Les membres du jury qui ont évalué ce mémoire ont demandé des corrections mineures.

HEC MONTRÉAL

BKTR : Suite logicielle de régression tensorielle bayésienne à noyau pour données spatio-temporelles avec coefficients évolutifs

par

Julien Lanthier

Aurélie Labbe
HEC Montréal
Codirectrice de recherche

Lijun Sun
Université McGill
Codirecteur de recherche

Sciences de la gestion
(Spécialisation science des données et analytique d'affaires)

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences
(M. Sc.)

Juillet 2023

Résumé

BKTR est une nouvelle suite logicielle d'analyse spatio-temporelle permettant d'appliquer facilement et de manière efficace une régression multivariée sur des ensembles de données qui varient dans le temps et l'espace. BKTR est implémentée de manière indépendante dans les langages Python et R, fournissant ainsi un cadre flexible et performant pour les modèles de régression spatio-temporelle. L'un des principaux défis de la modélisation spatio-temporelle lors de l'utilisation de régression locale est le coût de calcul lui étant associé. BKTR répond à ce défi de calcul en implémentant une régression tensorielle. Cette approche réduit considérablement le coût de calcul du modèle. L'efficacité des calculs est davantage améliorée par l'utilisation d'une bibliothèque spécialisée en calcul tensoriel nommé TORCH. Cette dernière est une librairie, disponible à la fois en R et en Python, qui est fortement optimisée pour les calculs matriciels, lesquels sont également accélérés par l'utilisation de matériel spécialisé comme des GPU ou des TPU. Afin de capturer les dépendances spatiales et temporelles des données, BKTR utilise des processus gaussiens. C'est pourquoi le nom de la bibliothèque BKTR signifie *régression tensorielle bayésienne à noyau* (en anglais *bayesian kernelized tensor regression*). La librairie écrite en Python PYBKTR est disponible sur PyPI et la librairie développée en R qui est nommée BKTR a été soumise à CRAN.

Mots-clés

Processus Gaussien, Régression Tensorielle, Régression Spatiotemporelle Locale, Cadre Bayésien, Logiciel Statistique, R, Python

Méthodes de recherche

Analyse Statistique, Implémentation et Développement de Logiciels, Conception Expérimentale, Analyse de Sensibilité, Études de Cas

Abstract

BKTR is a new software library for spatiotemporal regression analysis with varying coefficients that allows for efficient and easy-to-use inference over datasets that vary in time and space. The library is implemented as Python and R packages, providing a flexible and easy to use framework for spatiotemporal regression models. One of the main challenges in spatiotemporal modeling when using local regression is the computational cost. BKTR addresses this computational challenge by implementing a tensor regression approach. This approach greatly reduces the computational cost of the model. The calculation speed is further improved using the specialized tensor library TORCH (both in R and Python), which enables optimal matrix and tensor computation on GPUs and TPUs. The framework also utilizes Gaussian process (GP) priors to capture the spatial and temporal dependencies of the data. Hence, the full name of the framework, Bayesian Kernelized Tensor Regression, refers to the use of both tensor regression and GP models. The Python PYBKTR package is available on PyPI and the R BKTR package has been submitted to CRAN.

Keywords

Gaussian Process, Tensor Regression, Local Spatiotemporal Regression, Bayesian Framework, Statistical Software, R, Python

Research methods

Statistical Analysis, Implementation and Software Development, Experimental Design, Sensitivity Analysis, Case Studies

Table des matières

Liste des tableaux

Liste des figures

Liste des abréviations

AR Modèle Autorégressif (Autoregressive Model)

ARIMA Modèle Autorégressif à Moyenne Mobile Intégrée (Autoregressive Integrated Moving Average)

ARMA Modèle Autorégressif à Moyenne Mobile (Autoregressive — Moving-Average Model)

BKTR Régression Tensorielle Bayésienne à Noyau (Bayesian Kernelized Tensor Regression)

CP Type de Décomposision CANDECOMP/PARAFAC

CPU Processeur (Central Processing Unit)

CRAN Réseau complet d'archive R (Comprehensive R Archive Network)

GP Processus Gaussien (Gaussian Process)

GPS Système mondial de positionnement (Global Positioning System)

GPU Processeur Graphique (Graphic Processing Unit)

IoT Internet des objets (Internet of Things)

IP Protocole Internet (Internet Protocol)

MAE Erreur Moyenne Absolue (Mean Absolute Error)

MCMC Méthode de Monte-Carlo par chaînes de Markov (Markov chain Monte Carlo)

PyPI Index des Librairies Python (Python Package Index)

RMSE Racine de l'Erreur Quadratique Moyenne (Root-Mean-Square Error)

STVC Modèle de Coefficient Variant dans l'Espace et le Temps (Spatiotemporally Varying Coefficient model)

TPU Unité de Traitement de Tenseur (Tensor Processing Unit)

Avant-propos

Ce mémoire en science des données et analytique d'affaires est rédigé sous forme d'article, et ce, avec l'autorisation de la direction administrative du programme de Maîtrise ès Science à HEC Montréal.

L'article en question a été ajouté au mémoire avec le consentement signé des coauteurs Mengying Lei, Aurélie Labbe et Lijun Sun.

Il est à noter que l'article ajouté n'a pas encore été soumis pour publication. Par conséquent, des changements mineurs pourraient y être apposés avant une soumission finale, notamment suite à des changements requis pour l'acceptation de la librairie R par CRAN.

La rédaction de cet article a été fait dans le but d'être publié dans le *Journal of Statiscal Software*. À ces fins, il est important de noter que le gabarit LaTeX utilisé pour l'article joint provient du journal publication du même nom. Comme la mise en page de l'article est différente de celle du mémoire et que celui-ci a dû être écrit en anglais à des fins de publications, il a été muni d'un encadré lui permettant d'être bien identifié.

Remerciements

J'aimerais commencer par exprimer ma gratitude envers l'établissement d'enseignement d'excellence qu'est le HEC Montréal. Les méthodes pédagogiques, l'enthousiasme des étudiants et l'implication du personnel universitaire sont tous des qualités qui m'ont inspiré et ont permis de réanimer ma passion pour l'académie et la recherche.

Merci énormément à Professeure Aurélie Labbe et Professeur Lijun Sun de m'avoir donné la chance de travailler sur ce projet de grande envergure. L'encadrement continu que vous m'avez accordé me permet aujourd'hui de livrer un projet dont je suis extrêmement fier. Vos conseils précieux lors de nos multiples rencontres m'ont permis d'apprendre énormément et d'être réorienté lors des moments plus difficiles.

Merci aussi à Mengying pour sa participation assidue aux réunions de mise au point et de m'avoir éclairé dans la découverte de ses précédents travaux.

Je remercie spécialement Yamei, mes parents, ma famille et mes amis proches de toujours m'avoir supporté durant ces années de travail.

Merci aussi à Sylvie pour avoir encouragé pendant de nombreuses années la poursuite de mon parcours académique.

Introduction

Le 21e siècle est marqué par une croissance phénoménale des données de toute nature. Cela est d'autant plus vrai depuis l'essor du phénomène de l'Internet des objets (IoT). Le fait qu'il soit possible aujourd'hui de connecter à Internet des objets variés et inédits tel qu'un réfrigérateur, une montre, ou encore un grille-pain a pour effet d'augmenter de manière fulgurante le volume journalier de données générées. On peut aussi affirmer que ces données ont comme points en commun une localisation et un moment particulier d'émission et de collecte. Les données temporelles sont pour la plupart facilement identifiables, considérant que la majorité des informations peuplant les bases de données sont horodatées à l'aide d'estampille temporelle comme *created_at* ou *updated_at*. Grâce à l'adresse IP d'un objet connecté ou encore les coordonnées GPS qui y sont enregistrés, il est aujourd'hui possible d'obtenir une bonne estimation de la position géospatiale qu'avaient ces appareils à chaque moment auquel ceux-ci avaient recueilli des données. La proximité temporelle et spatiale entre ces observations est une mine d'or d'informations qui peut faire ressortir des relations ou des patrons important à l'intérieur même de jeux de données. Ce mémoire a comme objectif d'explorer plus en détails cet aspect qui met en évidence l'importance des processus d'analyse statistique appliqués à des données géospatiales et temporelles.

Un bon exemple de jeux de données contenant tant des informations de nature temporelle que de nature spatiale est celui de BIXI Montréal (2023) une compagnie montréalaise offrant un service de partage de vélo. BIXI est capable de collec-

ter à travers le temps un grand nombre de données grâce à ses senseurs connectés à des stations situées à différentes positions géographiques. Les données recueillies à l'aide de ces senseurs contiennent de l'information sur le nombre d'occasions où les utilisateurs ont utilisé un vélo depuis une station. Ce cas illustre bien à quel point l'IoT est important dans le développement des entreprises ayant une grande composante technologique. BIXI met gratuitement à disposition du public un jeu de données détaillé qui contient de l'information sur la position géographique de chacune de ses stations ainsi que le nombre journalier de départs depuis chacune de celle-ci. Nous présentons, dans cet ouvrage, une version remaniée du jeu de données de la saison 2019 (du 15 avril au 27 octobre) qui contient un total de 587 stations.

Afin de mieux comprendre les relations entre les covariables et la variable réponse présentées dans le jeu de données de BIXI, une technique comme la régression serait idéale. La régression est une technique statistique permettant de modéliser la relation entre une variable réponse (y) et une ou plusieurs variables explicatives ($x_1, x_2, \ldots, x_P$) lui étant associées. Un cas d'utilisation intéressant pour le jeu de données BIXI serait, par exemple, de modéliser le nombre de départs d'une station en fonction de la quantité totale de précipitation et l'humidité relative d'une journée. Pour ce faire, nous pourrions utiliser une régression linéaire multiple. L'équation générale de la régression linéaire multiple peut être représentée sous la forme suivante :

$$\boldsymbol{y} = \beta_0 + \beta_1 \boldsymbol{x}_1 + \beta_2 \boldsymbol{x}_2 + \cdots + \beta_P \boldsymbol{x}_P + \boldsymbol{\epsilon}, \tag{1}$$

où $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_P$ sont les P vecteurs représentants les variables explicatives, β_0 la valeur représentant l'ordonné à l'origine et $\beta_1, \beta_2, \ldots, \beta_P$ sont les P coefficients de régression associés à ces variables explicatives. Pour $\boldsymbol{\epsilon}$, celui-ci représente un vecteur contenant l'erreur du modèle pour chaque observation. Notons qu'à partir de l'équation ci-dessus, il est aussi possible de définir les vecteurs $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_P$ comme étant les vecteurs colonnes de la matrice de design $\boldsymbol{X}$, et les coefficients

de régression $\beta_1, \beta_2, \ldots, \beta_P$ comme étant les éléments du vecteur $\boldsymbol{\beta}$. Ceci nous permet de reformuler l'Équation 1 de la manière suivante :

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{2}$$

La régression linéaire multiple permet facilement de modéliser différents types de phénomènes ayant plusieurs covariables. Dans un langage de programmation statistique comme R (R Core Team, 2022), cette méthode de régression est facilement accessible. Elle est notamment présente dans les librairies de base de R sous la fonction `lm`.

Dans plusieurs cas de figure, il est très probable que l'aspect temporel des données soit un facteur important et que celui-ci nécessite d'être pris en compte. Plus précisément, cela signifie qu'une variable réponse y_t prise à un temps t est influencée par les variables réponses $y_0, \ldots, y_{t-1}$ observées à des pas de temps précédents $0 \ldots t-1$. L'idée d'inclure les résultats antérieurs de la variable réponse dans l'explication de celle-ci correspond à l'application de modèles de processus autorégressifs. Une classe particulière de ce type de modèle est le processus autorégressif linéaire d'ordre 1 (AR(1)) qui peut être formulé comme suit :

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t, \quad t = 1, 2, \ldots, N, \tag{3}$$

où ϵ_t est le terme d'erreur de prédiction à l'instant t et N est le nombre de pas de temps dans les données. Il existe plusieurs autres modèles de processus autorégressif permettant de faire ressortir des motifs temporels dans les données. Certains exemples de ces motifs sont les tendances, la saisonnalité, les cycles et l'effet de jours fériés. Des modèles plus complexes de séries temporelles peuvent également être utilisés pour modéliser ces patrons, par exemple les modèles autorégressifs à moyenne mobile (ARMA) et les modèles autorégressifs à moyenne mobile intégrée (ARIMA). Ces modèles sont largement utilisés pour la modélisation de séries temporelles, mais ceux-ci ne seront pas abordés dans ce travail. Il existe plusieurs ouvrages traitant de ces méthodes et de leur utilisation et nous

référons notamment le lecteur à l'ouvrage de Box et collab. (2016) pour plus de détails sur ceux-ci. Par ailleurs, plusieurs logiciels ont déjà été développés pour permettre de modéliser ces modèles dans le langage de programmation statistique R. Par exemple, le paquet logiciel STATS, qui est inclus de base dans R, permet d'utiliser plusieurs types de processus temporels. Pour une utilisation plus avancée des régressions temporelles en R, nous référons au lecteur le paquet ZOO (Zeileis et Grothendieck, 2005) qui offre des fonctionnalités telles que l'imputation de données manquantes et la manipulation des séries temporelles irrégulières.

Une approche alternative pouvant être utilisée dans le cas de la régression temporelle consiste à tenir compte du vecteur complet des observations temporelles et à utiliser des méthodes basées sur les noyaux pour modéliser la covariance. Ce type d'approche repose sur l'utilisation de processus Gaussien (GP) pour représenter la distribution des variables aléatoires sous la forme d'une distribution Gaussienne multivariée. En utilisant directement la définition de Rasmussen et Williams (2006), on peut décrire les processus Gaussien comme une collection de variables aléatoires, dont tout nombre fini possède une distribution gaussienne conjointe. Un processus gaussien spécifique peut être entièrement défini par sa fonction moyenne $m(x)$ et sa fonction de covariance $k(x, x')$. Il est donc possible de définir un processus réel $f(x)$ de la manière suivante :

$$f(x) \sim \mathcal{GP}\big(m(x), k(x, x')\big). \tag{4}$$

Dans plusieurs cas il est possible et plus simple de considérer la fonction moyenne comme nulle $m(x) = 0$ et c'est d'ailleurs ce qui sera utilisé tout au long de ce mémoire. En utilisant des fonctions noyaux appropriées (ou *kernels* en anglais), il est possible de capturer les motifs de corrélation temporelle entre les observations. Les noyaux agissent comme des fonctions décrivant la relation entre deux observations dans un espace donné. Ces fonctions peuvent prendre plusieurs formes et utiliser différents ensembles de paramètres. Dans la Table 1.3, nous présentons

un sous-ensemble de différentes fonctions noyaux permettant chacune de capturer différents motifs relationnels entre les observations. Dans un contexte temporel, on définit la matrice de covariance ($\boldsymbol{K}_t$) comme une matrice symétrique positive semi-définie étant construite à partir d'une fonction noyau $k_t(t_n, t_{n'}; \Gamma)$. Cette fonction k_t utilise un ensemble Γ de L paramètres prédéfini $\{\gamma_1, \ldots, \gamma_L\}$ et permet de mesurer la similarité entre N observations temporelles une paire $(t_n, t_{n'})$ à la fois. L'un des noyaux les plus utilisés dans le domaine temporel est le noyau de type exponentiel ou *Squared Exponential* (SE) en anglais. Ce noyau permet d'obtenir une matrice de covariance exprimant une tendance régulière de décroissance en fonction de la distance, et il est présenté plus en détail à la Section 1.4.2 de cet ouvrage. En fixant le paramètre d'échelle (ℓ) de la fonction SE et en l'appliquant à un ensemble d'observation donné, il est possible d'obtenir une matrice prédéfinie illustrant le motif de décroissance mentionné précédemment. C'est d'ailleurs ce qu'on peut visualiser dans l'illustration la plus à gauche de la Figure 1.3.

Le même concept de modélisation utilisant des processus Gaussien peut également être appliqué à des données de type spatiale. Dans ce contexte, il est possible de calculer une distance entre les observations à l'aide des coordonnés spatiale et de capturer la dépendance spatiale via la matrice de covariance $\boldsymbol{K}_s$ formée par la fonction noyau spatiale $k_s(s_m, s_{m'}; \Phi)$. Cette fonction permet de mesurer la covariance établie entre M localisations spatiales en utilisant un ensemble de paramètre Φ donné. Un exemple couramment utilisé de noyau dans un contexte spatial est le noyau Matérn. Ce noyau flexible permet de capturer des structures ayant des variations lisses ou rugueuses dans l'espace étudiée. De plus, ce noyau est étroitement relié à la théorie géostatistique et est basé sur la théorie des variogrammes (Minasny et McBratney, 2005), ce qui contribue à sa popularité dans les études produites pour un contexte géographique.

Il existe plusieurs autres types de noyaux pouvant être utilisé dans un cadre spatial et temporel. Ceux-ci sont abordés plus en détail à la Section 1.4.2 de ce mé-

moire. Par ailleurs, nous recommandons vivement aux lecteurs souhaitant obtenir plus d'informations détaillées sur les processus Gaussiens de consulter *Gaussian processes for machine learning* (Rasmussen et Williams, 2006) qui constitue un ouvrage de référence sur le sujet.

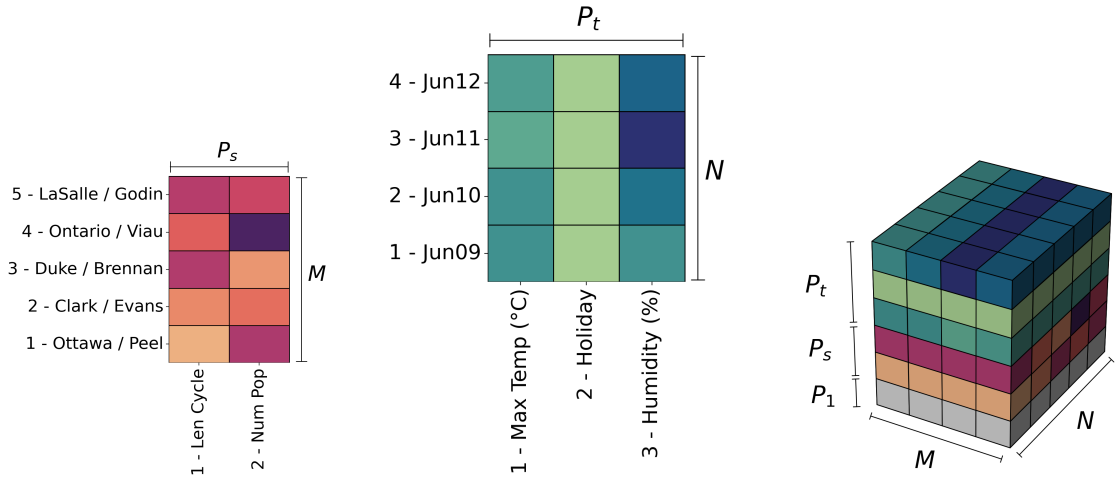
Pour le cas BIXI, il est possible d'étudier un ensemble de données qui contient à la fois des variables temporelles et spatiales. Ces données ont été préalablement rassemblées et préparées par Lei et collab. (2023). Le jeu de données BIXI initial ne contenait que le nombre de départs par station par jour et les coordonnées géographiques (longitude et latitude) de chaque station. Pour aider à mieux modéliser cette étude de cas, Lei et collab. (2023) ont rattaché des données provenant de source tierce à chacun des $N = 196$ points temporels (correspondant à une journée calendrier) et à chacune des $M = 587$ stations. Au total, $P_t = 5$ données temporelles ont été obtenues de cette manière, principalement des données climatiques telles que l'humidité relative et la température moyenne, provenant d'Environnement Canada. Une variable binaire indiquant si un jour était ou non férié a également été rassemblée à partir des données du Gouvernement du Québec. En ce qui concerne les variables spatiales étudiées, un total de $P_s = 13$ covariables différentes ont été rattachées aux $M = 587$ stations. Ces covariables proviennent d'une précédente étude de Wang et collab. (2021) qui jumelait chaque station à des informations sur la population, l'infrastructure publique et les points d'intérêts à proximité (e, g. université, restaurant, stations de métro, etc.). Ainsi, le jeu de données résultant contient trois parties :

— Une matrice pour la variable réponse $\boldsymbol{Y}$ contenant M ligne et N colonnes

— Une matrice contenant les covariables spatiales avec M lignes et P_s colonnes

— Une matrice contenant les covariables temporelles avec N lignes et P_t colonnes

Pour mieux visualiser la structure de cet ensemble de données, nous avons inclus l'illustration des covariables spatiales et des covariables temporelles dans



(a) Covariables Spatiales (b) Covariables Temporelles (c) Tenseur de covariables

FIGURE 0.1 – Illustration des valeurs et des dimensions des covariables spatiales et temporelles pour un jeu de données comprenant $M = 5$ localisations spatiales, $P_s = 2$ covariables spatiales, $N = 4$ points temporels et $P_t = 3$ covariables temporelles. Illustration d'un tenseur $\boldsymbol{\mathcal{B}}$ assemblé à partir de matrices de covariables et d'une matrice intercepte dans le but d'effectuer une régression spatiotemporelle locale.

la Figure 0.1a et Figure 0.1b, respectivement. Ces figures représentent un sous-ensemble abrégé de l'ensemble de données complet ayant $M = 5, N = 4, P_t = 3, P_s = 2$.

