引数

このような引数の長さ自体が変化する場合、Matlab では `varargin` という cell 配列を用いて引数をまとめて渡す。すなわち、先ほどの関数を例に取ると

```
function result = imrecognition(varargin)
```

可変長引数にデフォルト値を設定する

可変長引数にデフォルト値を設定する方法は、Matlab アドオンで提供される `.m` ファイルを参考にした。

その方法は、次の 3 つの関数を追加することである。

```
function inputArguments = iParseInputArguments(varargin)
    parser = iCreateParser;
    parser.parse(varargin{:});
    inputArguments = iConvertToCanonicalForm(parser.Results);
end

function p = iCreateParser()
    p = inputParser;
    defaultOption = 'global';
    defaultUpperThreshold = 0.8;
    defaultLowerThreshold = 0.5;

    addParameter(p, 'Option', defaultOption);
    addParameter(p, 'UpperThresh', defaultUpperThreshold);
    addParameter(p, 'LowerThresh', defaultLowerThreshold);
end

function inputArguments = iConvertToCanonicalForm( params )
    inputArguments = struct;

    inputArguments.Option = params.Option;
    inputArguments.UpperThresh = params.UpperThresh;
    inputArguments.LowerThresh = params.LowerThresh;
end
```

デフォルト引数に設定

今回実装する関数は、画像データを入力として確実に受け取りたい。すなわち、画像データが渡されない場合は、関数の処理を実行しないようにしたい。そのためには、画像データを 位置引数として `arguments` に設定し、`varargin` を繰り返し変数として `arguments` に設定すればよい。

よって、関数ファイルは最終的に次のようになった。

```
function [IMAGE_bw, IMAGE__] = ImBinarize( src, varargin )
% 画像の二値化を行う関数
%   Parames
%   -------
%   src (array):
%       二値化される画像の画素値配列
%   options (char):
%       二値化の方法
%           * 'global': 大域的二値化(大津の二値化)
%           * 'adaptive': 適応的二値化
%           * 'tow_thresh': 二つの閾値で二値化
%   threshold (float):
%       閾値( 0 <= 閾値 <= 1)

    % 位置引数を設定
    arguments
       src {mustBeNonempty, mustBeNumeric, mustBePositive}
    end
    % 繰り返し引数を設定
    arguments (Repeating)
        varargin
    end

    % Parse the input arguments.
    inputArguments = iParseInputArguments(varargin{:});


    if inputArguments.Option == "global"
        IMAGE_bw= imbinarize(src,'global');%Globalな閾値で二値化する
        IMAGE__ = imcomplement(IMAGE_bw);%その反転した値
    elseif inputArguments.Option == "adaptive"
        IMAGE_bw= imbinarize(src,'adaptive');%Adaptiveな閾値で二値化する
        IMAGE__ = imcomplement(IMAGE_bw);%その反転した値
    else
        %IMAGE_bw= imbinarize(src,app.UpperThresholdEditField.Value);%上側の閾値で二
値化する
        %IMAGE__ = imbinarize(src,app.LowerThresholdEditField.Value);%下側の閾値で二
値化する
        IMAGE_bw= imbinarize(src, inputArguments.UpperThresh);%上側の閾値で二値化する
        IMAGE__ = imbinarize(src, inputArguments.LowerThresh);%下側の閾値で二値化する
        IMAGE__ = imcomplement(IMAGE__);%下側の閾値で二値化した値を反転する。1⇒0　あるい
は　0⇒1　となる。
    end

end

function inputArguments = iParseInputArguments(varargin)
    parser = iCreateParser;
    parser.parse(varargin{:});
    inputArguments = iConvertToCanonicalForm(parser.Results);
end

function p = iCreateParser()
```

```
    p = inputParser;
    defaultOption = 'global';
    defaultUpperThreshold = 0.8;
    defaultLowerThreshold = 0.5;

    addParameter(p, 'Option', defaultOption);
    addParameter(p, 'UpperThresh', defaultUpperThreshold);
    addParameter(p, 'LowerThresh', defaultLowerThreshold);
end

function inputArguments = iConvertToCanonicalForm( params )
    inputArguments = struct;

    inputArguments.Option = params.Option;
    inputArguments.UpperThresh = params.UpperThresh;
    inputArguments.LowerThresh = params.LowerThresh;
end
```

この関数は次のように実行する。

```
>> a = ones(3, 3, 3, 'uint8')

>> [IMAGE_bw, IMAGE__] = ImBinarize(a)

  3×3×3 の logical 配列

IMAGE_bw(:,:,1) =

   1   1   1
   1   1   1
   1   1   1


IMAGE_bw(:,:,2) =

   1   1   1
   1   1   1
   1   1   1


IMAGE_bw(:,:,3) =

   1   1   1
   1   1   1
   1   1   1


  3×3×3 の logical 配列

IMAGE__(:,:,1) =

   0   0   0
```

```
    0    0    0
    0    0    0


IMAGE__(:,:,2) =

    0    0    0
    0    0    0
    0    0    0


IMAGE__(:,:,3) =

    0    0    0
    0    0    0
    0    0    0
```

また、option は次のように与えることができる。

```
>> [IMAGE_bw, IMAGE__] = ImBinarize(a, 'Option', "adaptive")

  3×3×3 の logical 配列

IMAGE_bw(:,:,1) =

    0    0    0
    0    0    0
    0    0    0


IMAGE_bw(:,:,2) =

    0    0    0
    0    0    0
    0    0    0


IMAGE_bw(:,:,3) =

    0    0    0
    0    0    0
    0    0    0


  3×3×3 の logical 配列

IMAGE__(:,:,1) =

    1    1    1
    1    1    1
    1    1    1
```

```
IMAGE__(:,:,2) =

   1   1   1
   1   1   1
   1   1   1


IMAGE__(:,:,3) =

   1   1   1
   1   1   1
   1   1   1
```

出力の配列の要素である 0, 1 が反転していることから、`option` を引数として渡すことで関数の振る舞いが変化したことがわかる。