bmm: Bayesian Map-matching

Samuel Duffield[1]

1 University of Cambridge

DOI: 10.21105/joss.03651

Software

- Review
- Repository
- Archive

Editor: Bita Hasheminezhad

Reviewers:

- @georgiastuart
- @matt-graham

Submitted: 26 July 2021
Published: 11 February 2022

Summary

bmm is a Python package providing probabilistic map-matching with uncertainty quantification. Map-matching is the task of converting a polyline (series of noisy location observations - e.g. GPS data) and a graph (collection of edges and nodes) into a continuous route trajectory restricted to the graph. Here a continuous route is represented by series of connected edges as well as positions along said edges at observation time. bmm uses Bayesian particle smoothing methods to produce a collection of particles, each of which representing a continuous, plausible route along edges in the graph.

bmm is built on top of osmnx (Boeing, 2017) - a python package assisting with the retrieval and processing of OpenStreetMap data (OpenStreetMap contributors, 2017). Although, bmm is applicable to be used on any suitably labelled NetworkX graph (Hagberg et al., 2008).

In addition, bmm utilises numpy (Harris et al., 2020) and numba (Lam et al., 2015) for fast scientific calculations, pandas (The pandas development team, 2020) and geopandas (Jordahl et al., 2020) for spatial data storage and manipulation as well as matplotlib (Hunter, 2007) for visualisation.

Documentation for bmm can be found at bmm.readthedocs.io.

Statement of need

Map-matching is a vital task for data driven inference involving GPS data. Map-matching is often non-trivial, i.e. when the graph is dense, the observation noise is significant and/or the time between observations is large. In these cases there may be multiple routes that could have feasibly generated the observed polyline and returning a single trajectory is suboptimal. Indeed, of 500 routes successfully map-matched using bmm from the Porto taxi dataset (Moreira-Matias et al., 2013), 467 exhibited multi-modality. This uncertainty over the inferred route would not be captured in the single trajectory approach that is adopted by the most prominent map-matching software Luxen & Vetter (2011) and Yang & Gidofalvi (2018), which adapt a Viterbi algorithm - first applied to map-matching in Newson & Krumm (2009). The code for Luxen & Vetter (2011) is found as part of the OSRM project and represents an efficient C++ implementation although is not easily accessible through Python. The software package accompanying Yang & Gidofalvi (2018) is found at fmm and provides extremely fast map-matching but without the convenience and accessibility of working directly with an osmnx graph.

bmm adopts a state-space model approach as described in Duffield & Singh (2022) and produces a particle approximation that duly represents probabilistic uncertainty in both the route taken and the positions at observation times. Additionally, bmm offers support for both offline and online computation.

Core Functionality

bmm can be used to convert a polyline (ordered series of GPS coordinates) into a collection of possible routes along edges within a graph.

We assume that the graph is stored as a NetworkX (Hagberg et al., 2008) object (which can easily be achieved for a given region using osmnx (Boeing, 2017)) and that the polyline is stored as an array or list of two-dimensional coordinates in the same coordinate system as the graph. A common choice for coordinate system is UTM (Universal Transverse Mercator) which as a square coordinate system (with unit metres) is less cumbersome than the spherical longitude-latitude coordinates system (with unit degrees). bmm can convert longitude-latitude to UTM using the bmm.long_lat_to_utm function.

Offline Map-matching

Given a suitable graph and polyline bmm can be easily used to map-match

```
matched_particles = bmm.offline_map_match(graph, polyline=polyline_utm,
                                          n_samps=100, timestamps=15)
```

Here the n_samps parameter represents the number of particles/trajectories to output and timestamps is the number of seconds between polyline observations - this can be a float if all observation times are equally spaced, an array of length one less than that of the polyline representing the unequal times between observations or an array of length equal to the polyline representing UNIX timestamps for the observation times.

The output of bmm.offline_map_match is a bmm.MMParticles object that contains a particles attributes listing the possible trajectories the algorithm has managed to fit to the polyline - full details can be found at bmm.readthedocs.io.