En examinant la structure de l'ensemble de données BIXI, il est évident que la variable réponse $\boldsymbol{Y}$ est influencée à la fois par des composantes temporelles et des composantes spatiales. Dans ce type de problème, la régression spatio-temporelle se révèle être un outil efficace qui permet de prendre en compte les motifs complexes pouvant être observés à travers le temps et l'espace (via différentes stations). En regardant ces deux aspects à la fois, il est clair qu'une méthode de régression spatio-temporelle pourrait nous permettre de mieux modéliser ces données et aussi d'obtenir des résultats de prédictions plus précis.

Dans un contexte spatio-temporel, plusieurs méthodes de régression sont disponibles afin de modéliser les relations entre la variable réponse et ses covariables en prenant en compte à la fois l'aspect temporel et spatial. Un élément clé en lien

avec ces méthodes est la distinction entre la régression globale et la régression locale. La régression globale utilise un ensemble de coefficients commun pour toutes les observations, ce qui fait en sorte qu'il est plus difficile de modéliser des motifs localisés dans une région temporelle ou spatiale donnée. En contrepartie, la régression locale, permet d'utiliser des coefficients pouvant varier à travers le temps et l'espace, ce qui permet de capturer des motifs étant non uniformes dans ces espaces. Néanmoins, la régression spatio-temporelle pose un défi de calcul important relié au nombre de paramètres que celui-ci doit évaluer. Dans le cas BIXI, si l'on souhaite modéliser une régression locale au niveau spatial, il faut estimer des coefficients distincts pour 587 stations, ce qui est beaucoup plus coûteux en termes de calcul qu'une régression globale. De ce fait, la régression locale est, en pratique, difficile à utiliser sur de grands jeux de données.

Certaines suites logicielles aujourd'hui permettent d'utiliser des régressions locales au niveau spatial en plus de prendre en compte l'aspect de proximité temporel via des modèles autorégressifs. C'est le cas notamment des librairies comme SPBAYES (Finley et collab., 2015) et SPTIMER (Bakar et Sahu, 2015).

Dans notre cas, nous nous concentrons spécifiquement sur les modèles qui présentent une variation de coefficients dans l'espace et dans le temps, permettant ainsi de capturer des motifs variant localement dans ces deux dimensions. L'équation caractérisant un modèle de cette forme peut être présentée sous la forme suivante :

$$y(s_m, t_n) = \boldsymbol{x}(s_m, t_n)^T \boldsymbol{\beta}(s_m, t_n) + \epsilon(s_m, t_n), \tag{5}$$

où $y(s_m, t_n)$ est la variable réponse à un emplacement s_m et un temps t_n, $x(x_m, t_n)$ et $\beta(s_m, t_n)$ sont les covariables et les coefficients à l'emplacement s_m et le temps t_n respectivement.

Cette équation montre que cela revient à ajuster un modèle linéaire en tout point spatio-temporel distinct. Conséquemment, il faut évaluer pour le produit cartésien de tout point spatial et tout point temporel un vecteur de coefficients

$\boldsymbol{\beta}$. Cet aspect de produit cartésien entraine l'utilisation d'un nombre considérable de coefficients et de covariables. Pour simplifier la formulation subséquente, il est possible de réarranger les coefficients $\boldsymbol{\beta}$ et les covariables $\boldsymbol{x}$ sous forme tensorielle $\boldsymbol{\mathcal{B}}$ et $\boldsymbol{\mathcal{X}}$ respectivement.

La Figure 0.1c montre à quoi ressemble un tenseur $\boldsymbol{\mathcal{X}}$ pour l'exemple réduit que nous avions introduits précédemment pour l'utilisation de ce type de régression locale. Pour ce cas réduit, nous voyons que le modèle doit utiliser un tenseur de taille $(M = 4) \times (N = 5) \times (P = 1 + 2 + 3)$ contenant 120 éléments. Cela veut dire que même pour ce petit exemple, il faut ajuster 120 coefficients lorsqu'on utilise ce type de modèle. Dans un grand jeu de données comme celui de BIXI, pour ajuster une régression locale, il faut estimer plus de 2 millions de coefficients. Certaines approches ont essayé d'intégrer la régression spatio-temporelle locale. C'est le cas notamment du modèle de coefficients spatiotemporels variants (STVC) proposé par (Gelfand et collab., 2003). Cependant, Lei et collab. (2023) ont su montrer que même pour des jeux de données de taille modérée (i.e, $M = 100$, $N = 100$ et $P = 10$), cette méthode est trop exigeante en temps de calcul pour obtenir des résultats. À notre connaissance, aucune autre librairie permet en date d'aujourd'hui d'implémenter une régression locale comme formulé dans l'Équation 5 sur de large jeux de données dans un temps raisonnable.

C'est pourquoi, c'est avec enthousiasme que nous vous présentons dans le chapitre qui suit un article faisant l'introduction des librairies BKTR et PYBKTR. Ces librairies sont écrites dans deux langages de programmation fortement utilisés dans le domaine statistique, soit R et Python. Ces bibliothèques permettent d'utiliser une régression locale sur des jeux de données volumineux, et ce, de manière flexible, efficace et rapide. Cette librairie rend accessible au grand publique les avancées proposées par l'ouvrage de Lei et collab. (2023).

Chapitre 1

BKTR : An efficient spatiotemporally varying coefficient regression package in R and Python

BKTR: An efficient spatiotemporally varying coefficient regression package in R and Python

Julien Lanthier
HEC Montréal

Mengying Lei
McGill University

Aurélie Labbe
HEC Montréal

Lijun Sun
McGill University

Abstract

BKTR is a new software library for spatiotemporal regression analysis with varying coefficients that allows for efficient and easy-to-use inference over datasets that vary in time and space. The library is implemented as Python and R packages, providing a flexible and easy to use framework for spatiotemporal regression models. One of the main challenges in spatiotemporal modeling when using local regression is the computational cost. **BKTR** addresses this computational challenge by implementing a tensor regression approach. This approach greatly reduces the computational cost of the model. The calculation speed is further improved using the specialized tensor library **torch** (both in R and Python), which enables optimal matrix and tensor computation on GPUs and TPUs. The framework also utilizes Gaussian process (GP) priors to capture the spatial and temporal dependencies of the data. Hence, the full name of the framework, Bayesian Kernelized Tensor Regression, refers to the use of both tensor regression and GP models. The Python **pyBKTR** package is available on PyPI and the R **BKTR** package has been submitted to CRAN.

Keywords: Gaussian process, Tensor regression, Local spatiotemporal regression, R, Python.

1. Introduction

With the rise of the Internet of Things (IoT) and a great increase of mobile device and sensor usage, the amount of available data varying through time and space has been growing rapidly. Thus, statistical analysis like spatiotemporal regressions that take into account the spatial and temporal aspects of data have become more and more valuable. Spatiotemporal regressions are especially useful for analyzing and predicting complex phenomena like weather patterns, agriculture output, transportation, disease outbreaks and much more. It is possible to regroup spatiotemporal regressions into two main categories: global and local regressions. The main difference between these two types of regressions are in their approach for modelling relationships between variables over space and time. Global regression assumes that the relationships between variables are constant across space and time. In contrast, local regression allows for the relationship between variables to vary across different locations and time points. Local regression is often viewed as a more flexible and better suited method to

capture changes in variables' relationships over time and space.

Even if local regression is usually more flexible and leads to better fit, this method is much more computationally expensive than global regression. This is due to the fact that local regression needs to estimate coefficients at each location and time point studied. In fact, for a response matrix $Y \in \mathbb{R}^{M \times N}$ observed from a set of locations $S = \{s_1, \ldots, s_M\}$ and a set of time points $T = \{t_1, \ldots, t_N\}$, we can define the model over a Cartesian product $S \times T = \{(s_m, t_n) : m = 1, \ldots, M,\ n = 1, \ldots, N\}$ and we can formulate local spatiotemporal regression as:

$$y(s_m, t_n) = \boldsymbol{x}(s_m, t_n)^T \boldsymbol{\beta}(s_m, t_n) + \epsilon(s_m, t_n), \tag{1}$$

where $y(s_m, t_n)$ is the response variable at location s_m and time t_n, $x(x_m, t_n)$ and $\beta(s_m, t_n)$ are the covariates and coefficients at location s_m and time t_n respectively. Local spatiotemporal regression already has been explored and implemented in the literature. For example, Gelfand *et al.* (2003) already suggested using a separable kernel to build a spatiotemporally varying coefficient model (STVC). However, even with the advent of new computing technologies, local regression for spatiotemporal data methodologies like STVC are still unable to run on even medium-sized datasets, due to the sheer size of local regression's number of parameters. To our knowledge there is, to this day, no readily available software packages allowing to use in an efficient manner a local spatiotemporal regression. The scalable spatiotemporally varying coefficient modelling with bayesian kernelized tensor regression (BKTR) method proposed by Lei *et al.* (2023) overcomes this limitation by using a tensor regression to estimate coefficients. In doing so, the time complexity of each sampling iteration changes from $\mathcal{O}(M^3N^3P^3)$ for the STVC method to $\mathcal{O}(R^3(M^3+N^3+P^3))$, where R is usually an arbitrary small value denoting the tensor rank. The BKTR method also uses a Gaussian process prior with a spatial and a temporal kernel to model the spatiotemporal dependence of the coefficients.

The packages presented in this paper, **BKTR** and **pyBKTR**, implement the BKTR method and provide a user-friendly interface to estimate coefficients for local spatiotemporal regression. The **BKTR** package is implemented in R (R Core Team 2017) and is available on CRAN. The **pyBKTR** package is implemented in Python and is available on PyPI (`https://pypi.org/project/pyBKTR`). Both packages provide the same functionalities and are designed to be used in a similar fashion. To mimic the object-oriented patterns of Python, we used the **R6** package (Chang 2021) in **BKTR** for classes and methods. Also, to be able to use a similar approach for tensor operations, we used torch in both R and Python packages which translate to **torch** (Falbel and Luraschi 2023) and **pytorch** (Paszke *et al.* 2019) respectively. The visualization of the results is done using the **ggplot2** package (Wickham 2016) in R and **plotly** (Plotly Technologies Inc. 2015) in Python. Furthermore, to be able to use Wilkinson formulae (Wilkinson and Rogers 1973), like in the R formula object, we use the **Formulaic** (Wardrop 2022) python package. For dataframe usage, we use the **pandas** package (The pandas development team 2022) in Python and **data.table** (Dowle and Srinivasan 2023) in R. All the examples shown in this paper are available in a GitHub repository (`https://github.com/julien-hec/bktr-examples`). This paper focuses mainly on the R implementation of **BKTR**, but the syntaxes of the R and Python packages we implemented are very similar. To convert covered examples from R to Python, it should be sufficient to convert all base 1 indexes to base 0 and to change "`$`" to "`.`", "`<-`" to "`=`" and "`$new()`" to "`()`". Also, the source code and all examples shown in this paper are available on GitHub at `https://github.com/julien-hec/BKTR/` and `https://github.com/julien-hec/pyBKTR/` for **BKTR** and **pyBKTR** respectively. Note that when we refer to the **BKTR** packages in this

paper (plural form), we are referring to both the R and Python packages at the same time.

The rest of this paper is organized as follows. Section 2 presents the BKTR algorithm. Section 3 presents the `BKTRRegressor` class and its attributes. Section 4 presents the different kernels available in the package. Section 5 presents a simulation study to validate our implementation of the BKTR regression. Section 6 presents an experimental study on bike sharing data. Finally, Section 7 concludes the paper.

2. BKTR algorithm

The objective of Bayesian Kernelized Tensor Regression (BKTR) is to model a response variable $\boldsymbol{Y}$ as a function of spatiotemporal covariates $\boldsymbol{\mathcal{X}}$, with the model's coefficients allowed to vary over space and time. The covariates $\boldsymbol{\mathcal{X}}$ represent a tensor of size $M \times N \times P$ where M is the number of locations, N is the number of time points and P is the number of covariates at each location and time point we want to model. The response variable $\boldsymbol{Y}$, for its part, is a matrix of size $M \times N$. An example of application could be to model the housing price of M district through N months, using covariates changing through time and space like temperature, population density and air pollution ($P = 3$).

This section is strongly based on the work of Lei *et al.* (2023) and aims to summarize the BKTR model they described. In Section 2.3, we discuss a previously unexplored aspect of BKTR regarding interpolation.

2.1. Model definition

It is possible to reshape the covariates tensor $\boldsymbol{\mathcal{X}}$ and the response variable $\boldsymbol{Y}$ mentioned above to obtain a vectorized version of Equation 1:

$$\boldsymbol{y} = (\boldsymbol{I}_{MN} \odot \boldsymbol{X}_{(3)})^\top \text{vec}(\boldsymbol{B}_{(3)}) + \boldsymbol{\epsilon}, \tag{2}$$

where $\boldsymbol{y}$ is the vectorized version in $\mathbb{R}^{MN}$ of $\boldsymbol{Y}$, $\boldsymbol{X}_{(3)}$ and $\boldsymbol{B}_{(3)}$ are the mode-3 unfolding of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{B}}$ respectively. The $\odot$ operator is the Khatri-Rao product and the product $\boldsymbol{I}_{MN} \odot \boldsymbol{X}_{(3)}$ is a sparse expansion of the covariates. The error term $\boldsymbol{\epsilon}$ is assumed to follow a multivariate normal distribution such that $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \tau^{-1}\boldsymbol{I}_{MN})$, where $\boldsymbol{I}_{MN}$ is an identity matrix of size $MN \times MN$. Assuming that $\boldsymbol{\mathcal{B}}$ admits a CANDECOMP/PARAFAC (CP) (Kolda and Bader 2009) decomposition with rank $R \ll \min\{M, N\}$, i.e. $\boldsymbol{\mathcal{B}} = \sum_{r=1}^{R} \boldsymbol{u}_r \circ \boldsymbol{v}_r \circ \boldsymbol{w}_r$, the model can be rewritten as:

$$\boldsymbol{y} = \tilde{\boldsymbol{X}} \text{vec}(\boldsymbol{W}(\boldsymbol{V} \odot \boldsymbol{U})^\top) + \boldsymbol{\epsilon}, \tag{3}$$

where $\tilde{\boldsymbol{X}} = (\boldsymbol{I}_{MN} \odot \boldsymbol{X}_{(3)})^\top$ and $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ are matrices of size $M \times R$, $N \times R$ and $P \times R$, respectively, containing the concatenated vectors of the CP decomposition of $\boldsymbol{\mathcal{B}}$.

To account for missing values in the response variable $\boldsymbol{Y}$, we can rewrite the model of Equation 3 as:

$$\boldsymbol{y}_\Omega \sim \mathcal{N}\left(\boldsymbol{O}\left(\tilde{\boldsymbol{X}} \text{vec}(\boldsymbol{W}(\boldsymbol{V} \odot \boldsymbol{U})^\top)\right), \tau^{-1}\boldsymbol{I}_{|\Omega|}\right). \tag{4}$$

where $\boldsymbol{y}_\Omega$ is the vectorized version of $\boldsymbol{Y}$ restricted to the observed entries Ω, $\boldsymbol{O}$ is the operator selecting the observed entries of a vector and $|\Omega|$ is the number of observed entries. Knowing this information, we can state that $\boldsymbol{y}_\Omega = \boldsymbol{O}\boldsymbol{y}$. To account for the spatial and temporal

correlation during CP decomposition, we use the following GP priors on the spatial and temporal component vectors:

$$\begin{aligned} \boldsymbol{u}_r &\sim \mathcal{GP}(0, \boldsymbol{K}_s),\ r = 1, \ldots, R, \\ \boldsymbol{v}_r &\sim \mathcal{GP}(0, \boldsymbol{K}_t),\ r = 1, \ldots, R, \end{aligned} \tag{5}$$

where $\boldsymbol{K}_s$ and $\boldsymbol{K}_t$ are covariance matrices of size $M \times M$ and $N \times N$ respectively created from two kernel functions $k_s(s_m, s_{m'}; \Phi)$ and $k_t(t_n, t_{n'}; \Gamma)$, where Φ is a vector of J spatial kernel hyperparameters $\Phi = \{\phi_1, \ldots, \phi_J\}$ and Γ is a vector of L temporal kernel hyperparameters $\Gamma = \{\gamma_1, \ldots, \gamma_L\}$. The priors of kernel hyperparameters are defined as:

$$\begin{aligned} \log(\phi_i) &\sim \mathcal{N}(\mu_{\phi_i}, \tau_{\phi_i}^{-1}),\ i = 1, \ldots, J, \\ \log(\gamma_i) &\sim \mathcal{N}(\mu_{\gamma_i}, \tau_{\gamma_i}^{-1}),\ i = 1, \ldots, L. \end{aligned} \tag{6}$$

It can be noted that for BKTR $\boldsymbol{K_s}$ and $\boldsymbol{K_t}$ are in fact correlation matrices since we set the kernel variance to 1 and that we capture the variance through $\boldsymbol{W}$.

For the components of the factor matrix $\boldsymbol{W}$, we use the following prior:

$$\begin{aligned} \boldsymbol{w}_r &\sim \mathcal{GP}(0, \boldsymbol{\Lambda}_w^{-1}),\ r = 1, \ldots, R, \\ \boldsymbol{\Lambda}_w &\sim \mathcal{W}(\boldsymbol{\Psi_0}, \nu_0), \end{aligned} \tag{7}$$

where $\boldsymbol{\Psi_0}$ is a $P \times P$ scale matrix and ν_0 is the degrees of freedom.

We also suppose a prior on the noise precision τ from Equation 4 that is $\tau \sim \text{Gamma}(a_0, b_0)$.

An illustration of the BKTR framework taken from (Lei *et al.* 2023) is shown in Figure 1 to help visualize the dependencies between variables and the different steps of the algorithm.

2.2. Sampling

This section only gives a brief overview of the sampling algorithm and its main steps which are described in the algorithm 1. More details about the conditional posterior distributions from which BKTR parameters are sampled can be found in the work of Lei *et al.* (2023).

The BKTR Markov chain Monte Carlo (MCMC) algorithm uses Gibbs sampling (Geman and Geman 1984) for the parameters $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}, \tau$ and the precision matrix $\boldsymbol{\Lambda}_w$. For the hyperparameters Φ and Γ, a slice sampling algorithm (Neal 2003) is used since the conditional posterior distribution of these parameters is not easy to sample from.

The sampling process uses K_1 burn-in iterations to help reaching the stationary distribution of the Markov chain and K_2 iterations to sample the posterior distribution of the parameters. The number of iterations K_1 and K_2 are arbitrary, chosen by the user and should be large enough to ensure the Markov chains reach stationary.

After the sampling process, the samples $\{\boldsymbol{\mathcal{B}}^{(k)}\}_{k=1}^{K_2}$ are used to approximate the posterior distribution of the coefficients $\boldsymbol{\mathcal{B}}$. The posterior coefficients can then be used directly to estimate unobserved data and to analyze the spatial and temporal patterns of the data.

