SRA の技術情報誌
GSLetterNeo
Vol.184　2023 年 11 月

PyGSP を使ったグラフ信号処理（２）

熊澤 努

技術本部 先端技術研究室

はじめに

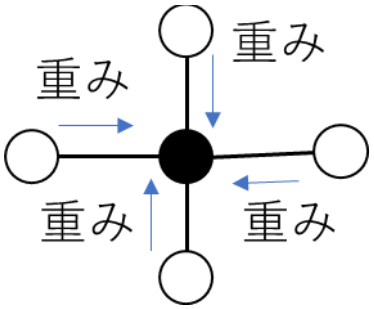
Vol.181 では、Python を使ったグラフ信号処理の一つであるグラフフーリエ変換に挑戦しました[1]。今回はその続きとして、グラフ上のフィルタリングに挑戦します。この記事でも、引き続きグラフ信号処理用の Python パッケージ PyGSP[2]を使います。

フィルタリング

グラフ信号のフィルタリングとは、グラフの各頂点の信号とその近くの頂点の信号との重み付き平均値を計算することです。重みづけに使う係数を並べたものをグラフフィルタと呼びます。頂点同士の近さは、辺を辿って一方の頂点から他方の頂点まで行くときの辿る辺の最小数で測ります。隣り合っている頂点同士の近さは１で、たくさん辺を辿る必要のある頂点ほど大きくなり、その頂点から遠いという意味を持ちます。フィルタはどのくらい近い頂点まで計算の対象にするか、という情報も含みます。下の図は、真ん中の黒い頂点の信号に対して、隣の頂点の信号使ってフィルタリングを行う様子を模式的に表しています。図を見ると分かるように隣の頂点の信号が伝わってくるので、フィルタリングは近くの頂点の情報

[1] https://www.sra.co.jp/Portals/0/files/gsletter/pdf/GSLetterNeoVol181.pdf
[2] https://pygsp.readthedocs.io/en/stable/

を伝搬させる処理とも理解することができます。フィルタリングについての詳しい解説は書籍 [1]を見ていただくとして、早速使ってみます。



組込みのフィルタを使って熱の拡散の様子を見る

PyGSP には組込みのフィルタが用意されています。公式サイトのチュートリアル[3]を元に、熱源から熱がグラフ上を伝わった状況を表示してみましょう。

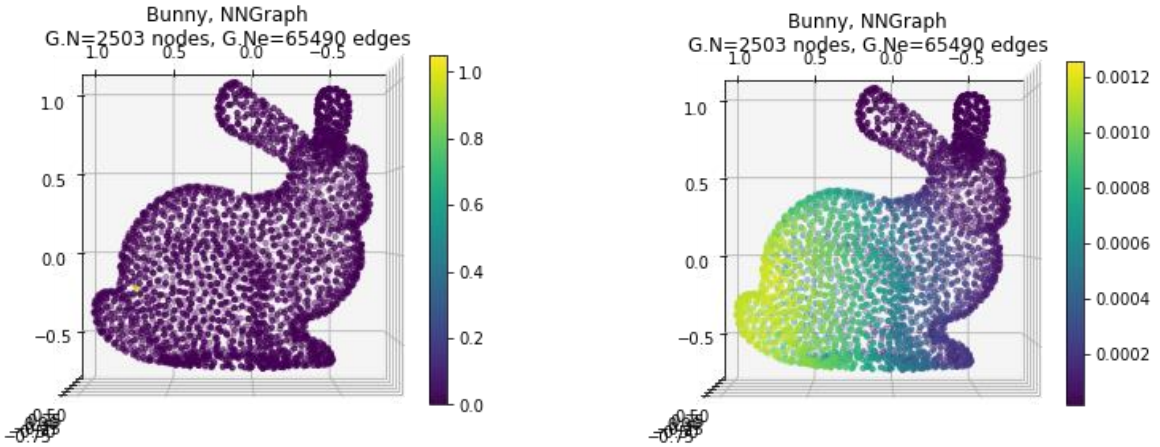
```
import numpy as np
from pygsp import graphs, filters

G = graphs.Bunny()

# 信号を生成
x = np.zeros(G.N)
x[1000] = 1
G.plot_signal(x, backend='matplotlib')

# フィルタリング
heat = filters.Heat(G, tau=100)
y = heat.filter(x)
G.plot_signal(y, backend='matplotlib')
```

今回使用するグラフは、Bunny という頂点を 2503 個持つウサギの形をした PyGSP の組込みグラフです。信号（変数 x）は 1 頂点（1000 番目の頂点）だけが 1 で、残りは全て 0 にしました。この 1000 番目の頂点が熱源を表しています。 熱が伝わる様子は Heat という組込みのフィルタを使うと分かります。パラメータ tau はチュートリアルの設定値をそのまま使っています。プログラムの下から 2 行目で、filter を使ってフィルタリングを実行します。

上のプログラムを実行した結果を下に示します。左がフィルタリングを行う前の原信号、右がフィルタリングを行った後の信号の様子です。熱源はウサギの尻尾のあたりの頂点ですが、頂点数が大きいため左の図では見えません。しかし、右の図から尻尾から胴体に向かって熱が伝わる様子が見て取れます。なお、Heat フィルタは低域通過フィルタというフィル

