SRA の技術情報誌
GS LetterNeo
Vol.181　2023 年 8 月

PyGSP を使ったグラフ信号処理

熊澤 努

技術本部 先端技術研究室

はじめに

今回の記事では、Python を使ったグラフ信号処理に挑戦します。従来の信号処理では、例えば、時系列データでは時間の前後関係が、画像データではピクセル同士の配置関係がといったように、信号同士の関係性が暗黙に仮定されていました。グラフ信号処理は信号同士の関係性を明示した処理を実現する技術です。この記事では，グラフ信号処理用の Python パッケージとして、PyGSP[1]を使います。

グラフ信号処理とは

グラフ信号処理は、構造を持った信号に対して行う信号処理技術です。文献 [1]では、構造を持った信号の例として、ソーシャルネットワークやセンサネットワーク上の高次元データが挙げられています。こうしたネットワークを構成する要素同士には地理的だけでなく、論理的な隣接関係などの関係性があります。グラフ信号処理はそのような関連性をグラフ構造として捉えて、フーリエ変換やフィルタリングなどのデータ処理を行うことを目指しています。グラフ信号処理の詳しい解説は、例えば、書籍 [2]を参照してください。

[1] https://pygsp.readthedocs.io/en/stable/

PyGSP を準備する

PyGSP は pip でインストールできます。

```
> pip install pygsp
```

PyGSP は NumPy をパッケージ内部で利用しています。また、可視化のために matplotlib を使うこともできるので、必要に応じてインストールしましょう。なお、この記事で紹介するプログラムは Python 3.6、Jupyter Notebook 上で動作を確認しています。

グラフを生成する

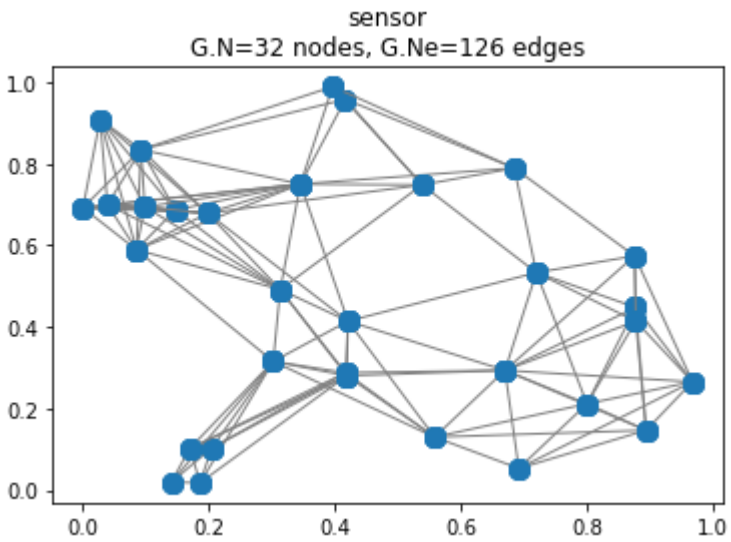
グラフは頂点とそれらを結ぶ辺によって構成される数学的構造です。グラフ信号処理は信号がグラフの頂点に乗っていることを仮定しています。まず、グラフを作り、信号の構造を決めましょう。今回の記事では、センサグラフという組込みのグラフを生成します。

```
import numpy as np
from pygsp import graphs

# Create Sensor Graph
G = graphs.Sensor(N=32、 seed=1)
G.compute_fourier_basis()

G.plot()
```

センサグラフは、`graphs.Sensor` オブジェクトを生成するだけで簡単に作ることができます。ここでは、センサグラフは `32` 個の頂点を持ち、それらの間を辺でランダムに接続しています。`compute_fourier_basis` メソッドは後で使うグラフフーリエ変換のための準備で、フーリエ変換の基底を計算します。最後に、`plot` メソッドを呼んで生成したグラフを描画しています。グラフの描画結果を以下に示します。頂点が青い丸で、辺が青丸を結ぶ実線で描かれています。