2.3. Interpolation

An important addition that we propose to the BKTR algorithm is to bring the capacity to do interpolation on new data. By interpolation, we mean the ability to estimate new beta

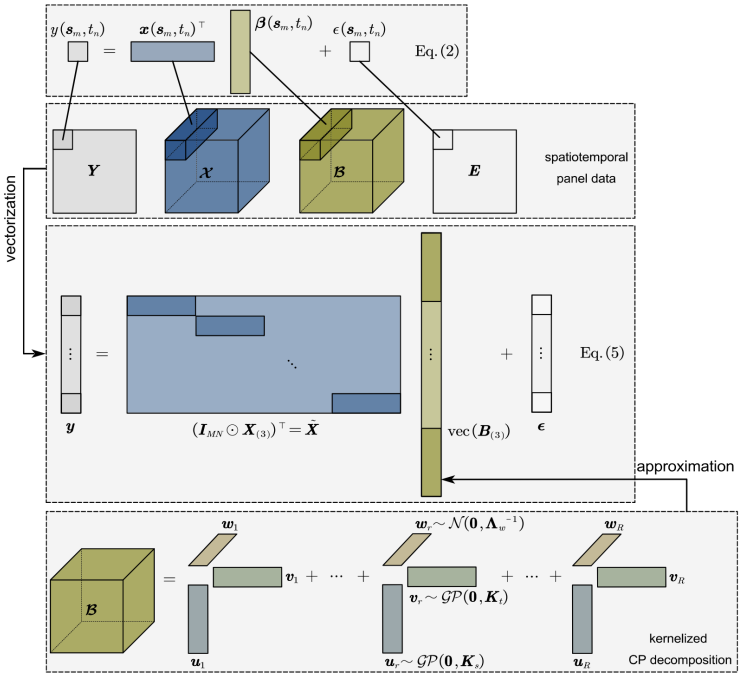


Figure 1: Illustration of the BKTR framework (Source: Figure 1 from Lei *et al.* (2023))

coefficients $\boldsymbol{\mathcal{B}}^{\text{new}}$ and response values $\boldsymbol{Y}^{\text{new}}$ at unobserved time points and locations. In the literature, this process is also often named bayesian kriging. Interpolation is a different process than imputation, which was already covered in BKTR. Imputation is used when parts of the response variables are missing at some of the M^{obs} locations or N^{obs} time points employed during regression. In contrast, interpolation is accomplished in a completely different step after MCMC sampling, and it is performed at M^{new} new locations and N^{new} time points. To perform interpolation, we need to estimate, for M^{new} unobserved locations and N^{new} unobserved time steps, the posterior distributions of the related beta coefficients $\boldsymbol{\mathcal{B}}^{\text{new}}$. For each new location that we want to interpolate, we need to estimate at this site the beta coefficients' posterior distributions of P features for N^{obs} observed time points and N^{new} new time points. In the same way, for each of the N^{new} interpolated time point, we need to estimate the beta coefficients posterior distributions of P features for M^{obs} observed locations and M^{new} new locations. The representation and dimensions of the $\boldsymbol{\mathcal{B}}^{\text{new}}$ coefficients can be somewhat difficult to visualize. Thus, we use a representation similar to the one presented by Takeuchi *et al.* (2017) and we illustrate it in Figure 2. Using this representation, the prediction results of $\boldsymbol{\mathcal{B}}^{\text{new}}$ can be represented by three tensors $\boldsymbol{\mathcal{B}}^1$ which is the beta coefficients for new locations at observed time points), $\boldsymbol{\mathcal{B}}^2$ (beta coefficients for new time points at observed locations) and $\boldsymbol{\mathcal{B}}^3$ (beta coefficients for new locations at new time points). We also include in Figure 2, the equivalent illustration for the newly provided covariates $\boldsymbol{\mathcal{X}}^{\text{new}}$ composed of $\boldsymbol{\mathcal{X}}^1$, $\boldsymbol{\mathcal{X}}^2$, $\boldsymbol{\mathcal{X}}^3$, on top of the related new response variable matrices $\boldsymbol{Y}^{\text{new}}$ composed of $\boldsymbol{Y}^1$, $\boldsymbol{Y}^2$

Algorithm 1 Simplified BKTR MCMC sampling process

Input: $\boldsymbol{y}_\Omega, \boldsymbol{\mathcal{X}}, R, K_1, K_2, k_s, k_t, \Phi, \Gamma, \tau_\phi, \tau_\gamma, a_0, b_0, \tau$
Intialize $\{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}\}$ as normally distributed random values
Set $\mu_{\phi_i} = \log(\phi_i)\ \forall \phi_i \in \Phi,\ \mu_{\gamma_i} = \log(\gamma_i)\ \forall \gamma_i \in \Gamma$
Sample $\Lambda_\omega \sim \mathcal{W}(\boldsymbol{I}_P, P)$
for $k = 1 : K_1 + K_2$ **do**
    Sample kernel hyperparameters Φ, Γ
        ▷ from slice sampling using $\boldsymbol{V}, \boldsymbol{W}, y_\Omega, \boldsymbol{\mathcal{X}}, \Phi, \tau_{\phi_1}, \ldots, \tau_{\phi_J}, \Gamma, \tau_{\gamma_1} \ldots, \tau_{\gamma_L}$
    Sample hyperparameters $\boldsymbol{\Lambda}_w$ ▷ from Wishart distribution using $\boldsymbol{W}$
    Sample factor vec($\boldsymbol{U}$) ▷ from Normal distribution using $\boldsymbol{W}, \boldsymbol{V}, y_\Omega, \boldsymbol{\mathcal{X}}, \tau, k_s$
    Sample factor vec($\boldsymbol{V}$) ▷ from Normal distribution using $\boldsymbol{W}, \boldsymbol{U}, y_\Omega, \boldsymbol{\mathcal{X}}, \tau, k_t$
    Sample factor vec($\boldsymbol{W}$) ▷ from Normal distribution using $\boldsymbol{U}, \boldsymbol{V}, y_\Omega, \boldsymbol{\mathcal{X}}, \tau, \Lambda_\omega$
    Sample precision τ ▷ from Gamma distribution using $\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}, y_\Omega, \boldsymbol{\mathcal{X}}, a_0, b_0$
    if $k > K_1$ **then**
        Collect the samples $\boldsymbol{U}^{(k-K_1)} = \boldsymbol{U}, \boldsymbol{V}^{(k-K_1)} = \boldsymbol{V}, \boldsymbol{W}^{(k-K_1)} = \boldsymbol{W}$;
        Compute and collect $\boldsymbol{\mathcal{B}}^{(k-K_1)}$ using $\boldsymbol{\mathcal{B}} = \sum_{r=1}^{R} \boldsymbol{u}_r \circ \boldsymbol{v}_r \circ \boldsymbol{w}_r$
    end if
end for
return $\{\boldsymbol{\mathcal{B}}^{(k)}\}_{k=1}^{K_2}$ to approximate posterior coefficients and estimate unobserved data.

and $\boldsymbol{Y}$[3].

We estimate the distributions of $\boldsymbol{\mathcal{B}}^{\text{new}}$ via MCMC sampling with an approach similar to the one presented by Gamerman *et al.* (2008). The posterior coefficients $\boldsymbol{\mathcal{B}}^{\text{new}}$ are estimated by doing interpolation on the spatial decomposition $\boldsymbol{U}$ and the temporal decomposition $\boldsymbol{V}$ separately. We can formulate the joint multivariate normal distribution for the r^{th} component of the spatial decomposition which includes the observed ($\boldsymbol{u}_r^{\text{obs}}$) and the new ($\boldsymbol{u}_r^{\text{new}}$) decomposition vectors as follows:

$$\begin{pmatrix} \boldsymbol{u}_r^{\text{obs}} \\ \boldsymbol{u}_r^{\text{new}} \mid \Phi \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \boldsymbol{0}_{M^{\text{obs}}} \\ \boldsymbol{0}_{M^{\text{new}}} \end{pmatrix}, \begin{pmatrix} \boldsymbol{R}_\Phi^{\text{obs}} & \boldsymbol{R}_\Phi^{\text{obs,new}} \\ \boldsymbol{R}_\Phi^{\text{new,obs}} & \boldsymbol{R}_\Phi^{\text{new}} \end{pmatrix} \right], \tag{8}$$

where $\boldsymbol{R}_\Phi^{\text{obs}}$ is the covariance matrix corresponding to the observed locations and $\boldsymbol{R}_\Phi^{\text{new}}$ is the covariance matrix corresponding to the unobserved locations for a known set of spatial kernel hyperparameters Φ. The covariance matrix between the observed and unobserved locations is denoted by $\boldsymbol{R}_\Phi^{\text{obs,new}}$ and $\boldsymbol{R}_\Phi^{\text{new,obs}}$ is its transpose. $\boldsymbol{0}_{M^{\text{new}}}$ and $\boldsymbol{0}_{M^{\text{obs}}}$ are zero vectors of size M^{new} and M^{obs} respectively representing the mean of the decompositions.

The same approach can be used to formulate the joint multivariate normal distribution for the temporal decomposition $\boldsymbol{V}$:

$$\begin{pmatrix} \boldsymbol{v}_r^{\text{obs}} \\ \boldsymbol{v}_r^{\text{new}} \mid \Gamma \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \boldsymbol{0}_{N^{\text{obs}}} \\ \boldsymbol{0}_{N^{\text{new}}} \end{pmatrix}, \begin{pmatrix} \boldsymbol{R}_\Gamma^{\text{obs}} & \boldsymbol{R}_\Gamma^{\text{obs,new}} \\ \boldsymbol{R}_\Gamma^{\text{new,obs}} & \boldsymbol{R}_\Gamma^{\text{new}} \end{pmatrix} \right]. \tag{9}$$

From Equation 8 and Equation 9, we can formulate the conditional distribution of the de-

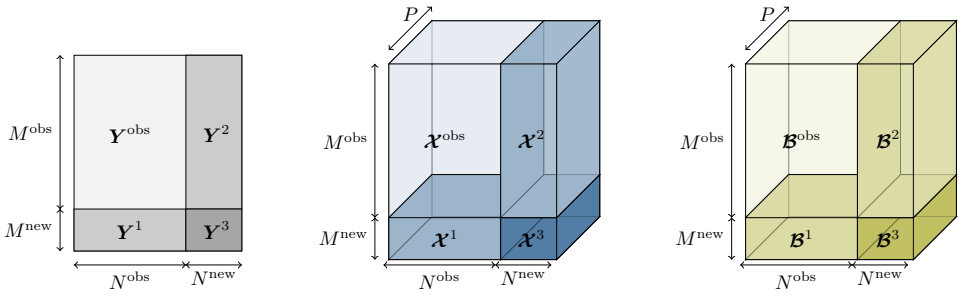


Figure 2: Illustration of BKTR interpolation data.

compositions at new locations and time steps as the following:

$$\begin{aligned}
\boldsymbol{u}_r^{\text{new}} | \boldsymbol{u}_r^{\text{obs}} &\sim \mathcal{N}(\boldsymbol{R}_\Phi^{\text{new,obs}} \boldsymbol{R}_\Phi^{\text{obs}^{-1}} \boldsymbol{u}_r^{\text{obs}},\ \boldsymbol{R}_\Phi^{\text{new}} - \boldsymbol{R}_\Phi^{\text{new,obs}} \boldsymbol{R}_\Phi^{\text{obs}^{-1}} \boldsymbol{R}_\Phi^{\text{obs,new}}),\ r = 1, \ldots, R, \\
\boldsymbol{v}_r^{\text{new}} | \boldsymbol{v}_r^{\text{obs}} &\sim \mathcal{N}(\boldsymbol{R}_\Gamma^{\text{new,obs}} \boldsymbol{R}_\Gamma^{\text{obs}^{-1}} \boldsymbol{v}_r^{\text{obs}},\ \boldsymbol{R}_\Gamma^{\text{new}} - \boldsymbol{R}_\Gamma^{\text{new,obs}} \boldsymbol{R}_\Gamma^{\text{obs}^{-1}} \boldsymbol{R}_\Gamma^{\text{obs,new}}),\ r = 1, \ldots, R.
\end{aligned} \tag{10}$$

From this point, we can estimate the parameters in Equation 10 using estimated values that were captured during the K_2 sampling iterations of the MCMC sampling described in the Algorithm 1. To estimate the distribution parameters of $\boldsymbol{u}_r^{\text{new}}$ and $\boldsymbol{v}_r^{\text{new}}$, we can use the accumulated estimated values of all spatial and temporal kernel hyperparameters Φ and Γ to evaluate, at each sampling, iteration the covariance matrices $\boldsymbol{R}_\Phi^{\text{obs}}$, $\boldsymbol{R}_\Phi^{\text{new}}$, $\boldsymbol{R}_\Phi^{\text{obs,new}}$, $\boldsymbol{R}_\Gamma^{\text{obs}}$, $\boldsymbol{R}_\Gamma^{\text{new}}$ and $\boldsymbol{R}_\Gamma^{\text{obs,new}}$. From the estimated distributions, we are able to sample spatial decomposition values at new locations $\boldsymbol{u}_r^{\text{new}^*}$ and temporal decomposition values at new time points $\boldsymbol{v}_r^{\text{new}^*}$ for each of the K_2 sampling iterations. From $\boldsymbol{u}_r^{\text{new}^*}$, $\boldsymbol{v}_r^{\text{new}^*}$ and the corresponding $\boldsymbol{w}_r$ of each sampling iteration, we are then able to approximate the values of $\boldsymbol{\mathcal{B}}^{\text{new}^*}$. Finally, by combining the K_2 sampled $\boldsymbol{\mathcal{B}}^{\text{new}^*}$ values, we can estimate the distribution of $\boldsymbol{\mathcal{B}}^{\text{new}}$.

After this sampling process, it is fairly simple, following Equation 2, to get the expectation of the response variable $E(\boldsymbol{Y}^{\text{new}} | \boldsymbol{Y}^{\text{obs}})$ utilizing $\boldsymbol{\mathcal{B}}^{\text{new}}$ and the corresponding covariates $\boldsymbol{\mathcal{X}}^{\text{new}}$.

An important aspect of using kriging for predictions is that it is mainly effective when new locations and time points are close to each other. Thus, BKTR predictions made in a relatively close spatial and temporal neighborhoods should be very effective. However, in tasks like predicting temporal values located very far in the future, the current prediction methodology might yield poor results due to the fact that kriging relies on nearby results to make predictions.

This interpolation sampling process is implemented in both **BKTR** packages. The performance and results of this implementation are reviewed extensively on simulated data in Section 5.5 and on real world data in Section 6.3.

3. BKTR regressor class

The **BKTR** packages provide a `BKTRRegressor` class that encapsulates the core concepts and functionalities of the BKTR algorithm. This class provides a simple interface to fit the BKTR model for a given data set and allows to visualize fitted coefficients as well as to predict values for new or missing observations.

The `BKTRRegressor` class is implemented in the `bktr` module of the **BKTR** packages. Even though the `BKTRRegressor` class has been designed in a user friendly manner, its flexibility and number of parameters need to be carefully considered. Thus, this section is dedicated to explain the different data inputs, parameter inputs and all the attributes and methods of the `BKTRRegressor` class.

3.1. Input data

For the BKTR algorithm, three data frame inputs need to be provided to a `BKTRRegressor` object initialization.

- A data frame `spatial_positions_df` containing information about the position of each spatial location with M rows and $1 + p_s$ columns. The first column is used to label each spatial location and the other p_s columns encapsulate the spatial position of each location in p_s dimensions. When we consider for example a location to be represented by longitude and latitude, we would have $p_s = 2$ and a 3 column data frame. For consistency, the first column containing location labels should contain only unique values and needs to be named `location`.
- A data frame `temporal_positions_df` containing information about each time point's timestamp with N rows and $1 + p_t$ columns. The first column labels each time point and the subsequent p_t columns are used to capture the timestamps temporal position. Typically, like when using dates as time points, $p_t = 1$ can be used resulting in a 2 columns dataframe. For consistency, the first column containing time point labels should contain only unique values and needs to be named `time`.
- A principal data frame `data_df` with MN rows and $3 + P$ columns, containing a location label column, a time point label columnn, a column containing the response variable and P columns for the covariates. The first column of the dataframe needs to be named `location`, it must contain the same values as the `spatial_positions_df`'s `location` column and each value must appear a N times. Similarly, the second column of the dataframe needs to be named `time`, it must contain the same values as the `temporal_positions_df`'s `time` column and each value must appear M times. In other words, the `data_df` dataframe must contain in the `location` and `time` columns all the possible combinations of `spatial_positions_df` locations and `temporal_positions_df` time points. In general, it is preferred but not mandatory that the third column of the dataframe contains the response variable data.

It is important to note that `data_df` can also contain missing values in the response variable y column. Those missing values must be properly encoded as `NaN` and represent, in fact, a flattened version of the matrix Ω in Equation 4.

To give us a better idea of what would be a valid shape for `BKTRRegressor` input data, let's take a look at the BIXI data presented in Section 6. To keep the visualization succinct, we will take a subset of two locations ($M = 2$), three time points ($N = 3$) and two covariates ($P = 2$). We can start by looking at a valid `spatial_positions_df`:

```
R> bixi_data <- BixiData$new()
R> ex_locs <- c('7114 - Smith / Peel', '6435 - Victoria Hall')
```

```
R> ex_times <- c('2019-04-17', '2019-04-18', '2019-04-19')
R> print(bixi_data\$spatial_positions_df[location %in% ex_locs])
```

```
               location latitude longitude
1: 6435 - Victoria Hall 45.48129 -73.60033
2:  7114 - Smith / Peel 45.49284 -73.55642
```

Then look at a valid `temporal_positions_df`:

```
R> print(bixi_data$temporal_positions_df[time %in% ex_times])
```

```
         time time_index
1: 2019-04-17          2
2: 2019-04-18          3
```

And finally look at the corresponding valid `data_df`:

```
R> print(bixi_data$data_df[
R>   location %in% ex_locs & time %in% ex_times,
R>   c(1:4, 17)
R> ])
```

```
               location       time nb_departure area_park  humidity
1: 6435 - Victoria Hall 2019-04-17    0.2178218 0.2254071 0.1919343
2: 6435 - Victoria Hall 2019-04-18    0.3366337 0.2254071 0.4697535
3: 6435 - Victoria Hall 2019-04-19    0.2178218 0.2254071 0.9350261
4:  7114 - Smith / Peel 2019-04-17    0.7821782 0.0652808 0.1919343
5:  7114 - Smith / Peel 2019-04-18    0.3861386 0.0652808 0.4697535
6:  7114 - Smith / Peel 2019-04-19    0.6534653 0.0652808 0.9350261
```

Another important data related input for the `BKTRRegressor` is the `formula`. By default, if no `formula` is provided, the third column of `data_df` will be used as the response variable $\boldsymbol{y}$ and the remaining columns will be used as covariates $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$. If a `formula` is provided, the model matrices $\boldsymbol{y}$ and $\boldsymbol{X}$ will be extracted from the `data_df` and the `formula`. The `formula` must be in form of `y ~ x1 + x2 + .  .. + xP` and correspond to a valid Wilkinson formula (Wilkinson and Rogers 1973). All the terms in the `formula` must correspond to valid column names of the `data_df` data frame. For R users, the `formula` must be a `formula` object. For Python users, the `formula` must be a string that can be parsed by the **Formulaic** library. In both cases, it is important to note that by default, an intercept term is automatically added to the model matrix $\boldsymbol{X}$ and can be removed by adding a `-1` term to the `formula`.

The covariates $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$ can describe the spatial and temporal attributes of the observations. By nature, the **BKTR** packages are designed to be able to take into accounts spatial attributes that vary through time (e.g. population density that varies through time) or temporal attributes that vary through space (e.g. temperature that varies through different countries). However, it is not uncommon to have spatial attributes that are constant through time and temporal attributes that are constant through space (like in the BIXI example,

see Section 6). When this is the case and the data is provided in a compressed manner, we provide a `reshape_covariate_dfs` utility function (see Appendix B) that can help users to obtain a valid `data_df`.

3.2. Input parameters

On top of dataframes, the user must provide sampling parameters to be able to initialize a regressor. Those parameters include `burn_in_iter` and `sampling_iter` which are integer inputs representing the number of iterations for the burn-in phase (K_1) and the sampling phase (K_2) respectively. There is also `rank_decomp` which is an integer input representing the rank of the CP decomposition (R). The default values for iterations are 500 iterations for each K_1 and K_2 whereas the default value for the rank decomposition is 10. Two different `Kernel` objects can also be provided (which are further described in Section 4.2), one for the spatial kernel `spatial_kernel` and one for the temporal kernel `temporal_kernel`. The user can also provide some distribution parameters like `sigma_r` which is a float representing the variance of the white noise process (τ^{-1}). We can supply `a_0` and `b_0` representing the initial values for the shape (α) and rate (β) of the gamma function generating τ. Since `sigma_r`, `a_0` and `b_0` are not always trivial to select, we provide sensible default values for those parameters of `sigma_r=1E-2`, `a_0=1E-6` and `b_0=1E-6` respectively. The kernels `spatial_kernel` and `temporal_kernel` are provided default `Kernel` objects as well. The default used spatial kernel is a Matérn kernel 5/2 while the default temporal kernel is a Squared Exponential (SE) kernel. The `BKTRRegressor` class can also take a boolean value input for the `has_geo_coords` parameter. This parameter, that has a default truthful value, indicates if we need to apply or not a Mercator projection (Snyder 1987) to the spatial locations. We usually need to apply a projection when the spatial positions are encoded in longitude and latitude so that we keep the kernel valid. The `geo_coords_scale` parameter is directly associated to the Mercator projection and dictates the scale in which it should be projected. More information about the projection are available in the Appendix C.

When using all the defaults parameters mentioned above, it is very simple to run a BKTR regression on a given dataset. Here is a very simple example of the usage of the `BKTRRegressor` on the full version of the BIXI data mentioned in Section 3.1:

```
R> bktr_regressor <- BKTRRegressor$new(
+    data_df=bixi_data$data_df,
+    spatial_positions_df=bixi_data$spatial_positions_df,
+    temporal_positions_df=bixi_data$temporal_positions_df)
```

Once the regressor is initialized, the user can launch the MCMC sampling process by calling the `mcmc_sampling` method. This method will launch the MCMC sampling process and store the results inside the `BKTRRegressor` object. Depending on the number of iterations and the size of the data, this process can take a long time to complete. Thus, we added to the sampling process a progress log showing the current iteration number, the elapsed time and the current error values. The displayed in-sample errors are the mean absolute error (MAE) and the root mean square error (RMSE) of the response variable. The following code chunk shows the usage of the `mcmc_sampling` method, but with a truncated output for the sake of brevity.

```
R>  bktr_regressor$mcmc_sampling()
```

`BKTRRegressor` Methods	
`mcmc_sampling`	Launch the MCMC sampling process for a predefined number of iterations and a given set of parameters
`predict`	Allows to estimate the response variable $\boldsymbol{y}$ and the $\boldsymbol{\mathcal{B}}$ coefficients for new locations or time points
`get_beta_summary_df`	Get a summary of estimated beta values (mean, stdev, quantiles). Labels can be provided for spatial locations, time points and features. When no labels are given for a dimension, all its betas are shown.
`BKTRRegressor` Attributes	
`summary`	Gives a summary of the MCMC regressor object
`beta_covariates_summary_df`	Gives a dataframe summarizing beta covariates per feature (mean, stdev, quantiles)
`y_estimates`	Gives a data frame for the estimated target variable
`imputed_y_estimates`	Gives a data frame of the estimated and imputed target variable including missing data (Ω)
`beta_estimates`	Gives a data frame of beta estimated values
`hyperparameters_per_iter_df`	Gives a data frame of estimated MCMC hyperparameters (Kernels' and τ) values per iteration

Table 1: `BKTRRegressor` attributes and methods

```
* Iter 1     | Elapsed    0.48s | MAE  0.1036 | RMSE  0.1465 *
* Iter 2     | Elapsed    0.32s | MAE  0.0644 | RMSE  0.0870 *
...
* Iter 1500  | Elapsed    2.83s | MAE  0.0543 | RMSE  0.0734 *
* Iter TOTAL | Elapsed 4528.07s | MAE  0.0525 | RMSE  0.0714 *
```

3.3. Attributes and visualizations

Once a `BKTRRegressor` object has been initialized, the user can access the attributes and methods shown in Table 1. Note that except for the `mcmc_sampling` method, all the other methods and attributes are available only after the MCMC sampling process is completed. When called on a `BKTRRegressor` object, the built-in `summary` and `print` R functions will simply display the `summary` attribute.