Online Map-matching

bmm can also map-match data that arrives in an online or sequential manner. Initiate a bmm.MMParticles with the first observation

```
matched_particles = bmm.initiate_particles(graph,
                                           first_observation=polyline_utm[0],
                                           n_samps=100)
```

and then update as new data comes in

```
matched_particles = bmm.update_particles(graph,
                                         matched_particles,
                                         new_observation=polyline_utm[1],
                                         time_interval=15)
```

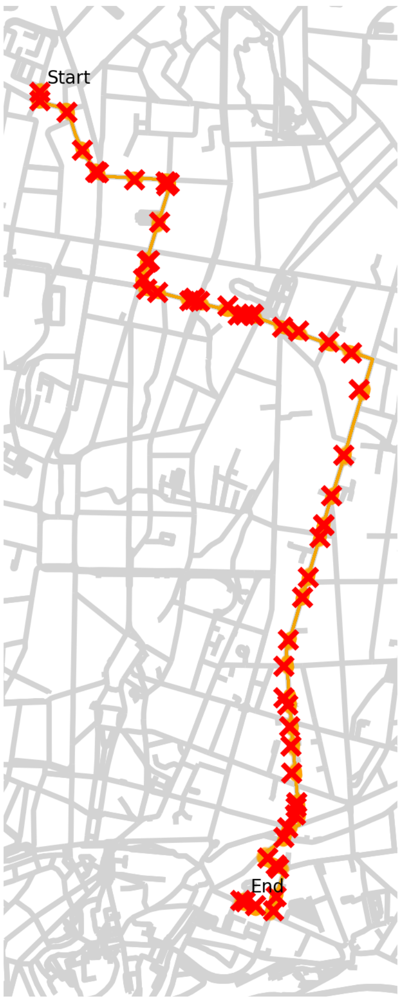
Parameter Tuning

The statistical model described in Duffield & Singh (2022) has various parameters which can be adjusted to fit the features of the graph and time interval setup. This can be done by adjusting a bmm.MapMatchingModel argument or its default bmm.ExponetialMapMatchingModel which is taken as an optional mm_model argument in the above map-matching functions. In addition, these parameters can be learnt from a series of polylines using bmm.offline_em.

Plotting

Once a polyline has been succesfully map-matched, it can be visualised using bmm

```
bmm.plot(graph, particles=matched_particles, polyline=polyline_utm)
```



Acknowledgements

Samuel Duffield acknowledges support from the EPSRC.

References

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, *65*, 126–139. https://doi.org/doi.org/10.1016/j.compenvurbsys.2017.05.004

Duffield, S., & Singh, S. S. (2022). Online particle smoothing with application to map-matching. *IEEE Transactions on Signal Processing*, *70*, 497–508. https://doi.org/10.1109/TSP.2022.3141259

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15).

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Jordahl, K., Bossche, J. V. den, Fleischmann, M., Wasserman, J., McBride, J., Gerard, J., Tratner, J., Perry, M., Badaracco, A. G., Farmer, C., Hjelle, G. A., Snow, A. D., Cochran, M., Gillies, S., Culbertson, L., Bartos, M., Eubank, N., maxalbert, Bilogur, A., … Leblanc, F. (2020). *Geopandas/geopandas: v0.8.1* (Version v0.8.1) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3946761

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A llvm-based python jit compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 1–6.

Luxen, D., & Vetter, C. (2011). Real-time routing with OpenStreetMap data. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 513–516. https://doi.org/10.1145/2093973.2094062

Moreira-Matias, L., Gama, J., Ferreira, M., Moreira, J., & Damas, L. (2013). Predicting taxi-passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, *14*, 1393–1402. https://doi.org/10.1109/TITS.2013.2262376

Newson, P., & Krumm, J. (2009). Hidden markov map matching through noise and sparseness. *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 336–343. https://doi.org/10.1145/1653771.1653818

OpenStreetMap contributors. (2017). *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org

The pandas development team. (2020). *Pandas-dev/pandas: pandas* (latest) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3509134

Yang, C., & Gidofalvi, G. (2018). Fast map matching, an algorithm integrating hidden markov model with precomputation. *International Journal of Geographical Information Science*, *32*(3), 547–570. https://doi.org/10.1080/13658816.2017.1400548