各頂点を 1 つのセンサと考えると、このグラフはセンサをネットワーク状に配置した構成図を表すとみなすことができます。

信号を生成する

次に各頂点に割り当てる信号を生成します。頂点をセンサと考えると、ここで生成する信号はセンサのデータに相当します。今回は 分かりやすさのため、numpy を使ってランダムに 2 種類の値を取る信号を生成します。

```
# Create signals on Sensor Graph
signal = np.copysign(np.ones(G.N)、 np.random.normal(size=(G.N、)))
low_freq_signal = (signal + 1) /2
high_freq_signal = signal * 2

print(low_freq_signal)
print(high_freq_signal)
```

変数 signal は 1 または-1 から成る配列です。その長さはグラフの頂点の数 32 個とします。グラフの頂点の数は、G.N で取得することができます。実際に解析を行う信号は signal 元に生成した low_freq_signal、 high_freq_signal です。

信号 low_freq_signal は以下のように 0 と 1 の値をランダムに取ります。配列の要素は頂点を表しており、配列の各要素が各頂点に割り当てる信号の値です。

```
[1. 0. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0.
 0. 1. 1. 0. 1. 1. 1. 1.]
```

もう一つの信号 high_freq_signal は、2 と-2 の値をランダムに取ります。

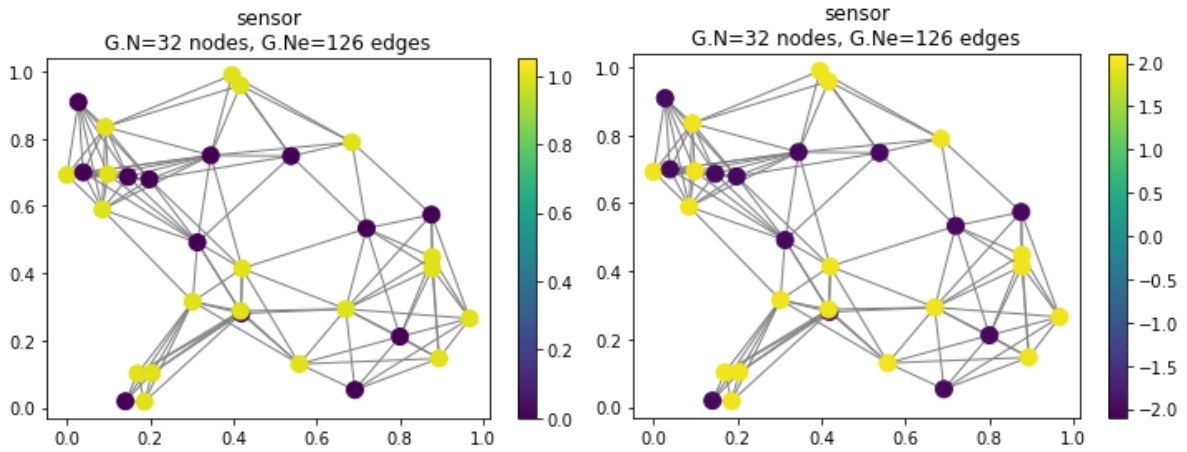
```
[ 2. -2.  2.  2. -2.  2.  2. -2.  2. -2. -2.  2.  2.  2. -2.  2.  2.  2.
 -2. -2. -2.  2. -2. -2. -2.  2.  2. -2.  2.  2.  2.  2.]
```

両者は値の取る範囲が異なります。その結果、low_freq_signal の方が high_freq_signal より値の変動が小さくなることが予想されます。後でこのことを確かめてみましょう。

準備の最後として、生成したグラフの各頂点に割り当てた結果を示します。各頂点の信号値を表示するには plot_signal メソッドを使います。

```
# Plot signals
G.plot_signal(low_freq_signal)
G.plot_signal(high_freq_signal)
```

表示した結果は下の図の通りです。左の図が low_freq_signal を割り当てた結果、右の図が high_freq_signal を割り当てた結果です。どちらも、大きい値が黄色で、小さい値が紫色で彩色されています。



信号の変動を解析する

今回行うグラフ信号処理は，隣り合う頂点に割り当てられた信号間の変動分析です。この信号の変動の分析は、従来のディジタル信号処理ではフーリエ変換を使った周波数解析で実現されています。例えば，画像信号にフーリエ変換を行うと、隣接ピクセル同士の色の変動を，色の変動の小さい低周波数域から，色の変動の大きい高周波数域までの情報として取り出すことができます。グラフ信号処理においてはグラフフーリエ変換がその役割を担います。グラフ信号の場合はスペクトルグラフ理論というグラフ理論で研究されてきた技術を利用しています。詳細は文献 [2]を参照してください。