The `predict` method allows to estimate the response variable $\boldsymbol{Y}$ and the $\boldsymbol{\mathcal{B}}$ coefficients for new locations and/or time points which we referred as interpolation in Section 2.3. The method takes as input a data frame containing the covariates for the new locations and/or time points `new_data_df`, a data frame containing the spatial positions of the new locations `new_spatial_positions_df` and finally a data frame containing the temporal positions of the new time points `new_temporal_positions_df`. It is possible to provide either a `new_spatial_positions_df` or a `new_temporal_positions_df` or both. In R, it can also be called via the `predict` built-in function using the `BKTRRegressor` object as first argument followed by all the other arguments in the same order as described above. Extensive usage examples of the `predict` method are provided in Section 5.5 and Section 6.3.

The `get_beta_summary_df` method allows to get a summary of the estimated $\boldsymbol{\mathcal{B}}$ coefficients.

`BKTRRegressor` Plot Functions	
`plot_temporal_betas`	Plot beta values through time for a given spatial point and a set of feature labels.
`plot_spatial_betas`	Plot beta values through space for a given time point and a set of feature labels.
`plot_covariates_beta_dists`	Plot the distribution of beta estimates regrouped by features. A subset of features can be provided to plot only a subset of them.
`plot_hyperparams_dists`	Plot the distribution of τ and kernels' hyperparameters for all sampling iterations. `hyperparameters` allows to plot only a subset of parameters.
`plot_hyperparams_per_iter`	Plot the values of τ and kernels' hyperparameters through sampling iterations (trace plot). The argument `hyperparameters` allows to plot only a subset of parameters.

Table 2: Plot functions that can be used on a `BKTRRegressor` object after MCMC sampling

It can take as inputs a list of labels for the spatial locations, a list of time point labels and a list of feature labels. When no labels are given for one of the input, all the $\boldsymbol{\mathcal{B}}$ coefficients for the corresponding dimension are shown. The method returns a data frame containing the mean, standard deviation, quantiles for each of the queried $\boldsymbol{\mathcal{B}}$ coefficient for the sampled posterior distribution. A usage example of the `get_beta_summary_df` method is provided in Section 6.1.

Multiple visualization functions are also available in the **BKTR** packages. Those visualization functions are shown in Table 2. All functions take a `BKTRRegressor` object as a first input for which the sampling process must be completed. We can also specify, via the `show_figure` boolean parameter, to either return a plot object (ggplot2 in R and plotly in Python) or to just display the plot. The plot object can then be used to customize the plot further if needed or to save it as an image. Also, all visualization functions can use the `width` or the `height` parameters to customize the size of the plotting area.

Most of the attributes, methods and visualization functions described in this section are further explored and tested in the Section 5 and Section 6.

4. BKTR Kernels

Kernels play a crucial role in Gaussian processes. They are used to model the similarity or dissimilarity between observations. This similarity can then serve to generate a covariance matrix that can be used as a prior distribution for given parameters. The choice of a kernel is very important as it influences properties of the gaussian process and makes hypotheses about the underlying structure of the function being modeled. For example, a squared exponential kernel (SE) assigns a high covariance to input points that are defined as close and low covariance to points that are distant. In contrast, a periodic kernel assigns high covariance to points that are separated by a multiple of a given period, resulting in a GP that can capture periodic patterns in the data (Duvenaud 2014). Thus, the pattern difference between the SE

and periodic kernel highlights the importance of choosing a sensible kernel when modeling a function with a GP.

In BKTR, we use kernels to model relationships between spatial locations and between time points. A `BKTRRegressor` object uses two distinct kernels which are the `spatial_kernel` and the `temporal_kernel`. The `spatial_kernel` is used to generate the spatial covariance matrix $\boldsymbol{K}_s$ and the `temporal_kernel` is used to generate the temporal covariance matrix $\boldsymbol{K}_t$ that are defined in Equation 5. The `spatial_kernel` and `temporal_kernel` objects can come from any kernel class implemented in the `kernels` module of BKTR (see Section 4.2). We can say that **BKTR** kernel classes themselves need two main components to be defined: the kernel parameters (see Section 4.1), and the kernel function (explained in Section 4.2). The kernels implemented in BKTR take inspiration from the kernels existing in the **GPyTorch** package (Gardner *et al.* 2018) and the **Scikit-learn** package (Pedregosa *et al.* 2011).

4.1. Kernel Parameters

All kernels implemented in BKTR have a set of parameters `KernelParameter` that can be optimized during the training process of the MCMC sampling. This set of parameters can be accessed via the `parameters` attribute of a kernel after its initialization. Kernels in general contain sensible default parameters. However, the user can change the behavior of a given `Kernel`'s parameter by providing a `KernelParameter` object to the related parameter argument.

The `KernelParameter` class can take multiple arguments at initialization:

- `value`: an initial value used for the parameter during slice sampling or the constant value used for the parameter when it is not optimized
- `is_fixed`: a boolean argument indicating if the hyperparameter value is fixed or optimized during sampling
- `lower_bound`: a value that indicates what is the minimum value that can be taken by the parameter during sampling
- `upper_bound`: a value that indicates what is the maximum value that can be taken by the parameter during sampling
- `slice_sampling_scale`: a value indicating the scale parameter for the slice sampling algorithm
- `hparam_precision`: a value indicating the precision of the hyperparameter

The `value` argument is the only required argument to initialize `KernelParameter`. The other arguments have default values if not provided. The `is_fixed` argument is set to false by default, which means that the parameter will necessarily be optimized during the training process. The `lower_bound` and `upper_bound` arguments are set to `1E-3` and `1E3` respectively by default, which means that the parameter will be sampled in a range of $[10^{-3}, 10^{3}]$ during the training process. The `slice_sampling_scale` argument is set to `2` by default, which represent the slice sampling step size during the MCMC sampling process. Finally, the `hparam_precision` argument is set to `1E-3` by default.

Kernel Class	Optimizable Parameters	Kernel Function
`KernelWhiteNoise` White noise		$k(x, x') = I_{\|x\|}$, where $\|x\|$ is the the number of elements in the position vector x.
`KernelSE` Squared exponential	`lengthscale` (ℓ)	$k(x, x'; \ell) = \exp\left(-\frac{d(x, x')^2}{2\ell^2}\right)$
`KernelRQ` Rational quadratic	`alpha` (α) `lengthscale` (ℓ)	$k(x, x'; \alpha, \ell) = \left(1 + \frac{d(x, x')^2}{2\alpha\ell^2}\right)^{-\alpha}$
`KernelPeriodic` Periodic	`lengthscale` (ℓ) `period_length` (t)	$k(x, x'; \ell, t) = \exp\left(-\frac{2\sin^2(\frac{\pi d(x,x')}{t})}{\ell^2}\right)$
`KernelMatern` Matérn	`lengthscale` (ℓ)	$k(x, x'; \ell, \nu) =$ $\begin{cases}\exp(-\frac{D}{\ell}), & \text{if } \nu = 1\\ (1 + \frac{\sqrt{3}D}{\ell})\exp(-\frac{D}{\ell}), & \text{if } \nu = 3,\\ (1 + \frac{\sqrt{5}D}{\ell} + \frac{5D^2}{3\ell^2})\exp(-\frac{D}{\ell}), & \text{if } \nu = 5\end{cases}$ where $D = d(x, x')$ and ν is a Matérn kernel input called the `smoothness_factor`. The `smoothness_factor` input can be either `1`, `3` or `5`, which correspond to so called Matérn $\frac{1}{2}$, $\frac{3}{2}$ and $\frac{5}{2}$ kernels respectively.
`KernelAddComposed` Composed via addition		$k(x, x'; \Lambda_a, \Lambda_b) = k_a(x, x'; \Lambda_a) + k_b(x, x'; \Lambda_b)$, where k_a and k_b are two kernel functions and Λ_a and Λ_b their sets of parameters.
`KernelMulComposed` Composed via multiplication		$k(x, x'; \Lambda_a, \Lambda_b) = k_a(x, x'; \Lambda_a) * k_b(x, x'; \Lambda_b)$, where k_a and k_b are two kernel functions and Λ_a and Λ_b their sets of parameters.

Table 3: List of kernel classes implemented in the **BKTR** packages and their respective parameters and equations. The $d(x, x')$ function is the Euclidean distance function applied on the observations' positions. Note that only the parameters that are objects coming from the `KernelParameter` class (parameters that can be sampled) are listed in the Optimizable Parameters column.

4.2. Kernel Classes

BKTR `Kernel` classes are core components encapsulating behavior and attributes used to compute the covariance matrix during Gaussian processes. They contain information about observations' positions, all the kernel's hyperparameters and a kernel function.

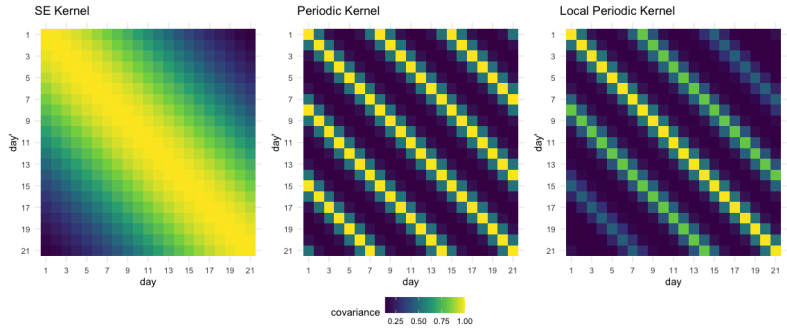


Figure 3: Heatmap plots of the covariance matrix for three different kernels implemented in the **BKTR** package calculated on 21 consecutive days. Presented kernels are, in order, a `KernelSE` with a `lengthscale` of 10, a `KernelPeriodic` with a `period_length` of 7 and a local periodic kernel (`KernelMulComposed`) resulting from the multiplication of two other kernels. The local periodic kernel plot shows that it contains the periodicity of the `KernelPeriodic` kernel and the decay of the `KernelSE` kernel.

For a kernel to induce a covariance matrix, it needs an initial input encoding the positions of the observations. This position vector is set for a `Kernel` object using the `set_positions_df` method. This method accept a dataframe having a number of rows equal to the number of observations and number of column equal to 1 + K, where the first column is for labeling each observation and the other columns contains the location information for the K dimensions of each observation.

When using stationary kernels, we can initially calculate a distance between observations $d(x, x')$ and use it for the generation of the covariance matrix. To keep kernels valid, in **BKTR**, we use the Euclidean distance function:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{K} (x_{ik} - x_{jk})^2}, \tag{11}$$

where x_i and x_j are two given observations' positions from the position vector $\boldsymbol{x}$ and K is the number of dimensions of a given position.

With this position information, we can use all the different kernel classes available in the **BKTR** package, which are enumerated in Table 3. It is possible to observe that different kernels yield completely different covariance matrices according to their function and parameters. For instance, the `KernelSE` kernel yields a covariance matrix with a smooth decreasing function from the diagonal, whereas the `KernelPeriodic` kernel yields a covariance matrix with a periodic pattern from the diagonal. Those different covariance matrices can be visualized using the `plot` method of any implemented `Kernel` class and the two aforementioned kernel examples are shown in Figure 3 (first two sub figures).

Like in the case of the local periodic kernel, sometimes it is useful to combine different kernels to obtain a composition of different covariance matrices. This feature is implemented in the **BKTR** package with the `KernelAddComposed` class when adding two kernels and `KernelMulComposed` class when multiplying two kernels. We can look at an example of a

composed kernel in BKTR by creating a local periodic kernel when multiplying a `KernelSE` kernel with a `KernelPeriodic` kernel. The `KernelSE` kernel is used to model the smooth decreasing function from the diagonal and the `KernelPeriodic` kernel is used to model the periodic function from the diagonal. In the BIXI example used by Lei *et al.* (2023), a local periodic kernel was used to model the BIXI stations' demand with a constant period of 7 days. We can easily create and visualize this local periodic kernel using the `KernelComposed` class as shown in the following code snippet:

```
R> library(data.table)
R> days_df <- data.table(day=1:21, position=1:21)
R> se_lengthscale <- KernelParameter$new(value=10)
R> per_length <- KernelParameter$new(value=7, is_fixed=TRUE)
R> kernel_periodic <- KernelPeriodic$new(period_length=per_length)
R> kernel_se <- KernelSE$new(lengthscale=se_lengthscale)
R> kernel_local_periodic <- kernel_periodic * kernel_se
R> kernel_local_periodic$set_positions(days_df)
R> kernel_local_periodic$kernel_gen()
R> kernel_local_periodic$plot()
```

The resulting kernel plot is illustrated as the rightmost plot of the Figure 3. The `lengthscale` parameter of the SE kernel was set to a value of 10 in this example, and the `period_length` of the periodic kernel was set to 7 to enhance kernel properties visualization.

5. Simulation-based study

To showcase the capabilities of the **BKTR** package, we will first generate simulated datasets with known ground truths and then utilize our software library to estimate the underlying parameters. This aims to demonstrate the package's capabilities and to illustrate the different insights that can be obtained from the results of **BKTR**. Subsequently, we will show the imputation and interpolation abilities of the **BKTR** packages on the aforementioned simulated dataset. The imputation and interpolation results will also be compared with the results of the python package implementation.

5.1. Simulation data

For the simulation, we will use the `simulate_spatiotemporal_data` function implemented in the `utils` module of the **BKTR** packages. This function allows to simulate a spatiotemporal dataset with a known ground truth. Its objective is to simulate four different dataframes: a dataframe of spatiotemporal locations, a dataframe of time points, a dataframe of covariates (including the response variable and an intercept term) and a dataframe of beta coefficients.

This function generate M spatial locations in a d dimension Euclidean space with each dimension being in a range of $[0, S_s]$, where S_s is the scale parameter of the spatial dimensions. It also generates N sequential time points that are in a range of $[0, S_t]$, where S_t is the time scale parameter. Resulting time points therefore have a time resolution of $\frac{S_t}{N-1}$.

The covariates are simulated using an intercept term, p_s spatial covariates and p_t temporal

covariates.

$$\begin{aligned}
\boldsymbol{\mathcal{X}}(:,:,p=1) &= 1 && \text{(intercept)},\\
\boldsymbol{\mathcal{X}}(:,:,p=2,\ldots,1+p_s) &= 1_N^\top \otimes \boldsymbol{x}_s^p, \text{ with } \boldsymbol{x_s^p} \sim \mathcal{N}(\boldsymbol{\mu}_s^p, \boldsymbol{I}_M) && \text{(spatial covariates)},\\
\boldsymbol{\mathcal{X}}(:,:,p=2+p_s,\ldots,P) &= \boldsymbol{x}_t^p \otimes 1_M^\top, \text{ with } \boldsymbol{x_t^p} \sim \mathcal{N}(\boldsymbol{\mu}_t^p, \boldsymbol{I}_N) && \text{(temporal covariates)},
\end{aligned} \tag{12}$$

Note that P is the total number of covariates and is equal to $1 + p_s + p_t$, that $\boldsymbol{\mu}_s^p$ and $\boldsymbol{\mu}_t^p$ are vectors of length M and N respectively which represent the mean of the normal distribution from which the covariates are sampled.

The $\boldsymbol{\beta}$ values are generated following a multivariate normal distribution of the form:

$$\text{vec}(\boldsymbol{\beta}_{(3)}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_t^{\text{sim}} \otimes \boldsymbol{K}_s^{\text{sim}} \otimes \boldsymbol{\Lambda}_w^{-1}), \tag{13}$$

Note that $\boldsymbol{K}_s^{\text{sim}}$ and $\boldsymbol{K}_t^{\text{sim}}$ are the covariance matrices generated by the spatial and temporal kernels, respectively. Both kernels are provided as input parameters to the function `simulate_spatiotemporal_data`.

The response variable $\boldsymbol{Y}$, for its part, is created from the product of the covariates and the $\boldsymbol{\mathcal{B}}$ values to which we added an error term $\epsilon \sim \mathcal{N}(0, S_\epsilon \boldsymbol{I}_{MN})$, where S_ϵ is the scale parameter of the error term and $\boldsymbol{I}_{MN}$ is an identity matrix of size $MN \times MN$.

We can formulate the `simulate_spatiotemporal_data` function input parameters as the following:

- M: `nb_spatial_locations` (number of spatial locations)
- N: `nb_time_points` (number of time points)
- d: `nb_spatial_dimensions` (number of spatial dimensions in the Euclidean space)
- S_s: `spatial_scale` (scale of the spatial dimensions)
- S_t: `time_scale` (scale of the time dimension)
- $\boldsymbol{\mu}_s$: `spatial_covariates_means` (mean vector of the spatial covariates)
- $\boldsymbol{\mu}_t$: `temporal_covariates_means` (mean vector of the temporal covariates)
- $k_s^{\text{sim}}(s_m, s_{m'}; \Phi^{\text{sim}})$: `spatial_kernel` (spatial kernel generating $\boldsymbol{K}_s^{\text{sim}}$)
- $k_t^{\text{sim}}(t_n, t_{n'}; \Gamma^{\text{sim}})$: `temporal_kernel` (temporal kernel generating $\boldsymbol{K}_t^{\text{sim}}$)
- S_ϵ: `noise_scale` (scale parameter for the added noise)

The `simulate_spatiotemporal_data` function returns a list (dictionary in Python) containing 4 data frames. The first data frame, `data_df`, has MN rows representing the cross product of all spatial locations and temporal points, coupled with $1+P$ columns representing the response variable and all the covariates. The second data frame, `spatial_locations_df`, has M rows and d columns representing the simulated spatial locations. The third data frame, `time_points_df`, has N rows and 1 column representing the simulated time points. The last dataframe `beta_df` has MN rows and P columns representing the true $\boldsymbol{\mathcal{B}}$ values. The data generated has spatial covariates that are independent of the spatial locations and temporal

covariates that are independent of the time points. Therefore, we used a process similar to the `reshape_covariate_dfs` utility function (see Appendix B) to create a valid `data_df` data frame.

It is important to note that because of the double Kronecker product used in the Equation 13, the covariance matrix that will be generated will use a sizeable amount of memory since its shape will be $MNP \times MNP$. For this reason, we decided to sample from a matrix normal distribution instead of generating the whole covariance matrix. The used matrix normal distribution is defined as follows:

$$\text{vec}(\boldsymbol{\beta}_{(3)}) \sim \mathcal{MN}(\mathbf{0}_{N \times MP}, \boldsymbol{K}_t^{\text{sim}}, \boldsymbol{K}_s^{\text{sim}} \otimes \boldsymbol{\Lambda}_w^{-1}), \tag{14}$$

From this distribution we can significantly reduce memory usage by sampling a tensor of size $N \times MP$ from MNP independent $\mathcal{N}(0, 1)$ distributions. The generated tensor can then be multiplied by the Cholesky decomposition of $\boldsymbol{K}_t^{\text{sim}}$ and be followed by a matrix multiplication with the Cholesky decomposition of $\boldsymbol{K}_s^{\text{sim}} \otimes \boldsymbol{\Lambda}_w^{-1}$.

For the following part of this section, we will use two different shapes of simulated datasets coming from the `simulate_spatiotemporal_data` function. The former, that we will call the *Smaller* dataset, will hold a $\boldsymbol{\mathcal{B}}$ tensor with 2,400 values having $M = 20$ spatial locations, $N = 30$ time points, 2 spatial covariate with $\boldsymbol{\mu}_s = [0, 2]$ and 1 temporal covariate $\boldsymbol{\mu}_t = [1]$. The other type of simulated dataset, that we will call the *Larger* one, will use a $\boldsymbol{\mathcal{B}}$ with 90,000 values having $M = 100$ spatial locations, $N = 150$ time points, 3 spatial covariates with $\boldsymbol{\mu}_s = [0, 2, 4]$ and 2 temporal covariates $\boldsymbol{\mu}_t = [1, 3]$. Both datasets have a noise scale of $S_\epsilon = 1$, a spatial scale of $S_s = 10$, a temporal scale of $S_t = 10$ and spatial data in $d = 2$ dimensions. Spatial and temporal kernel functions used to obtain covariance matrices will vary on a case by case basis, depending on subsections.

5.2. Simulation analysis

We can start by simulating a completely new dataset using the function presented in the last section `simulate_spatiotemporal_data`. We simulate a dataset of the *Larger* shape using a spatial Matérn 5/2 Kernel with a lengthscale parameter value $\phi_1^{\text{sim}} = 14$ and a temporal SE kernel with a lengthscale parameter $\gamma_1^{\text{sim}} = 5$.

```
R> TSR$set_params(seed = 1)
R> matern_lengthscale <- KernelParameter$new(value = 14)
R> se_lengthscale <- KernelParameter$new(value = 5)
R> spatial_kernel <- KernelMatern$new(
+    lengthscale = matern_lengthscale,
+    smoothness_factor = 5)
R> temporal_kernel <- KernelSE$new(lengthscale = se_lengthscale)
R>
R> # Simulate data
R> simu_data <- simulate_spatiotemporal_data(
+    nb_locations=100,
+    nb_time_points=150,
+    nb_spatial_dimensions=2,
+    spatial_scale=10,
```

```
+    time_scale=10,
+    spatial_covariates_means=c(0, 2, 4),
+    temporal_covariates_means=c(1, 3),
+    spatial_kernel=spatial_kernel,
+    temporal_kernel=temporal_kernel,
+    noise_variance_scale=1)
```

Now that the data is simulated, we can fit a BKTR model to it. We will use new kernels for the spatial and temporal dimensions to be sure that the model is able to recover parameters used to simulate the data.

```
R> bktr_regressor <- BKTRRegressor$new(
+    data_df = simu_data$data_df,
+    spatial_kernel = KernelMatern$new(smoothness_factor = 5),
+    spatial_positions_df = simu_data$spatial_positions_df,
+    temporal_kernel = KernelSE$new(),
+    temporal_positions_df = simu_data$temporal_positions_df,
+    has_geo_coords=FALSE)
R> bktr_regressor$mcmc_sampling()

[1] "Iter 1     | Elapsed Time:     1.60s | MAE:  2.9356 | RMSE:  3.7555"
...
[1] "Iter 1000  | Elapsed Time:     1.84s | MAE:  0.8026 | RMSE:  1.0075"
[1] "Iter TOTAL | Elapsed Time:  1684.75s | MAE:  0.7976 | RMSE:  1.0008"
```

Note that the `mcmc_sampling` method will print the results for each iteration and we truncated the output for brevity.