[3] https://pygsp.readthedocs.io/en/stable/tutorials/wavelet.html

タの一種で、信号を滑らかにする処理を行います。図からは分かりにくいですが、熱源の熱が周囲に伝わることで首位の温度に近づいています。色で表された信号値の変化がスムーズになっていることからも、このことが分かります。



独自に作ったフィルタでフィルタリングをする

今度は自分で設計したフィルタを使ってフィルタリングを実行します。書籍 [1]を参考に、簡単な低域フィルタを作ることに挑戦します。

```
import numpy as np
from pygsp import graphs, filters
from scipy import sparse

# グラフを生成
G = graphs.Bunny()
G.compute_fourier_basis()
G.compute_laplacian()
L = G.L.toarray() # グラフラプラシアン

# 信号を生成
r = np.random.RandomState(1)
x = 3*np.copysign(np.ones(G.N), G.U[:, 2]) - 2*np.copysign(np.ones(G.N),
G.U[:, 1]) + 0.3*r.standard_normal(G.N)
G.plot_signal(x, backend='matplotlib')

# フィルタを生成
I = sparse.identity(G.N, format='csc')
H = (I - L) ** 2

# 頂点領域でのフィルタリング
y = np.dot(x, H)
G.plot_signal(np.array(y.flatten().tolist()[0]), backend='matplotlib')
```

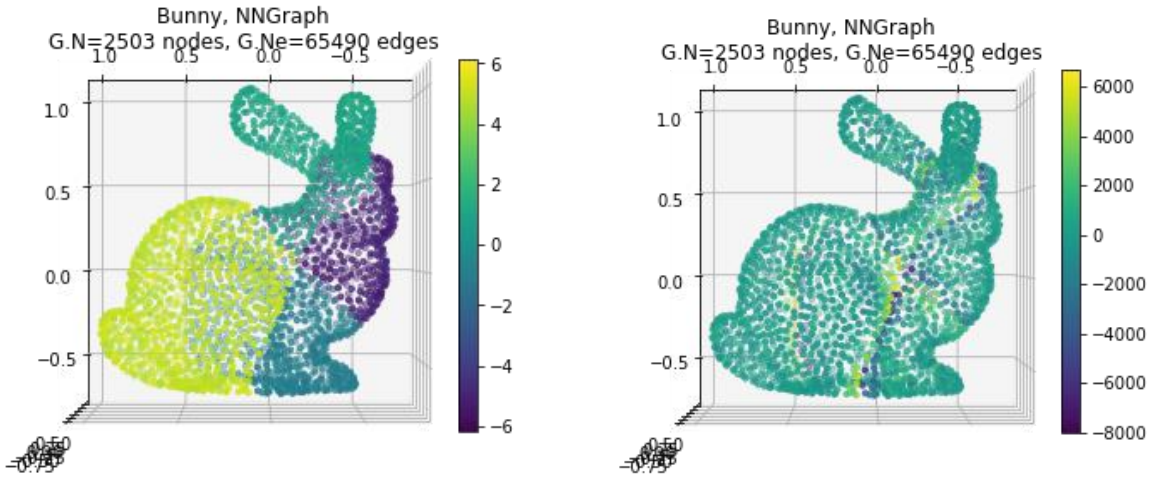
グラフは Bunny を使います。先ほど異なり、グラフの生成後にいくつか計算をしていますが、これはグラフラプラシアンという値を算出するための計算です。グラフラプラシアンは後でフィルタをつくるときに使います。グラフラプラシアンについては書籍 [1]を参照してください。

次に信号を生成します。はっきりと区別のできる値のグループができるように、信号を生成するため、少し複雑になっています。また、擬似乱数を利用して雑音も付加しています。

今使うフィルタ（変数 H で表しています）は、2 次の多項式グラフフィルタというフィルタの一種です。各頂点について、その頂点の隣の頂点の信号と、さらにその隣の頂点の信号を伝搬させて重み付き和を計算することで信号を合成します。このことは、H の計算に出てくる 2 乗に反映されています。

下から 2 行目でフィルタリングを実行しています。ここでは PyGSP で用意しているフィルタリングの仕組みを使わずに、フィルタ（行列、つまり 2 次元配列）と信号（ベクトル、つまり 1 次元配列）の積でフィルタリングを行っています。行列をベクトルでフィルタや信号を表現することで、掛け算だけでフィルタリングを実行することができます。

先ほど同じように、実行結果を下の図に示します。左がフィルタリングを行う前の原信号で、右がフィルタリングを行った後の信号です。



原信号でははっきりと色分けされてグループに分かれていた信号が、フィルタリングを行うことで、全体的にぼやけています。特にグループの境界部分でその傾向が顕著です。 フィルタリングによって 2 つ隣の頂点の信号まで含めて重み付き和を取ることは、それらの頂点の情報の平均を取る操作に相当します。つまり、全体的に同じような値に近づくことになるので、グループの境界が分かりにくくなります。右の図でこのことが反映されていることが分かります。これはグラフ信号を滑らかにする処理なので、使ったフィルタも低域通過フィルタです。

おわりに

今回も Vol.181 に続いて PyGSP を使い、グラフ信号のフィルタリングに挑戦しました．フィルタリングは雑音の除去など応用範囲も広いため、いろいろ試すと面白い結果が得られるのではないかと思います。

引用文献

[1] 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.184
2023 年 11 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

夢を。Yawaraka Innovation
やわらかいのべーしょん