グラフフーリエ変換は PyGSP で実装されており、以下のように信号を引数として `gft` メソッドを呼び出すだけで実行することができます。

```
# Graph Fourier Transform
print(G.gft(low_freq_signal))
print(G.gft(high_freq_signal))
```

信号 `low_freq_signal` に対するグラフフーリエ変換の実行結果を以下に示します。

```
[-3.35875721 -0.36662622  0.18788458  0.50282386 -0.30518139 -0.24069401
 -1.03533948  0.53930636  0.08058275  0.3262177  -0.16108244  0.75739795
 -0.56374348  0.07105476 -0.49971729 -0.46159463  0.14982828 -0.19320994
  0.3664829   0.06481266 -0.37229327  0.62127059 -0.95353396 -0.47163051
 -1.06183908  0.3094165  -0.03031026  0.53239583 -0.39142869  0.28768766
  0.74663902 -0.35862939]
```

グラフフーリエ変換の結果は配列で与えられます。配列には、隣り合う頂点間を比較した際の，信号値の変動の少ない成分から大きい成分が先頭から順に並んでいます。この信号の場合は、変動のない成分の絶対値が最も大きいことが分かります。これは、隣り合う頂点との信号の変動が緩やかであることを意味しています。実際、上のグラフの図を見ると。同じ値（色）の頂点が隣り合っていることが多いため、信号の変動があまり見られないことが分かります。

次に、信号 high_freq_signal に対するグラフフーリエ変換の実行結果を以下に示します。

```
[-2.12132034 -1.46650489  0.75153831  2.01129542 -1.22072556 -0.96277602
 -4.1413579   2.15722543  0.32233098  1.3048708  -0.64432975  3.02959181
 -2.25497392  0.28421903 -1.99886916 -1.84637851  0.59931312 -0.77283974
  1.46593159  0.25925065 -1.48917309  2.48508236 -3.81413586 -1.88652206
 -4.2473563   1.237666   -0.12124104  2.12958331 -1.56571474  1.15075065
  2.98655607 -1.43451755]
```

配列の各要素の絶対値が、先頭要素の絶対値と比べて大きくなっていることが分かります。このことは、変動の大きい成分も含んでいることを意味しています。信号 high_freq_signal は、信号 low_freq_signal と比べて，信号値の異なる頂点同士が隣り合っている箇所での値の変動が大きいことが結果に反映されていると考えられます。以上から、グラフフーリエ変換により、low_freq_signal の方が high_freq_signal より値の変動が小さいことを確かめることができました。

今回作成したプログラムの全体を示します。

```
import numpy as np
from pygsp import graphs

# Create Sensor Graph
G = graphs.Sensor(N=32、 seed=1)
G.compute_fourier_basis()

G.plot()

# Create signals on Sensor Graph
signal = np.copysign(np.ones(G.N)、 np.random.normal(size=(G.N、)))
low_freq_signal = (signal + 1) /2
high_freq_signal = signal * 2

print(low_freq_signal)
print(high_freq_signal)

# Plot signals
G.plot_signal(low_freq_signal)
G.plot_signal(high_freq_signal)

# Graph Fourier Transform
print(G.gft(low_freq_signal))
print(G.gft(high_freq_signal))
```

おわりに

今回は、Python 用グラフ信号処理パッケージ PyGSP を使い、グラフ信号処理を体験しました。グラフフーリエ変換を使い、グラフ上に割り当てた信号の変動を分析しました。PyGSP には、他にもフィルタリング、フィルタバンク、ウェーブレット変換などの各種処理があり、

様々な信号処理を実行することができます。グラフ構造は様々な分野で応用されているため、グラフ信号処理の活用の幅がますます広がることを期待したいと思います。

引用文献

[1] 田中雄一, "グラフ信号処理のすゝめ," *IEICE Fundamental Review,* vol. 8, no. 1, pp. 15-29, 2014.

[2] 田中聡久監修 , 田中雄一著, グラフ信号処理の基礎と応用 - ネットワーク上データのフーリエ変換，フィルタリング，学習 -, コロナ社, 2023.

GSLetterNeo Vol.181
2023 年 8 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芬
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8