We can now print the summary of the BKTR model to have a quick overview of its parameters and the quality of the fit.

```
>  summary(bktr_regressor)

======================================================================
                        BKTR Regressor Summary
======================================================================
Formula: y ~ .

Burn-in iterations: 500
Sampling iterations: 500
Rank decomposition: 10
Nb Spatial Locations: 100
Nb Temporal Points: 150
Nb Covariates: 6
======================================================================
In Sample Errors:
  RMSE: 1.001
  MAE: 0.798
```

```
Computation time: 1684.75s.
======================================================================
-- Spatial Kernel --
Matern 5/2 Kernel
Parameter(s):
                      Mean   Median       SD  Low2.5p  Up97.5p
lengthscale         13.608   13.602    0.576   12.515   14.799

-- Temporal Kernel --
SE Kernel
Parameter(s):
                      Mean   Median       SD  Low2.5p  Up97.5p
lengthscale          4.534    4.545    0.237    4.007    4.974
======================================================================
Beta Estimates Summary (Aggregated Per Covariates)

                      Mean   Median       SD
Intercept            2.405    2.622    1.688
s_cov_0              2.450    2.634    1.410
s_cov_1             -4.224   -4.166    2.775
s_cov_2             -1.935   -1.954    0.470
t_cov_0             -2.373   -2.612    1.493
t_cov_1              0.071    0.160    0.799
======================================================================
```

We can see that the model was able to recover the kernel parameters used to simulate the data. The estimated lengthscale ϕ_1 for the spatial kernel is 13.608 while the underlying value was $\phi_1^{\text{sim}} = 14$. The estimated lengthscale γ_1 for the temporal kernel is 4.534 while the true value was $\gamma_1^{\text{sim}} = 5$. The estimated noise variance is $1.001^2 = 1.002$ while the true value was 1. We can also compare the estimated $\boldsymbol{\mathcal{B}}$ coefficients with their simulated values.

```
R> beta_err <- unlist(abs(
+    bktr_regressor$beta_estimates[, -c(1, 2)]
+    - simu_data$beta_df[, -c(1, 2)]))
R> print(sprintf('Beta RMSE: %.4f', sqrt(mean(beta_err^2))))
R> print(sprintf('Beta MAE: %.4f', mean(abs(beta_err))))

[1] "Beta RMSE: 0.1449"
[1] "Beta MAE: 0.0855"
```

It is possible to observe that the model was able to find $\boldsymbol{\mathcal{B}}$ coefficients close to the ones used to simulate the data, with a $\text{MAE}_{\boldsymbol{\mathcal{B}}}$ of 0.0855 and a $\text{RMSE}_{\boldsymbol{\mathcal{B}}}$ of 0.1449.

From here, we can plot, using the `plot_hyperparams_traceplot` function, the estimated hyperparameters values (kernels' and τ) through iterations to see how they evolved during the MCMC sampling. This function will plot a subset of the aforementioned hyperparameters

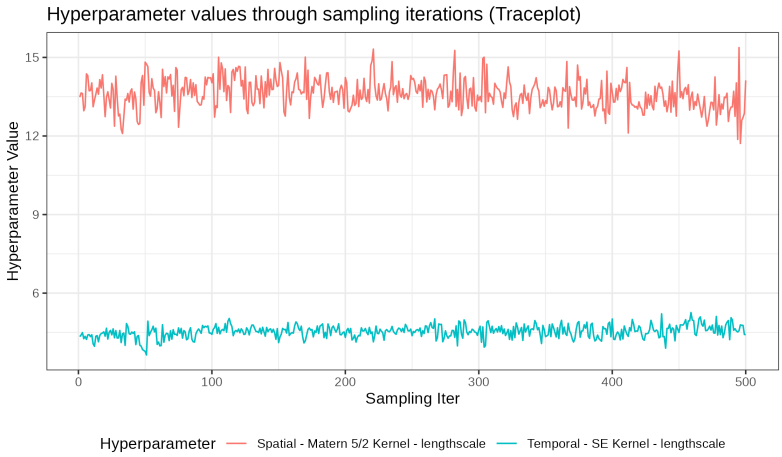


Figure 4: Traceplot of the hyperparameters through sampling iterations for a simulated dataset with 100 spatial locations, 150 temporal points, 6 covariates using a rank decomposition of 10, 500 burn-in iterations and 500 sampling iterations.

if a vector of hyperparameter names is provided as the `hyperparameters` argument. In our case, we find interesting to plot the evolution of all kernel hyperparameters, which we can do using the following code:

```
R> plot_hyperparams_traceplot(bktr_regressor, c(
+     'Spatial - Matern 5/2 Kernel - lengthscale',
+     'Temporal - SE Kernel - lengthscale'))
```

The plot resulting from the `plot_hyperparams_traceplot` function is shown in Figure 4. In this figure, we can observe that during the sampling iterations, the posterior distribution of the hyperparameters was truly hovering around the true values used to simulate the data, which were $\phi_1^{\text{sim}} = 14$ for the spatial kernel's lengthscale and $\gamma_1^{\text{sim}} = 5$ for the temporal kernel's lengthscale. This is a good indication of the strength of the BKTR model, which is able to recover the underlying hyperparameters used to simulate the data. It is also interesting to note that it would have been possible to plot the posterior distribution of the hyperparameters using the `plot_hyperparams_dists` function with the same parameters.

5.3. Influence of device and floating point format

In this section, we look into the influence of the device and floating point format used during tensor operations. The goal of this exercise is to be able to select a default type of floating point format and calculation device when using BKTR.

As some reader might have caught, there is a `TSR` component that we used during all our operations so far. `TSR` is a wrapper containing all used tensor operations functions, and is further detailed in Appendix A. This object allows us to set the seed for our operations via the `seed` argument of the `set_params` method. On top of the `seed` argument, the `set_params`

fp_device	fp_type	Performance Metrics		
		$\text{MAE}_{\boldsymbol{\mathcal{B}}}/\text{RMSE}_{\boldsymbol{\mathcal{B}}}$	$\text{MAE}_{\boldsymbol{Y}}/\text{RMSE}_{\boldsymbol{Y}}$	Time (s)
'cpu'	'float64'	0.078±0.01/0.125±0.02	0.792±0.00/0.992±0.00	713±9
	'float32'	0.080±0.01/0.126±0.02	0.787±0.01/0.987±0.01	525±11
'cuda'	'float64'	0.075±0.01/0.125±0.03	0.793±0.00/0.994±0.01	221±10
	'float32'	0.080±0.02/0.123±0.03	0.787±0.01/0.986±0.01	210±9

Table 4: BKTR regression fitting performance comparison on simulated data using different processing device (`fp_device`) and float point format (`fp_type`).

method can also receive information about where the calculation should be done (on which device) and using which type of floating point format. This information can be passed via the `fp_device` and `fp_type` parameters of the `set_params` method. The available values for `fp_device` at the moment are `'cpu'` and `'cuda'`, where `'cpu'` use the central processing unit of the computer (CPU) and `'cuda'` use the graphic processing unit (GPU) if there is one on the system being used. For the floating point format, there are also two different values that can be passed to the `fp_type` parameter, which are `'float32'` for using single precision number format and `'float64'` to use double precision.

In some software packages or applications, using single precision instead of double precision can degrade the precision of the obtained results, while providing major computational speed upticks. When using a sizeable amount of data, running matrix and tensor operations on GPU instead of CPU can usually provide great computational speed gains. Therefore, we test the four possible combinations of device and floating point format on simulated data to assess their performance. We use, for each combination, 10 new different simulation datasets of the *Larger* size (mentioned in 5.1, on which we fit the **BKTR** package using 500 burn-in iterations, 500 sampling iterations and a rank decomposition of 10. The obtained results are shown in Table 4.

The results show that there is no significant difference in the parameter estimation precision when using different floating point format or device. However, it is interesting to see that the usage of `'float32'` over `'float64'` leads to important computational speed improvement when we compare the mean of sampling runtime, with an improvement of 36% on CPU and a lesser uptick of 5% on GPU. Looking at the influence of the device, it is also possible to perceive that using GPU improves the execution speed for both floating point format (223% using `'float64'`and 138% using `'float32'`). Thus, for every example moving forward, we will use an `fp_type` of `'float32'` and `'cuda'` as a `fp_device`.

5.4. Imputation Results

In this section, we will take a look at the effectiveness of the BKTR packages at doing imputation. By imputation, we mean being able to find the best estimate for response values that were missing in the dataset. For our testing purposes, we simulate again datasets of the *Larger* shape. Then, from those datasets, we remove at random a certain percentage of response variable values and replace them by `NaN`. We use three scenarios of missing values rate which are 10%, 50% and 90%.

In Section 5.2 and Section 5.3 we obtained results that were extremely close to the ground truth. This can be seen through the fact that the $\text{RMSE}_{\boldsymbol{Y}}$ always stayed around 1 which

Simulation Settings	Missing	Lang.	Performance Metrics	
			$\text{MAE}_{\boldsymbol{\mathcal{B}}}/\text{RMSE}_{\boldsymbol{\mathcal{B}}}$	$\text{MAE}_{\boldsymbol{Y}}/\text{RMSE}_{\boldsymbol{Y}}$
$\phi_1^{\text{sim}} = 3$ $\gamma_1^{\text{sim}} = 3$	10%	R	0.76±0.21/1.21±0.31	0.87±0.02/1.10±0.03
		Python	0.83±0.33/1.38±0.66	0.87±0.02/1.10±0.03
	50%	R	0.67±0.14/1.04±0.22	0.93±0.03/1.17±0.0
		Python	0.72±0.11/1.17±0.17	0.91±0.02/1.16±0.04
	90%	R	1.02±0.20/1.50±0.33	2.47±0.39/3.54±0.61
		Python	0.95±0.08/1.43±0.14	2.27±0.26/3.31±0.52
$\phi_1^{\text{sim}} = 6$ $\gamma_1^{\text{sim}} = 6$	10%	R	0.23±0.05/0.38±0.09	0.81±0.02/1.02±0.02
		Python	0.18±0.04/0.28±0.07	0.81±0.01/1.02±0.02
	50%	R	0.20±0.06/0.31±0.11	0.83±0.01/1.05±0.01
		Python	0.18±0.03/0.28±0.07	0.83±0.01/1.04±0.01
	90%	R	0.40±0.07/0.60±0.13	1.24±0.10/1.68±0.19
		Python	0.39±0.06/0.60±0.10	1.23±0.10/1.65±0.17

Table 5: BKTR imputation performance comparison on simulated data. Mean ± standard deviation of the MAE and RMSE for $\boldsymbol{Y}$ and $\boldsymbol{\mathcal{B}}$ computed across 10 distinct simulated datasets for different simulation scenarios.

was the value of the noise scale S_ϵ and that for all floating point format and device used in Section 5.3, we obtained $\text{MAE}_{\boldsymbol{\mathcal{B}}}$ that were below 0.1. One of the factor that could have helped BKTR to pick up the underlying structure of the data so well might have been the fact that spatial and temporal covariance matrices used for the simulation had very high values, making the synthetic data being highly correlated within space and time. Thus, to verify the effect of the kernel parameters on the beta convergence and the imputation method, we will test the imputation implementation on two different lengthscale values scenarios. The first scenario will use lengthscale values of 3 for both kernel, $\phi_1^{\text{sim}} = \gamma_1^{\text{sim}} = 3$ and the second scenario will use a value of $\phi_1^{\text{sim}} = \gamma_1^{\text{sim}} = 6$. Both of those scenarios are coupled with each missing percentage scenario, analyzing a total of 6 different settings. For all different settings, we simulate 10 new datasets, on which we fit $K_1 = 500$ burn-in iterations, $K_2 = 500$ sampling iterations and a rank decomposition $R = 10$.

After the completion of the MCMC sampling, we use the `imputed_y_estimates` attribute to get the imputed values and compare them with equivalent initial values that were removed to get the $\text{MAE}_{\boldsymbol{Y}}$ and $\text{RMSE}_{\boldsymbol{Y}}$. We also look at the impact of missing values on the evaluation of underlying beta values in each scenario by calculating $\text{MAE}_{\boldsymbol{\mathcal{B}}}$ and $\text{RMSE}_{\boldsymbol{\mathcal{B}}}$. We compare the results obtain with the **BKTR** package implemented in R with the results of the **pyBKTR** package implemented in Python. The results of this experiment are shown in Table 5.

We can see that the BKTR packages yield a respectable accuracy on the estimation of missing values. We can also observe, that estimation of missing $\boldsymbol{Y}$ values seems much more accurate when using kernel parameters creating matrices with higher correlations. Moreover, it is possible to notice that reaching 90% of missing values, in all scenarios, is related to an important increase in $\text{MAE}_{\boldsymbol{Y}}$ and $\text{RMSE}_{\boldsymbol{Y}}$. Nonetheless, it is quite impressive to see that the $\text{RMSE}_{\boldsymbol{Y}}$ still has an average value of 1.05 even when there is 50% of $\boldsymbol{Y}$ values missing for lengthscale values of 6. The results in R and Python are very similar, showing us that the calculation implementations correspond well in both languages.

Dataset	Simulation Settings	Lang.	Performance Metrics	
			$\text{MAE}_{\boldsymbol{\mathcal{B}}}/\text{RMSE}_{\boldsymbol{\mathcal{B}}}$	$\text{MAE}_{\boldsymbol{Y}}/\text{RMSE}_{\boldsymbol{Y}}$
Smaller $M=20$ $N=30$ $P=4$	$\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=3$	R	0.81±0.19/1.08±0.27	1.79±0.69/2.48±1.03
		Python	0.76±0.19/1.03±0.27	1.68±0.50/2.35±0.97
	$\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=6$	R	0.51±0.11/0.71±0.19	1.16±0.12/1.49±0.16
		Python	0.52±0.10/0.71±0.19	1.11±0.13/1.42±0.21
Larger $M=100$ $N=150$ $P=6$	$\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=3$	R	1.23±0.73/2.26±1.96	2.18±0.58/3.20±0.93
		Python	1.26±0.64/2.13±1.24	2.60±1.23/3.93±2.17
	$\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=6$	R	0.27±0.08/0.44±0.14	1.00±0.13/1.27±0.17
		Python	0.25±0.08/0.40±0.15	0.96±0.18/1.23±0.29

Table 6: BKTR interpolation performance comparison on simulated data. Mean ± standard deviation of the MAE and RMSE for $\boldsymbol{Y}$ and $\boldsymbol{\mathcal{B}}$ computed across 10 distinct simulated datasets for different simulation scenarios.

5.5. Interpolation Results

For interpolation, we start by exploring how to use the `predict` method of the `BKTRRegressor` class to complete interpolation. As mention in section 3.3, the shape of the provided data is crucial and can be non-trivial to visualize. Thus, we take the time to show how to use the `predict` method using a dataset of a *Larger* shape with simulation's kernels values of $\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=6$.

Since interpolation has not been explored previously for BKTR, we test the `predict` method results on the two types of datasets *Smaller* and *Larger*. We also look at the results for both the R and the Python packages. For each dataset, we also use two different values (3 and 6) for both the spatial kernel lengthscale ϕ_1^{sim} and the temporal kernel lengthscale γ_1^{sim}. For each of the 4 aforementioned scenarios, we simulate 10 different datasets, from which we randomly keep aside 4 spatial locations and 6 time points during the training phase of BKTR. We fit the regression on $K_1=500$ burn-in iterations and $K_2=500$ sampling iterations with a rank decomposition of 10. Then, we use the predict method to try to find the $\boldsymbol{Y}$ and $\boldsymbol{\mathcal{B}}$ values of the kept aside locations. Subsequently, we calculate the error between the initial data and the kept aside data via the $\text{MAE}_{\boldsymbol{\mathcal{B}}}$, $\text{RMSE}_{\boldsymbol{\mathcal{B}}}$, $\text{MAE}_{\boldsymbol{Y}}$, $\text{RMSE}_{\boldsymbol{Y}}$ that we show in Table 6. From this table, we can observe again that using kernels that have higher correlation $\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=6$ give results with a lower error in all scenario. The best results were obtained from the *Larger* dataset and the highest lengthscale values, giving a $\text{RMSE}_{\boldsymbol{Y}}$ of 1.27 and 1.23 for R and Python respectively.

A last exercise that we wanted to achieve on simulated data was to look at the division of the errors through the different segments of the predictions that were presented in Section 2.3. Therefore, we wanted to estimate the $\text{RMSE}_{\boldsymbol{Y}}$ and $\text{MAE}_{\boldsymbol{Y}}$ errors on $\boldsymbol{Y}^{\text{new}}$ distributed across $\boldsymbol{Y}^1, \boldsymbol{Y}^2, \boldsymbol{Y}^3$, as well as the $\text{RMSE}_{\boldsymbol{\mathcal{B}}}$ $\text{MAE}_{\boldsymbol{\mathcal{B}}}$ errors on $\boldsymbol{\mathcal{B}}^{\text{new}}$ across $\boldsymbol{\mathcal{B}}^1, \boldsymbol{\mathcal{B}}^2, \boldsymbol{\mathcal{B}}^3$. We did so with a scenario that used a simulation dataset of the *Larger* shape and simulation kernel lengthscale values of $\phi_1^{\text{sim}}=\gamma_1^{\text{sim}}=6$. We kept aside 10 locations and 20 time points during regression for them to be predicted afterward. Which translated to having $M^{\text{obs}}=90$ observed locations and $M^{\text{new}}=10$ new locations. For the time points, the equivalent sizes were $N^{\text{obs}}=130$ and $N^{\text{new}}=20$. We fitted the BKTR regression on the synthetic data using $K_1=1000$, $K_2=500$ and $R=10$. We repeated the simulation, regression and interpolation exercise 10 times and

Interpolated Portion	Performance Metrics	
	$\text{MAE}_{\boldsymbol{\mathcal{B}}}/\text{RMSE}_{\boldsymbol{\mathcal{B}}}$	$\text{MAE}_{\boldsymbol{Y}}/\text{RMSE}_{\boldsymbol{Y}}$
$\boldsymbol{Y}^1$ and $\boldsymbol{\mathcal{B}}^1$; $M_{\text{new}} = 10 \times N_{\text{obs}} = 130$	0.24±0.06/0.38±0.10	1.05±0.14/1.34±0.22
$\boldsymbol{Y}^2$ and $\boldsymbol{\mathcal{B}}^2$; $M_{\text{obs}} = 90 \times N_{\text{new}} = 20$	0.22±0.03/0.35±0.06	0.81±0.01/1.01±0.01
$\boldsymbol{Y}^3$ and $\boldsymbol{\mathcal{B}}^3$; $M_{\text{new}} = 10 \times N_{\text{new}} = 20$	0.24±0.06/0.37±0.10	1.02±0.14/1.31±0.20
Total $\boldsymbol{Y}^{\text{new}}$ and $\boldsymbol{\mathcal{B}}^{\text{new}}$	0.23±0.04/0.37±0.07	0.92±0.07/1.18±0.11

Table 7: BKTR interpolation performance breakdown on the different portions of the predicted data. Mean ± standard deviation of the MAE and RMSE for $\boldsymbol{Y}$ and $\boldsymbol{\mathcal{B}}$ computed across 10 distinct simulated datasets for different interpolation portions.

the results are shown in Table 7.

From this table, we can observe, via all error metrics, that the interpolation struggles more doing interpolation for new locations ($\boldsymbol{Y}^1$ and $\boldsymbol{\mathcal{B}}^1$) than for new time points ($\boldsymbol{Y}^2$ and $\boldsymbol{\mathcal{B}}^2$). Since, time points span only one dimension, compared to the two dimensions of spatial points, we could think that the covariance pattern between observations his easier to estimate for new time points than new locations. It also seems like the new points situated at new location and new timestamps, $\boldsymbol{Y}^3$ and $\boldsymbol{\mathcal{B}}^3$, have error values very similar to the one of new locations, suggesting that the majority of the errors in this example comes from predicting values for unseen locations.

6. Experimental study

This section aims to test the capacity of the **BKTR** package on real world data. We achieve this task using the same bike sharing demand dataset that was used by Lei *et al.* (2023). A great portion of this dataset was initially gathered by Wang *et al.* (2021) who used information from multiple sources to create a feature rich set of data points.

This dataset contains five distinct data frames

- `bixi_station_departures` (the response variable)
- `bixi_temporal_features` (the temporal covariates)
- `bixi_spatial_features` (the spatial covariates)
- `bixi_spatial_locations` (the location of each spatial point)
- `bixi_temporal_locations` (the location of each temporal point)

The `station_departures` dataframe comes from BIXI Montréal (2023), which is a Montreal, Canada based bike sharing company. It contains the actual response variable that we want to be able to predict, which is the total daily bike departure for each station held by BIXI during its 2019 season. It contains $M = 587$ rows each representing one station and $N = 196$ columns representing different days from April 15 to October 27, 2019. The value of each data point in this dataframe is equivalent to the total number of bike departures for a station on a given day. The `bixi_temporal_features` data includes daily meteorological covariates that vary through time like temperature, precipitation, humidity, etc. This information was

collected from the Environment and Climate Change Canada Historical Climate Data website. Temporal features also include data regarding if each date was a holiday or not, according to the Quebec government. The `station_features` data frame includes data related to the location of each bike sharing stations like surrounding population (taken from the 2016 Canada census data at a dissemination block level), walk score (Walk Score 2023) and a number of other information collected with DMTI Spatial Inc. (2019) concerning point of interests in a surrounding radius of each station like the number of university, the number of metro stations, etc. The dataframe `temporal_locations` is simply representing the position (`time_index`) of each of the $N = 196$ day relative to each other. Since there is no day missing, it simply translates to a range from 0 to 195 associated to the order of each date. The last dataframe, `spatial_locations`, encode the position for each of the $M = 587$ bike station with geographic coordinates (e.g, latitude and longitude).

All dataframes are available, as is, in the **BKTR** library. A normalize version of all datasets can also be accessed through the `BixiData` class of the `examples` module in both the **BKTR** packages. As mentioned previously, when the spatial covariates do not vary through time, the temporal covariates do not vary through space, and they are provided in the form of two different data frames, the **BKTR** packages offer a convenience function allowing to merge those data frames. This function called `reshape_covariate_dfs` can be found in the utils module. It was already used to merge `bixi_station_departures`, `bixi_temporal_features` and `bixi_spatial_features` into the `data_df` dataframe which can be found in the `BixiData` class as well. Note that the dataframes used in this section are the same as the ones used by Lei *et al.* (2023), except for having their column header renamed to obtain a consistent naming convention.

As a result, in this section, we will directly work with `data_df`, `spatial_positions_df` and `temporal_positions_df` from the `BixiData` class. With this dataset, we will try to explore and compare the full capacity of the **BKTR** package on real world data.

6.1. Analysis

In this section we will try to fit a `BKTRRegressor` to the data present in the `BixiData` class, and then we will interpret its results using different properties and visualizations that the **BKTR** package offers.

We will begin by fitting the number of bike departures using three covariates: the mean temperature, the total precipitation in mm and the total park area. To accomplish this task, we will pass the formula `nb_departure ~ 1 + mean_temp_c + area_park + walkscore` to the initialization of the `BKTRRegressor` object. We will then run a MCMC sampling for 1500 iterations, including $K_1 = 1000$ burn-in iterations and $K_2 = 500$ sampling iterations. Additionally, we will utilize a rank decomposition of 8, a spatial Matérn 5/2 kernel and a local periodic temporal kernel like the one presented in Section 4.2. To initialize a `BKTRRegressor` object with the aforementioned parameters, we can simply run the following code chunk (note that we truncated its output for brevity).

```
R> TSR$set_params(seed = 1, fp_type = 'float32', fp_device = 'cuda')
R> bixi_data <- BixiData$new()
R> <- KernelParameter$new(value = 7, is_fixed = TRUE)
R> k_local_periodic <- KernelSE$new() * KernelPeriodic$new(
+    period_length = KernelParameter$new(value = 7, is_fixed = TRUE))
```

```
R> bktr_regressor <- BKTRRegressor$new(
+    formula = nb_departure ~ 1 + mean_temp_c + area_park + total_precip_mm,
+    data_df = bixi_data$data_df,
+    spatial_positions_df = bixi_data$spatial_positions_df,
+    temporal_positions_df = bixi_data$temporal_positions_df,
+    rank = 8,
+    spatial_kernel = KernelMatern$new(smoothness_factor = 5),
+    temporal_kernel = kernel_local_periodic,
+    burn_in_iter = 1000,
+    sampling_iter = 500)
R> bktr_regressor$mcmc_sampling()
```

As soon as the sampling is done, we can access the summary of the `BKTRRegressor` object by calling its `summary` method. This method will print a summary of the model and its parameters as well as the in-sample errors allowing us to evaluate the performance and the shape of the fitted model. The summary of the model is shown in the following code chunk.

```
> summary(bktr_regressor)

======================================================================
                        BKTR Regressor Summary
======================================================================
Formula: nb_departure ~ 1 + mean_temp_c + area_park + total_precip_mm

Burn-in iterations: 1000
Sampling iterations: 500
Rank decomposition: 8
Nb Spatial Locations: 587
Nb Temporal Points: 196
Nb Covariates: 4
======================================================================
In Sample Errors:
  RMSE: 0.072
  MAE: 0.053
Computation time: 1235.50s.
======================================================================
-- Spatial Kernel --
Matern 5/2 Kernel
Parameter(s):
                      Mean   Median       SD  Low2.5p  Up97.5p
lengthscale         21.128   20.877    1.401   18.658   23.738

-- Temporal Kernel --
Composed Kernel (Mul)
  SE Kernel
  Parameter(s):
                        Mean   Median       SD  Low2.5p  Up97.5p
```

```
  lengthscale           6.448    6.437    0.114    6.252    6.685
  *
  Periodic Kernel
  Parameter(s):
                         Mean   Median       SD  Low2.5p  Up97.5p
  lengthscale           0.941    0.942    0.020    0.899    0.979
  period length          Fixed Value: 7.000
======================================================================
Beta Estimates Summary (Aggregated Per Covariates)

                       Mean   Median       SD
Intercept             0.447    0.376    0.306
mean_temp_c          -0.011   -0.008    0.042
area_park            -0.005   -0.011    0.185
total_precip_mm      -0.260   -0.214    0.189
======================================================================
```

The displayed model summary is divided in 4 different sections. The first section shows the parameters used to fit the model, which aligns with the parameters, formula and dataframe provided at the initialization of the `BKTRRegressor` object. The second section display the in-sample errors obtained when fitting of the model, which are the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE) and the total runtime used by the `mcmc_sampling` method. The third section gives information about the parameters of the spatial and temporal kernels used to fit the model. We can see that even the kernel composed via the multiplication of two kernels shows a parameter summary in a very comprehensive fashion. The fourth section shows the summary of the beta coefficients estimates aggregated per covariates. We can see that the `total_precip_mm` has the lowest coefficient estimate, which means that it has, in average, the most negative effect on the number of bike departures. The summary reveals that it took 1235.5 seconds to complete MCMC sampling and fit over 460k coefficients. It is interesting to note that GPU acceleration played a significant role in handling this extensive dataset scenario, as running the same code once on our system with `fp_type='cpu'` took approximately 9.5h hours to complete (34,224.62s).

Another very interesting way to visualize some aspects of the fitted model is to plot the coefficient estimates of covariates through time for a given spatial point. This can be done by calling the `plot_temporal_betas` function on a `BKTRRegressor` object.

```
R> plot_temporal_betas(
+    bktr_regressor,
+    plot_feature_labels = c('mean_temp_c', 'area_park', 'total_precip_mm'),
+    spatial_point_label = '7114 - Smith / Peel')
```

The result of this call is shown in Figure 5. This plot helps us visualize the non stationarity of the coefficients via the evolution of the influence of the covariates through time. For instance, we can see that the influence of the Mean Temperature on the number of bike departures is lower during the beginning and ending months of the season compared to the mid summer months. Also, we can see that, for this station, precipitations have more negative impact on departures during summer.

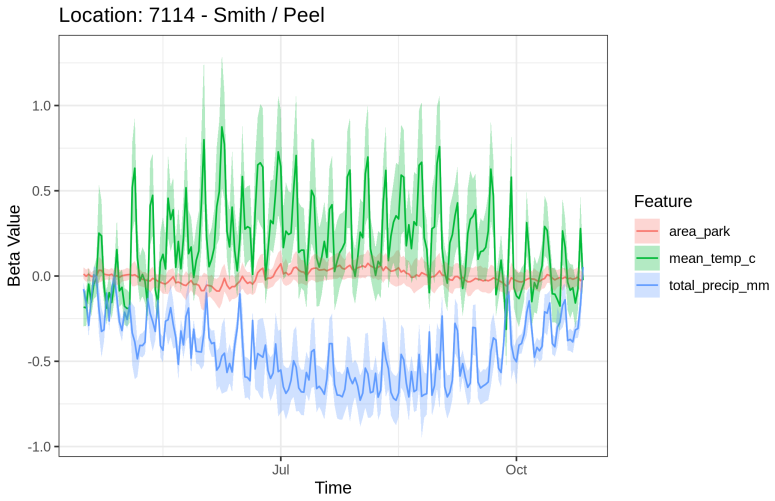


Figure 5: Result of the `plot_temporal_betas` function to plot the coefficient estimates of the covariates through time for a given spatial location.

In the same way, we can plot the coefficient estimates of the covariates through space for a given time point. This can be done similarly by calling the `plot_spatial_betas` function.

```
R> plot_spatial_betas(
+    bktr_regressor,
+    plot_feature_labels = c('mean_temp_c', 'area_park', 'total_precip_mm'),
+    temporal_point_label = '2019-07-19',
+    nb_cols = 3)
```

The result of this function call is shown in Figure 6. Again, this plot helps us to better visualize non stationarity with the evolution of the influence of the covariates through space. We can observe that the precipitation has more negative influence on the number of bike departures in the center of the city when compared to the periphery of the city.

We can see that the **BKTR** package provides a very easy way to fit local regression models on a given dataset. And that it also provides useful and user-friendly integrated functions to help visualize and understand the fitted model.

6.2. Imputation Example

In this section, we will explore how to use BKTR to estimate the values for missing data points in the response variable $\boldsymbol{Y}$ of the BIXI dataset. Imputation for BKTR can only be done at the response variable level, which means that it is not possible at the moment to estimate missing covariate values. It is possible to observe that there is already some missing values in the `data_df` dataframe of the BIXI dataset for the response variable column `nb_departure`, as the next code snippet can illustrate.

```
R> y_is_na <- is.na(bixi_data$data_df$nb_departure)
```

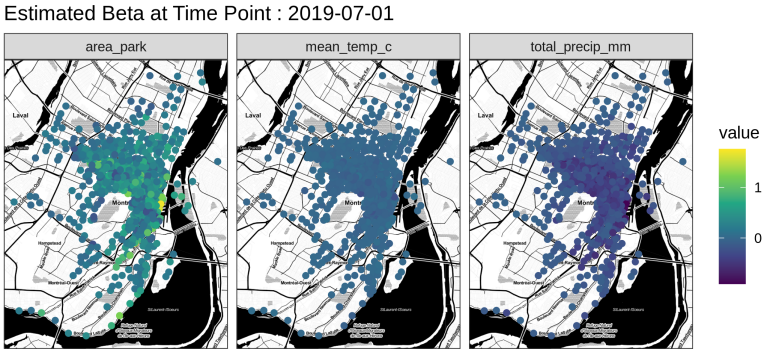


Figure 6: Result of the `plot_spatial_betas` function to plot the coefficient estimates of the covariates through space for a given time point.

```
R> nb_y_na <- sum(y_is_na)
R> sprintf(
+   'There is %.d missing `nb_departure` values representing ~%.2f%%',
+   nb_y_na,
+   nb_y_na / length(y_is_na) * 100))
```

```
"There is 14940 missing `nb_departure` values representing ~12.99%"
```

This shows us that there is already around 13% of the response variable values missing in our dataset. Let's take a look at the first 3 missing values of the dataframe to be able to find at which location and which moment those values were situated.

```
R> bixi_data$data_df[which(y_is_na)[1:3], 1:3]
```

```
                                       location       time nb_departure
1: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-04-22           NA
2: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-05-08           NA
3: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-05-16           NA
```

Knowing that the location *10002 - Métro Charlevoix (Centre / Charlevoix)* is missing multiple values, we can effortlessly use the `imputed_y_estimates` attribute from the fitted regressor instance of Section 6.1 to estimate those data points.

```
R> bktr_regressor$imputed_y_estimates[which(y_is_na)[1:3]]
```

```
                                       location       time     y_est
1: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-04-22 0.8021128
```

```
2: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-05-08 1.0260799
3: 10002 - Métro Charlevoix (Centre / Charlevoix) 2019-05-16 1.0563858
```

After observing the ease of imputing missing response variables in BIXI, we can further validate its efficiency by arbitrarily masking 20% of the non-missing `nb_departures` values. We then fit a `BKTRRegressor` object using the exact same code as in Section 6.1 for the dataset for which we added missing data. Afterwards, we can compare the imputed results `bktr_regressor$imputed_y_estimates` with the original dataset response variable values to validate imputation performance.

```
R> y_is_not_na <- which(!y_is_na)
R> nb_masked_vals <- round(length(y_is_not_na) * 0.2)
R> masked_indices <- sample(y_is_not_na, nb_masked_vals)
R> new_data_df <- bixi_data$data_df
R> new_data_df[masked_indices, 'nb_departure'] <- NA
R> bktr_regressor <- BKTRRegressor$new(new_data_df,
+    ...) # Same parameters as previous section
R> bktr_regressor$mcmc_sampling()
R> y_err <- (bixi_data$data_df$nb_departure[masked_indices]
+    - bktr_regressor$imputed_y_estimates$y_est[masked_indices])
R> sprintf('MAE: %.2f  ||  RMSE: %.2f', mean(abs(y_err)),
+  sqrt(mean(y_err ^ 2)))

MAE: 0.0566  ||  RMSE: 0.0774
```

The imputed data error calculated $\text{RMSE}_{\boldsymbol{Y}} = 0.077$ and $\text{MAE}_{\boldsymbol{Y}} = 0.057$ gives results that are very close to the ones observed in-sample via Section 6.1 summary which was $\text{RMSE}_{\boldsymbol{Y}} = 0.072$ and $\text{MAE}_{\boldsymbol{Y}} = 0.053$. Therefore, we find **BKTR** being a powerful tool for spatiotemporal data imputation.

6.3. Interpolation Example

In the BIXI case, interpolation can prove to be highly valuable in estimating the number of departures at newly planned locations. This estimation could help significantly the allocation planning of number of docks needed at a given new location. Furthermore, when there is complete absence of data for a given time period (e, g. due to a reporting datacenter shortage with a duration of 2 days), we could simply use interpolation to gather an estimate of the number of departure per station during that time period. To test the interpolation capabilities of **BKTR**, we select arbitrarily three bike stations and two contiguous days in the dataset. We then temporarily remove those days and stations from the initial dataset and put them aside. Next, we fit the regressor on the remaining data, and finally we use the `predict` method on the set-aside data with the goal of estimating the number of departures at those days and locations.

We start by setting aside the following three locations *4002 - Graham / Wicksteed*, *7079 - Notre-Dame / Gauvin* and *6236 - Laurier / de Bordeaux*. We also set aside two time points equivalent to the dates *2019-05-01* and *2019-05-02*. We then separate the three main BIXI dataframes in two portions which are the observed data and the new data.

```
R> library(data.table)
R> TSR$set_params(seed=0, fp_type='float32')
R> bixi_data <- BixiData$new()
R> data_df <- bixi_data$data_df
R> spa_df <- bixi_data$spatial_positions_df
R> tem_df <- bixi_data$temporal_positions_df
R> # New locations and times
R> new_s <- c('4002 - Graham / Wicksteed',
+    '7079 - Notre-Dame / Gauvin',
+    '6236 - Laurier / de Bordeaux')
R> new_t <- c('2019-05-01', '2019-05-02')
R> # Cast to IDate to match implicit cast of data.table
R> new_t <- as.IDate(new_t)
R> # Get obs data
R> obs_s <- setdiff(unlist(spa_df$location), new_s)
R> obs_t <- setdiff(unlist(tem_df$time), new_t)
R> obs_data_df <- data_df[data_df[, .I[
+    location %in% obs_s & time %in% obs_t]], ]
R> obs_spa_df <- spa_df[spa_df[, .I[location %in% obs_s]], ]
R> obs_tem_df <- tem_df[tem_df[,.I[time %in% obs_t]], ]
R> # Get new data
R> new_data_df <- data_df[data_df[, .I[
+    location %in% new_s | time %in% new_t]], ]
R> new_spa_df <- spa_df[spa_df[, .I[location %in% new_s]], ]
R> new_tem_df <- tem_df[tem_df[, .I[time %in% new_t]], ]
```

Subsequently, we train the BKTR model on the observed data to be able to do predictions on the new unobserved datasets. As a last step, we compare the interpolation results with the real observed response values when they are not missing with the usual error metrics.

```
R> # Train and predict
R> bktr_regressor <- BKTRRegressor$new(
+    data_df = obs_data_df,
+    spatial_positions_df = obs_spa_df,
+    temporal_positions_df = obs_tem_df,
+    #... other parameters like section 6.1)
R> preds <- bktr_regressor$predict(
+    new_data_df, new_spa_df, new_tem_df)
R> # Sort data for comparison and remove na values
R> new_data_df <- data_df[
+    data_df[, .I[location %in% new_s | time %in% new_t]],
+    c('location', 'time', 'nb_departure')]
R> pred_y_df <- preds$new_y_df
R> setkey(new_data_df, location, time)
R> setkey(pred_y_df, location, time)
R> non_na_indices <- which(!is.na(new_data_df$nb_departure))
R> # Compare predictions
```

```
R> y_err <- (new_data_df$nb_departure[non_na_indices]
+    - pred_y_df$y_est[non_na_indices])
R> sprintf('Predicting %d y values || MAE: %.4f || RMSE: %.4f',
+    length(non_na_indices), mean(abs(y_err)), sqrt(mean(y_err ^ 2)))

[1] "Predicting 1664 y values || MAE: 0.0878 || RMSE: 0.1278"
```

We can perceive that once the data is sorted and split into training and prediction, the other commands related to predictions are fairly easy to use. Also, since the $\boldsymbol{Y}$ values had some missing data points, we needed to remove them from the evaluated vector to calculate the error metrics properly. Nonetheless, we can see that the results of interpolation on BIXI data are convincing. Obtaining a $\text{RMSE}_{\boldsymbol{Y}} = 0.088$ and a $\text{MAE}_{\boldsymbol{Y}} = 0.128$ higher than the in-sample errors, but that are still showcasing decent prediction capabilities on a real set of data.

7. Summary and discussion

The BKTR (Bayesian Kernelized Tensor Regression) packages, presented through this work, introduce a compelling tool for local spatiotemporal regression analysis in both R and Python. This framework offers a user-friendly endpoint, while also focusing on flexibility and computational performance. Leveraging a library specific for tensor computation like Falbel and Luraschi (2023) allows this package to be extremely efficient and to also harness the computing speed of dedicated floating point operation hardware like GPUs. The sensible default values and choices used in **BKTR** allows for a low friction starting point for the majority of users, while enabling complex fine-tuning for the more advanced one through composed kernels and access to multiple regression parameters. This library, implemented in two of the most prominent statistical programming languages, makes the work realized by Lei *et al.* (2023) available to the research community. The **BKTR** packages also bring to life a new feature that is the capacity to do predictions on unobserved time points and locations, called interpolation. We show, through extensive testing, the important capabilities of **BKTR** in regression, imputation and interpolation. Since it is one of the very first package enabling local regressions to fit coefficients for every location and time points combinations of very large dataset, we think that this breakthrough will greatly improve the modelling process of a vast amount of spatiotemporal analysis.

We aim to continuously improve and expand the functionalities of the **BKTR** packages that we hereby presented. We will do so by actively monitoring the user issues and requests that will be provided to the respective GitHub of each package. By having developed the package in both languages, we also commit to continue developing features in a way that will be accessible for R and Python developers. Since both packages are based on **torch** and **pytorch** we will keep a close eye on both packages' advancements to be able to provide to our users the latest enhancements in tensor computation.

Computational details

Results in this paper were obtained using Google Colab instances using a type of shape that was *High-Ram* and a V100 GPU. Thus, the calculations were done with an 8 core Xeon

processor, 25.5GB of total CPU Ram and 16GB of GPU Ram. We also used the default R and Python versions in Colab which were R 4.3.1 with **torch** 0.11.0 and Python 3.10.6 with **pytorch** 1.12.1.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015). "**TensorFlow**: Large-Scale Machine Learning on Heterogeneous Systems." Software available from tensorflow.org, URL https://www.tensorflow.org/.

BIXI Montréal (2023). "Open Data." Accessed: 2023-07-11, URL https://bixi.com/en/open-data.

Chang W (2021). ***R6***: *Encapsulated Classes with Reference Semantics*. R package version 2.5.1, URL https://CRAN.R-project.org/package=R6.

DMTI Spatial Inc (2019). "Enhanced Point of Interest (DMTI)."

Dowle M, Srinivasan A (2023). ***data.table***: *Extension of 'data.frame'*. R package version 1.14.8, URL https://CRAN.R-project.org/package=data.table.

Duvenaud D (2014). *Automatic model construction with Gaussian processes*. Ph.D. thesis, University of Cambridge.

Falbel D, Luraschi J (2023). ***torch***: *Tensors and Neural Networks with 'GPU' Acceleration*. R package version 0.9.1, URL https://CRAN.R-project.org/package=torch.

Gamerman D, Lopes HF, Salazar E (2008). "Spatial dynamic factor analysis." *Bayesian Analysis*, **3**(4), 759 – 792. doi:10.1214/08-BA329. URL https://doi.org/10.1214/08-BA329.

Gardner J, Pleiss G, Weinberger KQ, Bindel D, Wilson AG (2018). "Gpytorch: Blackbox Matrix-Matrix Gaussian Process Inference with Gpu Acceleration." *Advances in neural information processing systems*, **31**.

Gelfand AE, Kim HJ, Sirmans CF, Banerjee S (2003). "Spatial Modeling With Spatially Varying Coefficient Processes." *Journal of the American Statistical Association*, **98**(462), 387–396. doi:10.1198/016214503000170. https://doi.org/10.1198/016214503000170, URL https://doi.org/10.1198/016214503000170.

Geman S, Geman D (1984). "Geman, D.: Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. IEEE Trans. Pattern Anal. Mach. Intell. PAMI-6(6), 721-741." *IEEE Trans. Pattern Anal. Mach. Intell.*, **6**, 721–741. doi:10.1109/TPAMI.1984.4767596.

Kolda TG, Bader BW (2009). "Tensor decompositions and applications." *SIAM review*, **51**(3), 455–500.

Lei M, Labbe A, Sun L (2023). "Scalable Spatiotemporally Varying Coefficient Modelling with Bayesian Kernelized Tensor Regression." 2109.00046.

Lustiger H (2022). *Design Patterns in R*. Tidylab, Auckland, New Zealand. URL https://github.com/tidylab/R6P.

Neal RM (2003). "Slice sampling." *The Annals of Statistics*, **31**(3), 705 – 767. doi:10.1214/aos/1056562461. URL https://doi.org/10.1214/aos/1056562461.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019). "**PyTorch**: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

Plotly Technologies Inc (2015). "Collaborative data science." URL https://plot.ly.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Snyder JP (1987). *Map projections–A working manual*, volume 1395. US Government Printing Office.

Takeuchi K, Kashima H, Ueda N (2017). "Autoregressive Tensor Factorization for Spatio-Temporal Predictions." In *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 1105–1110. doi:10.1109/ICDM.2017.146.

The pandas development team (2022). "**Pandas** 1.4.2." doi:10.5281/zenodo.6408044. URL https://doi.org/10.5281/zenodo.6408044.

Walk Score (2023). "Walk Score Methodology." Accessed: 2023-07-11, URL https://www.walkscore.com/methodology.shtml.

Wang X, Cheng Z, Trépanier M, Sun L (2021). "Modeling bike-sharing demand using a regression model with spatially varying coefficients." *Journal of Transport Geography*, **93**, 103059. ISSN 0966-6923. doi:https://doi.org/10.1016/j.jtrangeo.2021.103059. URL https://www.sciencedirect.com/science/article/pii/S0966692321001125.

Wardrop M (2022). "**Formulaic**: An implementation of Wilkinson formulas." URL https://matthewwardrop.github.io/formulaic.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.

Wilkinson GN, Rogers CE (1973). "Symbolic Description of Factorial Models for Analysis of Variance." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **22**(3), 392–399. ISSN 00359254, 14679876. URL http://www.jstor.org/stable/2346786.

A. Tensor context

In both **BKTR** packages we created modules called `tensor_ops` containing an object named `TSR`. The goal of `TSR` it to be able to centralize all tensor operations and properties in one object. This way, if in the future we needed to fix an operation, or we wanted to integrate another tensor operation backend like **tensorflow** (Abadi *et al.* 2015), we could do it at a central location. Since we wanted to be able to use kernels and the `BKTRRegressor` independently, we decided to set the tensor environment directly with the `TSR` object via the `set_params` method.

One of the few issue that we met with this implementation was that we were not able to use the equivalent of Python's `classmethods` and class properties with the **R6** package. Therefore, we decided to opt for a singleton design pattern to keep only one `TSR` instance with one setting alive at a time inside the R environment. To use this design pattern, we found that the **R6P** package (Lustiger 2022) was a sound implementation of it and thus decided to add it in our dependencies.

B. Covariates reshaping

With spatiotemporal data, it is not rare to observe datasets where the spatial covariates are not varying through time and where the temporal covariates are not varying through space. It is notably the case in the Section 6 for the BIXI data. When such a case arise, we often see that the dataset is expressed in a much more compressed manner. Effectively, when facing this setup, we can reformulate `data_df` into three matrices. The first matrix being the response variable that we call `y_df` which can be of shape $M \times N$, the spatial covariates matrix that we can call `spatial_df` having p_s spatial features and M locations with a shape of $M \times p_s$, and finally the temporal covariates with p_t temporal features and N time points with a shape of $N \times p_t$. Using those three dataframes, we can create a `data_df` containing $MN \times (1 + p_s + p_t)$ and we can achieve that with few operations, repeating the matrices a certain number of times and stacking them together. However, it is possible to achieve that in a simpler manner using the `reshape_covariate_dfs` utility functions that we implemented. This function takes the three above-mentioned dataframes as arguments and another argument called `y_column_name` denoting what is the desired column name for the response variable in the `y_df` dataframe. With all those parameters provided, the `reshape_covariate_dfs` function will return a single dataframe containing the information of the 3 dataframes combined. Let see an example in action of this function using the BIXI dataset.

```
R> spatial_df <- bixi_data$spatial_features_df
R> temporal_df <- bixi_data$temporal_features_df
R> y_df <- bixi_data$departure_df
R> p_s <- ncol(spatial_df) - 1 # Not counting index column
R> p_t <- ncol(temporal_df) - 1
R> sprintf('Response M=%d and N=%d', nrow(y_df), ncol(y_df))
R> sprintf('Spatial features M=%d x p_s=%d', nrow(spatial_df), p_s)
R> sprintf('Temporal features N=%d x p_t=%d', nrow(temporal_df), p_t)
R> data_df <- reshape_covariate_dfs(spatial_df, temporal_df,
+    y_df, 'nb_departure')
R> sprintf('Should obtain MN=%d x P=%d', nrow(spatial_df) *
```

```
+    nrow(temporal_df), 1 + p_s + p_t)
R> sprintf('Reshaped MN=%d x P=%d', nrow(data_df), ncol(data_df) - 2)

[1] "Response M=587 and N=197"
[1] "Spatial features M=587 x p_s=13"
[1] "Temporal features N=196 x p_t=5"
[1] "Should see MN=115052 x P=19"
[1] "Reshaped MN=115052 x P=19"
```

C. Spatial coordinates projection

By default, a vast quantity of geographic tools use longitude and latitude measurements. Since those values represent angles on earth from the meridian and equator, we need to transform them into coordinates that can be projected in a Euclidean space. This is needed to ensure that we keep valid kernels for the MCMC sampling process. One of the simple forms of projection used on a multitude of maps, even today, is called the Mercator projection (Snyder 1987). This type of projection allows to transform longitude and latitude in a 2D space by projecting them on a plane. By doing so, the coordinates become simple x and y coordinates on a map that can be drawn on a paper. This however comes with some limitations. To transform a sphere into a plane, the Mercator projection greatly distorts the size of areas that are far from the equator. To be able to do this cylindrical projection, the projection also needs to have a cutting point, which is usually the meridian. This means that even if two points are close but are located on different side of the meridian, they will appear to be extremely far on a Mercator projections.

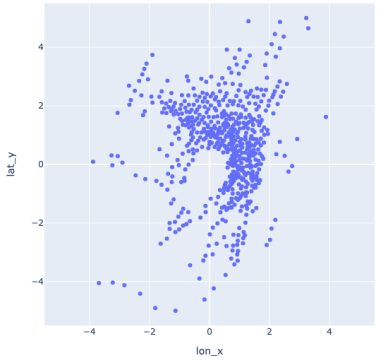
The above-mentioned limitations can have important impacts on some datasets. However, when using coordinates that are very close to each others (e, g. at a city level) like in Section 6, those limitations become negligible. Therefore, we implemented a way to seamlessly transform by default, with a Mercator projection, all longitude and latitude coordinates provided to the `BKTRRegressor` class. When projecting the coordinates in a 2D plan, we allow the user to choose a given scale at which they want the projection to be done. This scale S^p is used to transform all provided coordinates into a square with a domain for the X and Y coordinates of $[-\frac{S^p}{2}, \frac{S^p}{2}]$. Knowing the scale in which the data was projected can help the user choosing sensible values for kernel lengthscale afterward. To help visualize the projection, we compare an example of the Mercator projected data for the BIXI dataset with a plot of the respective geographic coordinates (longitude, latitude) from Plotly Technologies Inc. (2015) using open street map. The comparison is shown is shown in Figure 7 for a scaling parameter of $S^p = 10$.

The figure show that the position property of the points are similar in both cases, but in the BKTR mercator's projection it is now located in on a X and Y axis having ranges of 10.

Affiliation:

Julien Lanthier, Aurélie Labbe
Department of decision sciences
HEC Montréal

(a) Plotly OpenStreetMap scatterbox plot on latitude and longitude



(b) Mercator projection with a scale factor of $S^p = 10$

Figure 7: Visualization comparison between Plotly OpenStreetMap scatter box plot and BKTR Mercator's projection for BIXI spatial locations.

3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec), Canada H3T 2A7
E-mail: julien.lanthier@hec.ca, aurelie.labbe@hec.ca

Mengying Lei, Lijun Sun
Department of civil engineering
McGill University
817 Sherbrooke Street West, Macdonald Engineering Building
Montréal (Québec), Canada H3A 0C3
E-mail: mengying.lei@mail.mcgill.ca, lijun.sun@mcgill.ca

Conclusion

Ce mémoire introduit une nouvelle librairie BKTR, écrite de manière indépendante dans les langages de programmation R et Python qui apporte une innovation significative pour la régression spatiotemporelle. BKTR est, à notre connaissance, la première solution permettant d'utiliser, de manière efficace, une régression spatiotemporelle locale. Cette solution permet d'obtenir des résultats de pointe en des temps records grâce à l'emploi de régression tensorielle et de noyaux permettant de préserver la covariance spatiale et temporelle.

Pour l'instant, la librairie BKTR offre la possibilité de modéliser des variables réponses suivant une fonction normale. Ceci s'applique très bien à un grand nombre de scénarios où les variables réponses peuvent être représentées dans le domaine des réels. Cependant, dans le cas où on voudrait représenter un dénombrement ou une variable binaire, il serait préférable d'utiliser une loi Poisson ou binomiale. La partie reliée à la régression tensorielle de ces distributions a déjà été résolu dans d'autres publications, comme celle de Zhou et collab. (2013). Néanmoins, une méthodologie permettant d'intégrer l'utilisation de processus Gaussiens à BKTR pour les distributions Poisson et Binomiale nécessite davantage de développement. Ceci consiste en un futur projet de recherche que nous prévoyons entreprendre prochainement.

D'autre part, le développement d'un module d'extension de données est aussi un projet que nous trouvons particulièrement intéressant et que nous souhaitons intégrer à BKTR. En utilisant les données de BIXI, nous avons observé qu'il était

fastidieux de mettre à jour les données spatiotemporelles associées aux emplacements (par exemple, longitude et latitude) et aux points temporels (par exemple, la date) étudiés. Si nous voulions obtenir des données pour une nouvelle année, nous devions chercher manuellement les informations météorologiques et les associées au jeu de données. De même, lorsqu'une nouvelle station était utilisée pour une année donnée, il fallait trouver, à travers divers sources, toutes les covariables spatiales reliées à cette station et les rassembler à la main. Pour résoudre ce problème, nous envisageons implémenter un module lier à BKTR permettant de faciliter cette tâche en utilisant des données disponibles en ligne. Ce module permettrait, entre autres, d'associer automatiquement des données météorologiques (par exemple Environnement Canada) ou de jours fériés à des points temporels (dates). Il pourrait également associer des données spatiales à un nouvel emplacement via ses coordonnées géographiques, telles que des données sur les points d'intérêts (e.g. nombre de stations de métro, de restaurants ou d'université à proximité), les infrastructures (e.g. longueur totale de piste cyclable ou de routes à proximité) ou la population (provenant de données gouvernementales comme les recensements). Un avantage considérable d'un tel outil serait de pouvoir aisément mettre à jour les jeux de données utilisés tel que BIXI au fil du temps. Comme les intrants utilisés pour associer les données seraient des coordonnées et des dates, il serait peut-être également possible d'utiliser une certaine partie de cette méthode de collecte sur des données fournies par les utilisateurs. C'est d'ailleurs un aspect qu'on aimerait explorer et développer dans le futur.

Convaincus que BKTR est une solution puissante et novatrice pour l'analyse de données spatiotemporelles, nous nous engageons à la maintenir et à prendre en compte les retours et commentaires de ses futurs utilisateurs. Nous espérons que ce mémoire aura su convaincre son lectorat des progrès apportés par le développement de BKTR et qu'ils pourront à leur tour appliquer cette solution à leur propre jeu de données.

Bibliographie générale

Bakar, K. S. et S. K. Sahu. 2015, «sptimer : Spatio-temporal bayesian modeling using r», *Journal of Statistical Software*, vol. 63, nº 15, doi :10.18637/jss.v063.i15, p. 1–32. URL https://www.jstatsoft.org/index.php/jss/article/view/v063i15.

BIXI Montréal. 2023, «Open data», URL https://bixi.com/en/open-data, accessed : 2023-07-11.

Box, B., G. Jenkins, G. Reinsel et G. Ljung. 2016, *Time Series Analysis : Forecasting and Control*, vol. 68, ISBN 978-1-118-67502-1, doi :10.2307/2284112.

Finley, A. O., S. Banerjee et A. E.Gelfand. 2015, «SPBAYES for large univariate and multivariate point-referenced spatio-temporal data models», *Journal of Statistical Software*, vol. 63, nº 13, p. 1–28. URL https://www.jstatsoft.org/article/view/v063i13.

Gelfand, A. E., H.-J. Kim, C. F. Sirmans et S. Banerjee. 2003, «Spatial modeling with spatially varying coefficient processes», *Journal of the American Statistical Association*, vol. 98, nº 462, doi :10.1198/016214503000170, p. 387–396. URL https://doi.org/10.1198/016214503000170.

Lei, M., A. Labbe et L. Sun. 2023, «Scalable spatiotemporally varying coefficient modelling with bayesian kernelized tensor regression», .

Minasny, B. et A. B. McBratney. 2005, «The matérn function as a general model for soil variograms», *Geoderma*, vol. 128, nº 3, doi :https://doi.org/10.1016/j.geoderma.2005.04.003, p. 192–207, ISSN 0016-7061. URL https://www.sciencedirect.com/science/article/pii/S0016706105000911, pedometrics 2003.

R Core Team. 2022, *R : A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rasmussen, C. et C. Williams. 2006, *Gaussian processes for machine learning*, vol. 2, MIT Press, ISBN 978-0-262-18253-9.

Wang, X., Z. Cheng, M. Trépanier et L. Sun. 2021, «Modeling bike-sharing demand using a regression model with spatially varying coefficients», *Journal of Transport Geography*, vol. 93, doi :https://doi.org/10.1016/j.jtrangeo.2021.103059, p. 103 059, ISSN 0966-6923. URL https://www.sciencedirect.com/science/article/pii/S0966692321001125.

Zeileis, A. et G. Grothendieck. 2005, «zoo : S3 infrastructure for regular and irregular time series», *Journal of Statistical Software*, vol. 14, nº 6, doi :10.18637/jss.v014.i06, p. 1–27.

Zhou, H., L. Li et H. Zhu. 2013, «Tensor regression with applications in neuroimaging data analysis», *Journal of the American Statistical Association*, vol. 108, nº 502, p. 540–552.

Annexe A – Index des Notations

Dans cet ouvrage, nous avons inclus un grand nombre de notations, et pour aider la compréhension des lecteurs, nous avons crû bon d'ajouter une section recueillant et expliquant davantage celles-ci.

Il est à noter que le format et la police de chaque symbole fournissent de l'information par rapport au type d'objet qu'il représente, notamment :

— Les valeurs scalaires n'ont aucun attribut typographique spécifique (ex : x)

— Les vecteurs sont représentés en gras (ex : $\boldsymbol{x}$)

— Les matrices sont représentées en gras et en majuscules (ex : $\boldsymbol{X}$)

— Les tenseurs sont représentés en gras, en majuscules et en police calligraphique (ex : $\boldsymbol{\mathcal{X}}$)

Les tables qui suivent fournissent donc une définition et un contexte pour la grande majorité des notations abordées dans ce mémoire afin de mieux guider les lecteurs.

Symbole	Définition
$\otimes$	Produit de Kronecker
$\odot$	Produit de Khatri Rao
$\circ$	Produit Externe
$\text{vec}(\boldsymbol{X})$	Vectorisation d'une matrice $\boldsymbol{X}$ en empilant chaque colonne de celle-ci
$\boldsymbol{X}_{(k)}$	Dépliement de mode-k d'un tenseur $\boldsymbol{\mathcal{X}}$ vers une forme matricielle $\boldsymbol{X}$
O	Opérateur sélectionnant les valeurs observées d'un vecteur, où $\lvert\Omega\rvert$ représente le nombre de valeurs observées
$\boldsymbol{I}_N$	Matrice identité de taille $N \times N$
$\mathcal{O}$	Notation grand O de Landau décrivant asymptotiquement la complexité d'un algorithme en temps ou en espace

TABLE A1 – Notations générales

Symbole	Définition
$\sim \mathcal{N}(m, s)$	Suit une loi normale de moyenne m et de variance s
$\sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$	Suit une loi normale multivariée de moyenne $\boldsymbol{m}$ et ayant une matrice de covariance $\boldsymbol{\Sigma}$
$\sim \mathcal{MN}_{N\times P}(\boldsymbol{M}, \boldsymbol{U}, \boldsymbol{V})$	Suit une loi matrice normale de moyenne $\boldsymbol{M}$ ayant des matrices d'échelle $\boldsymbol{U}$ $(N \times N)$ et $\boldsymbol{V}$ $(M \times M)$
$\sim \text{Gamma}(a, b)$	Suit une loi gamma avec un paramètre de forme a et un paramètre d'intensité b
$\sim \mathcal{W}(\boldsymbol{U}, v)$	Suit une loi Wishart avec une matrice d'échelle $\boldsymbol{U}$ et un degré de liberté v

TABLE A2 – Notations de distributions

Symbole	Définition
K_1	Nombre d'itérations de rodage
K_2	Nombre d'itérations d'échantillonnage
R	Rang de décomposition
$\boldsymbol{M}$	Nombre d'emplacements étudiés
$\boldsymbol{N}$	Nombre de points temporels étudiés
$\boldsymbol{S}$	Ensemble des emplacements étudiés $s_1, \ldots, s_M$
$\boldsymbol{T}$	Ensemble des points temporels étudiés $t_1, \ldots, t_N$
$\boldsymbol{P}$	Nombre de covariables décrivant une observation donnée. $P = 1 + P_s + P_t$, où P_s et P_t sont le nombre de covariable spatiales et temporelles respectivement
$\boldsymbol{\mathcal{X}}$	Tenseur de covariables de taille $M \times N \times P$
$\boldsymbol{\mathcal{B}}$	Tenseur de coefficients de taille $M \times N \times P$
$\boldsymbol{Y}$	Matrice de variables réponses de taille $M \times N$
$\boldsymbol{U}$	Matrice des facteurs de décomposition spatiale de taille $M \times R$
$\boldsymbol{V}$	Matrice des facteurs de décomposition temporelle de taille $N \times R$
$\boldsymbol{W}$	Matrice des facteurs de décomposition liée aux covariables de taille $P \times R$
$k_s(s_m, s_{m'}; \Phi)$	fonction noyau k_s permettant d'obtenir la matrice de covariance spatiale $\boldsymbol{K}_s$
Φ	ensemble de J paramètres $\phi_1, \ldots, \phi_J$ pour la fonction noyau k_s.
$k_t(t_n, t_{n'}; \Gamma)$	fonction noyau k_t permettant d'obtenir la matrice de covariance temporelle $\boldsymbol{K}_t$
Γ	ensemble de L paramètres $\gamma_1, \ldots, \gamma_L$ pour la fonction noyau k_t.

TABLE A3 – Notations pour BKTR

Symbole	Définition
M^{obs} et M^{new}	Le nombre d'emplacements respectivement précédemment observés et nouveaux. Les nouveaux emplacements étant ceux interpolés.
N^{obs} et N^{new}	Le nombre de points temporels respectivement précédemment observés et nouveaux. Les nouveaux points temporels étant ceux interpolés.
$\boldsymbol{\mathcal{B}}^{\text{new}}$ $(\boldsymbol{\mathcal{B}}^1, \boldsymbol{\mathcal{B}}^2, \boldsymbol{\mathcal{B}}^3)$	Ensemble de tenseurs des coefficients pour les points nouvellement observés. Ses trois composants et leurs dimensions respectives sont $\boldsymbol{\mathcal{B}}^1$ $(M^{\text{new}} \times N^{\text{obs}} \times P)$, $\boldsymbol{\mathcal{B}}^2$ $(M^{\text{obs}} \times N^{\text{new}} \times P)$ et $\boldsymbol{\mathcal{B}}^3$ $(M^{\text{new}} \times N^{\text{new}} \times P)$
$\boldsymbol{\mathcal{X}}^{\text{new}}$ $(\boldsymbol{\mathcal{X}}^1, \boldsymbol{\mathcal{X}}^2, \boldsymbol{\mathcal{X}}^3)$	Ensemble de tenseurs des covariables pour les points nouvellement observés. Ses trois composants et leurs dimensions respectives sont $\boldsymbol{\mathcal{X}}^1$ $(M^{\text{new}} \times N^{\text{obs}} \times P)$, $\boldsymbol{\mathcal{X}}^2$ $(M^{\text{obs}} \times N^{\text{new}} \times P)$ et $\boldsymbol{\mathcal{X}}^3$ $(M^{\text{new}} \times N^{\text{new}} \times P)$
$\boldsymbol{\mathcal{Y}}^{\text{new}}$ $(\boldsymbol{\mathcal{Y}}^1, \boldsymbol{\mathcal{Y}}^2, \boldsymbol{\mathcal{Y}}^3)$	Ensemble de matrices des variables réponses pour les points nouvellement observés. Les trois matrices le composant et leurs dimensions respectives sont $\boldsymbol{\mathcal{Y}}^1$ $(M^{\text{new}} \times N^{\text{obs}})$, $\boldsymbol{\mathcal{Y}}^2$ $(M^{\text{obs}} \times N^{\text{new}})$ et $\boldsymbol{\mathcal{Y}}^3$ $(M^{\text{new}} \times N^{\text{new}})$
$\boldsymbol{R}_\Phi^{\text{obs}}$, $\boldsymbol{R}_\Phi^{\text{new}}$, $\boldsymbol{R}_\Phi^{\text{obs,new}}$	Matrices de covariances spatiales pour un ensemble de paramètres Φ d'une fonction noyau. $\boldsymbol{R}_\Phi^{\text{obs}}$ représente la covariance entre les données observées, $\boldsymbol{R}_\Phi^{\text{new}}$ entre les nouvelles données et $\boldsymbol{R}_\Phi^{\text{obs,new}}$ entre les nouvelles données et celles observées. Notons que $\boldsymbol{R}_\Phi^{\text{new,obs}}$ est simplement $\boldsymbol{R}_\Phi^{\text{obs,new}}$ transposée.
$\boldsymbol{R}_\Gamma^{\text{obs}}$, $\boldsymbol{R}_\Gamma^{\text{new}}$, $\boldsymbol{R}_\Gamma^{\text{obs,new}}$	Matrices de covariances temporelles pour un ensemble de paramètres Γ d'une fonction noyau. $\boldsymbol{R}_\Gamma^{\text{obs}}$ représente la covariance entre les données observées, $\boldsymbol{R}_\Gamma^{\text{new}}$ entre les nouvelles données et $\boldsymbol{R}_\Gamma^{\text{obs,new}}$ entre les nouvelles données et celles observées. Notons que $\boldsymbol{R}_\Gamma^{\text{new,obs}}$ est simplement $\boldsymbol{R}_\Gamma^{\text{obs,new}}$ transposée.
$\boldsymbol{u}_r^{\text{obs}}$ et $\boldsymbol{u}_r^{\text{new}}$	Vecteur de décomposition spatiale de rang r pour les emplacements observés et nouveaux respectivement.
$\boldsymbol{v}_r^{\text{obs}}$ et $\boldsymbol{v}_r^{\text{new}}$	Vecteur de décomposition temporelle de rang r pour les temps observés et nouveaux respectivement.

TABLE A4 – Notations pour l'interpolation

Symbole	Définition
$MAE_{\mathcal{B}}$	$\frac{1}{MNP}\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{p=1}^{P}\lvert\beta_{m,n,p}-\hat{\beta}_{m,n,p}\rvert$
$RMSE_{\mathcal{B}}$	$\sqrt{\frac{1}{MNP}\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{p=1}^{P}(\beta_{m,n,p}-\hat{\beta}_{m,n,p})^2}$
$MAE_{\mathcal{Y}}$	$\frac{1}{MN-\lvert\Omega\rvert}\sum_{i\notin\Omega}\lvert y_i-\hat{y}_i\rvert$
$RMSE_{\mathcal{Y}}$	$\sqrt{\frac{1}{MN-\lvert\Omega\rvert}\sum_{i\notin\Omega}(y_i-\hat{y}_i)^2}$

TABLE A5 – Notations pour les métriques d'erreur

Annexe B – Exemple avec pyBKTR

L'article présenté dans ce mémoire explique en profondeur comment utiliser la bibliothèque BKTR pour le langage de programmation R. Dans le but de captiver également les lecteurs préférant l'utilisation de Python, un exemple détaillé a été ajouté à cet ouvrage afin d'aborder l'utilisation de PYBKTR.

Cette section reprend l'exemple, en partie, d'analyse des données BIXI présenté à la Section 1.6.1. Des parties portant sur la génération de graphiques pour les hyperparamètres et sur les sommaires des coefficients sont également couvertes.

Pour commencer, nous pouvons utiliser la classe `BixiData` qui contient l'ensemble des jeux de données utilisés pour BIXI. Pour utiliser BKTR, nous optons pour un noyau périodique local ayant une période fixe de longueur 7, représentant l'effet cyclique hebdomadaire. Nous utilisons également un noyau spatial Matérn avec un facteur de lissage de valeur 5. Les mêmes paramètres qu'à la Section 1.6.1, c'est-à-dire $R = 8$, $K_1 = 1000$, $K_2 = 500$, sont utilisés. Cependant, cette fois-ci, nous effectuons l'analyse sur les covariables `humidity`, `walkscore` et `total_precip_mm`. Avec cette configuration, nous pouvons initialiser un objet de la classe `BKTRRegressor` et démarrer un échantillonnage MCMC avec la méthode `mcmc_sampling`.

```
>>> from pyBKTR.bktr import BKTRRegressor
>>> from pyBKTR.examples.bixi import BixiData
>>> from pyBKTR.kernels import (
...     KernelMatern, KernelParameter,
```

```
...     KernelPeriodic, KernelSE)
>>> from pyBKTR.plots import plot_spatial_betas, plot_temporal_betas
>>> from pyBKTR.tensor_ops import TSR

>>> TSR.set_params('float32', 'cpu', 1)
>>> bixi_data = BixiData()

>>> fm = 'nb_departure ~ humidity + walkscore + total_precip_mm'
>>> p_len = KernelParameter(value = 7, is_fixed = True)
>>> k_loc_per = KernelSE() * KernelPeriodic(period_length = p_len)

>>> bktr_regressor = BKTRRegressor(
...     formula = fm,
...     data_df = bixi_data.data_df,
...     spatial_positions_df = bixi_data.spatial_positions_df,
...     temporal_positions_df = bixi_data.temporal_positions_df,
...     rank_decomp = 8,
...     spatial_kernel = KernelMatern(smoothness_factor = 5),
...     temporal_kernel = k_loc_per,
...     burn_in_iter = 1000,
...     sampling_iter = 500
... )
>>> bktr_regressor.mcmc_sampling()
```

La complétion de l'échantillonnage rend disponible une panoplie de méthodes et propriétés depuis l'instance `bktr_regressor`. L'une de ses propriétés les plus utiles est le résumé (`summary`) qui offre plusieurs informations à propos de l'échantillonnage MCMC. Pour afficher ce résumer, nous pouvons simplement utiliser la commande suivante :

```
>>> print(bktr_regressor.summary)
```

```
==========================================================================
                         BKTR Regressor Summary
==========================================================================
Formula: nb_departure ~ 1 + humidity + walkscore + total_precip_mm
```

```
Burn-in iterations: 1000
Sampling iterations: 500
Rank decomposition: 8
Nb Spatial Locations: 587
Nb Temporal Points: 196
Nb Covariates: 4
======================================================================
In Sample Errors:
  RMSE: 0.072
  MAE: 0.053
Computation time: 523.54s.
======================================================================
-- Spatial Kernel --
Matern 5/2 Kernel
Parameter(s):
                      Mean   Median       SD  Low2.5p  Up97.5p
lengthscale         22.364   22.513    1.931   18.925   26.711

-- Temporal Kernel --
Composed Kernel (Mul)
  SE Kernel
  Parameter(s):
                        Mean   Median       SD  Low2.5p  Up97.5p
  lengthscale          6.641    6.636    0.107    6.438    6.881
  *
  Periodic Kernel
  Parameter(s):
                        Mean   Median       SD  Low2.5p  Up97.5p
  lengthscale          1.012    1.012    0.019    0.973    1.048
  period length         Fixed Value: 7.000
======================================================================
Beta Estimates Summary (Aggregated Per Covariates)

                       Mean   Median       SD
feature
Intercept             0.323    0.277    0.236
```

```
humidity              -0.147   -0.127    0.097
walkscore              0.222    0.168    0.250
total_precip_mm       -0.221   -0.194    0.139
==========================================================================
```

Ce sommaire, nous rappelle notamment les paramètres utilisés durant la régression (nombre d'itérations, formule, dimensions) et nous présente également l'erreur estimée (RMSE et MAE) sur l'échantillon pour le nombre de départs. Il permet aussi d'observer les valeurs clés de la distribution estimée de chacun des hyperparamètres des noyaux utilisés. Par exemple, nous pouvons voir que le paramètre d'échelle estimé du noyau Matérn semble oscillé autour de 22.4. Enfin, la dernière partie nous donne une vue d'ensemble des coefficients estimés par notre modèle. Elle calcule la moyenne de tous les coefficients estimés à travers le temps et l'espace, et ce, par covariable. Cela nous montre que l'humidité et les précipitations ont en moyenne un effet négatif sur le nombre de départs d'une station, tandis que le `walkscore` a plutôt un effet positif.

En employant la fonction `plot_hyperparams_traceplot`, il est possible d'obtenir une représentation plus visuelle de l'estimation des hyperparamètres du modèle utilisé. Nous pouvons obtenir un graphique montrant l'évolution des paramètres au fil des K_2 itérations d'échantillonnage. En fournissant une liste de noms comme paramètre à la fonction, il est possible de ne tracer qu'un sous ensemble des paramètres du modèle. Il est donc possible de tracer l'évolution des paramètres d'échelles du noyau SE et du noyau Matérn en utilisant l'instruction suivante :

```
>>> from pyBKTR.plots import plot_hyperparams_traceplot
>>> plot_hyperparams_traceplot(bktr_regressor, [
...     'Spatial - Matern 5/2 Kernel - lengthscale',
...     'Temporal - SE Kernel - lengthscale'])
```

Le résultat de cette commande est présenté dans la Figure B1. Cette figure nous permet de constater que les valeurs des paramètres d'échelle des noyaux

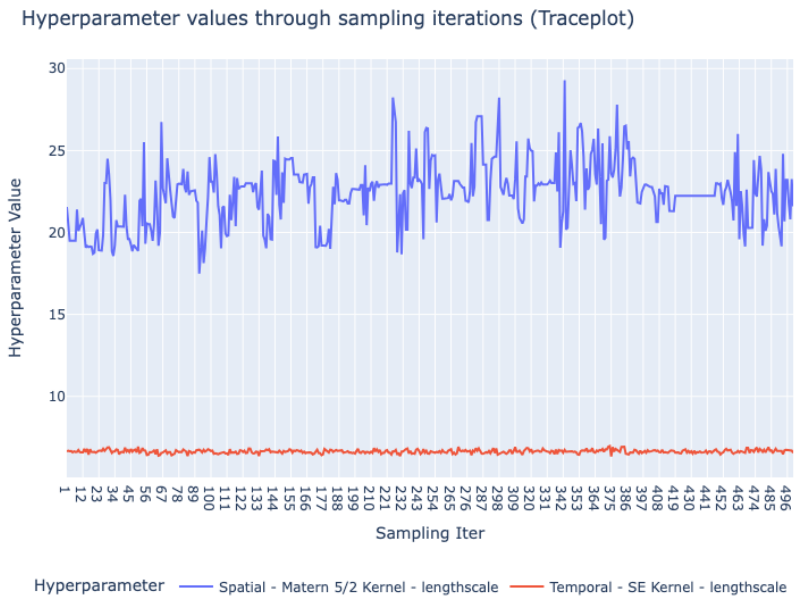


FIGURE B1 – Graphique de traçage des hyperparamètres d'échelle pour les noyaux SE et Matérn utilisés pour modéliser les données BIXI au travers de $K_2 = 500$ itérations d'échantillonnage

Matérn et SE oscillent de manière constante autour de leur valeur estimée, qui est de 22 pour le noyau Matérn et 7 pour le noyau SE, et ce, tout au long des K_2 itérations d'échantillonnages. Une autre méthode permettant d'obtenir une visualisation différente de ces valeurs est la fonction `plot_hyperparams_dists`. Nous pouvons appeler cette dernière à nouveau sur les paramètres d'échelles des noyaux SE et Matérn de la manière suivante :

```
>>> from pyBKTR.plots import plot_hyperparams_dists
>>> plot_hyperparams_dists(bktr_regressor, [
...     'Spatial - Matern 5/2 Kernel - lengthscale',
...     'Temporal - SE Kernel - lengthscale'])
```

La Figure B2 présente la distribution estimée de chacun des paramètres d'échelle. Ces distributions montrent un dispersement assez symétrique par rapport aux moyennes estimées et semblent bien être centrés autour de 22 pour le noyau Matérn et de 7 pour le noyau SE.

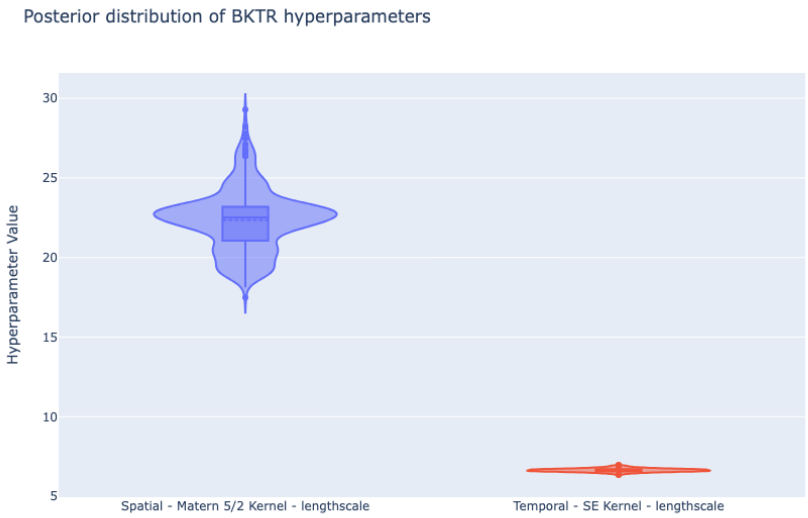


FIGURE B2 – Distribution estimée des hyperparamètres d'échelle pour les noyaux SE et Matérn ayant été utilisés dans la modélisation des données BIXI au cours de $K_2 = 500$ itérations d'échantillonnage

En utilisant l'objet `bktr_regressor`, il est possible d'appeler plusieurs fonctions qui permettent de visualiser les coefficients estimés. Pour une location donnée, la fonction `plot_temporal_betas` permet notamment de tracer l'estimation de coefficients au fil du temps pour un ensemble de covariables. Voici comment utiliser cette fonction pour un emplacement comme *7114 - Smith / Peel* :

```
>>> from pyBKTR.plots import plot_temporal_betas
>>> feature_labels = ['humidity', 'walkscore', 'total_precip_mm']
>>> plot_temporal_betas(
...     bktr_regressor,
...     plot_feature_labels = feature_labels,
...     spatial_point_label = '7114 - Smith / Peel')
```

Le graphique résultant est affiché dans la Figure B3. Cette représentation est très intéressante puisqu'elle nous permet d'observer l'évolution des coefficients au cours du temps. En effet, nous pouvons remarquer que comme dans la Section 1.6.1 l'effet des coefficients relié à la météo et à la localisation est plus prononcé en milieu de saison qu'en début ou fin de saison.

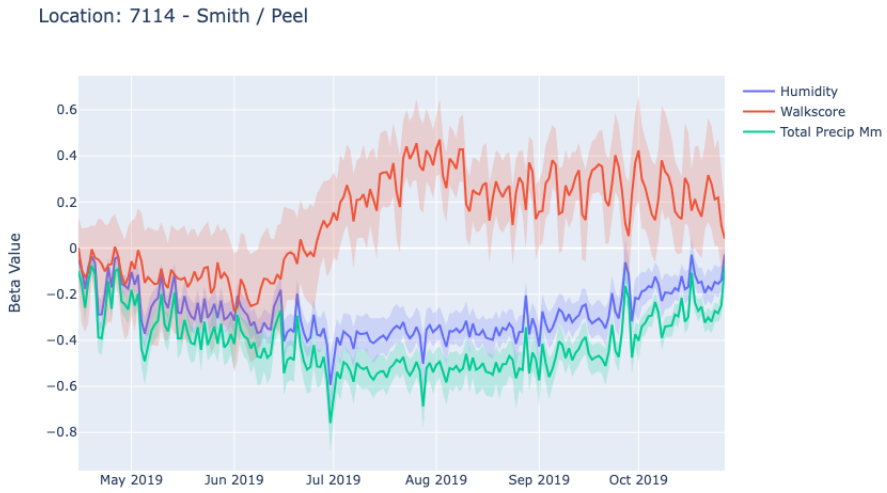


FIGURE B3 – Évolution des coefficients estimés des covariables dans le temps pour la localisation spatiale spécifique *7114 - Smith / Peel*.

Une autre fonction, `plot_spatial_betas`, permet également d'illustrer la variation des coefficients, mais cette fois, à travers l'espace pour un point temporel donné. Nous pouvons appliquer cette fonction, pour la date du 19 juillet 2019, à la même sélection de covariables utilisée précédemment à l'aide de la commande suivante :

```
>>> from pyBKTR.plots import plot_spatial_betas
>>> plot_spatial_betas(
...     bktr_regressor,
...     plot_feature_labels = feature_labels,
...     temporal_point_label = '2019-07-19',
...     nb_cols = 3)
```

Le résultat de cette fonction est présenté dans la Figure B4. Il nous permet également de voir, qu'en date du 19 juillet 2019, l'influence des coefficients reliés à la météo ou au `walkscore` est plus importante au centre de la ville de Montréal que dans sa périphérie.

Si nous souhaitons obtenir des résultats numériques pour effectuer d'autres analyses, nous pouvons utiliser la méthode `get_beta_summary_df` qui est dis-

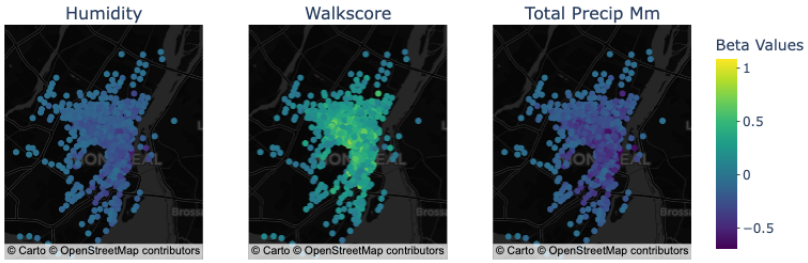


FIGURE B4 – Évolution des coefficients estimés des covariables dans l'espace pour la date spécifique du *19 juillet 2019*.

ponible depuis l'objet `bktr_regressor` à la suite de la complétion de l'échantillonnage MCMC. Cette méthode fournit l'estimation de coefficients pour un ensemble de dimensions données. Les trois paramètres de cette méthode sont `spatial_labels`, `temporal_labels` et `feature_labels` qui prennent des listes de libellés pour les localisations, les points temporels et les covariables respectivement. Ces paramètres définissent les dimensions pour lesquels les coefficients doivent être estimés. Dans le cas où une valeur nulle (`None`) serait fourni à un paramètre, il faudrait évaluer les coefficients pour toutes les valeurs de cette dimension. Pour obtenir l'estimation des coefficients de toutes les covariables, la position spatiale *7114 - Smith / Peel* et les dates du 18 et 19 juillet 2019, nous pouvons utiliser l'instruction suivante :

```
>>> b_df = bktr_regressor.get_beta_summary_df(
...     spatial_labels=['7114 - Smith / Peel'],
...     temporal_labels=['2019-07-18', '2019-07-19'],
...     feature_labels=None)
>>> print('Columns: ' + ' '.join(b_df.columns.to_list()))
>>> print()
>>> b_df = b_df[['Mean', 'SD', 'Low2.5p', 'Up97.5p']]
>>> b_df = b_df.droplevel(0)
>>> print(b_df.to_string(float_format='{:.2f}'.format))
```

```
Columns: Mean SD Min Q1 Median Q3 Max Low2.5p Up97.5p

                           Mean   SD  Low2.5p  Up97.5p
2019-07-18 Intercept        0.89 0.06     0.78     0.99
           humidity        -0.37 0.03    -0.44    -0.29
           walkscore        0.30 0.07     0.18     0.44
           total_precip_mm -0.52 0.04    -0.60    -0.45
2019-07-19 Intercept        0.85 0.06     0.72     0.97
           humidity        -0.35 0.04    -0.42    -0.26
           walkscore        0.37 0.08     0.23     0.54
           total_precip_mm -0.51 0.03    -0.57    -0.45
```

La commande présentée nous permet de visualiser un résumé de la distribution pour chaque combinaison possible des trois dimensions mentionnées. La première partie du message de sortie montre toutes les colonnes disponibles dans la table provenant de la méthode `get_beta_summary_df`. Dans notre cas, afin d'avoir des résultats affichés de manière plus concise, nous avons uniquement montré les résultats de quatre de ces colonnes et nous avons omis le premier index de la table qui indiquait la même valeur de position spatiale pour toutes les lignes. Cette table illustre que durant les 2 jours, les valeurs de coefficients estimées sont assez similaires et que la précipitation ainsi que l'humidité ont tous les deux un effet négatif sur le nombre de départs, tandis que le `walkscore` a un effet positif.

En conclusion, nous espérons que cet exemple fait preuve de la similitude entre les bibliothèques BKTR et PYBKTR et qu'il permet de montrer à quel point il est facile de les utiliser à la fois en R et en Python.