Raum-Zeit-Gaussfeld: SPDE-Ansatz mit R-INLA

Der im Folgenden beschriebene Modellansatz enstammt dem SPDE-Buch (Krainski et al. 2018). Eine darauf basierende Umsetzung, welche diesem Projekt ähnelt, kann in Cameletti et al. (2013) nachgeschlagen werden.

Modellansatz

Wir haben punktbezogene Daten ("point-referenced data"), d.h. die Daten wurden an festen, bekannten Messorten s_i, $i = 1, \dots, d$, zu den diskreten Zeitpunkten $t = 1, \dots, T$ erhoben. Wir nehmen an, dass der beobachtete Prozess an den diskreten Messorten normalverteilt ist (siehe vorheriges Kapitel). Je näher die Orte beeinander liegen, desto stäker die Abhängigkeit der Messungen. Für jeden Zeitpunkt t kann diese räumliche Abhängigkeit durch ein d-dimensionales Gaussfeld $\eta_t \sim \mathcal{N}(0, K(\sigma_\eta^2, \nu, \kappa))$ mit Matérn-Kovarianz

$$\text{Cov}(\eta(s_i, t), \eta(s_j, t')) = \begin{cases} 0, & t \neq t' \\ \sigma_\eta^2 C(h), & t = t' \end{cases} \tag{1}$$

beschrieben werden. Hierbei ist $C(h)$, $h = ||s_i - s_j||$, die Matérn-Funktion mit Parametern ν und κ.

Neben der räumlichen Abhängigkeit existiert auch eine zeitliche Abhängigkeit; diese modellieren wir mithilfe eines stationären AR(1)-Prozesses, welcher - gegeben einen Ort s - durch ein T-dimensionales GMRF mit Präzisionsmatrix Q_T beschrieben werden kann. Für die Interaktionen der räumlichen und zeitlichen Dimensionen nehmen wir an, dass an zwei verschiedenen Zeitpunkten keine Abhängigkeiten zwischen Messungen an verschiedenen Orten bestehen (dies wird in (1) spezifiziert).

Zusammen ergibt sich für $i = 1, \dots, d$, $t = 2, \dots, T$ ein Raum-Zeit-Gaussfeld

$$\xi(s_i, t) = a\xi(s_i, t-1) + \eta(s_i, t), \tag{2}$$

mit $|a| < 1$ und $\xi(s_i, 1) \sim \mathcal{N}(0, \sigma_\eta^2)$.

Zusätzlich nehmen wir an, dass der gemessene Feinstaub von meteorologischen und geographischen Kovariablen abhängt und die Daten mit einem Messfehler behaftet sind. Mit diesen Überlegungen ergibt sich für die Zeitpunkte $t = 1, \dots, T$, dass

$$y(s_i, t) = x(s_i, t)\beta + z(s_i, t)u + \xi(s_i, t) + \epsilon(s_i, t), \tag{3}$$

$$\xi(s_i, t) = a\xi(s_i, t-1)\beta + \eta(s_i, t), \tag{4}$$

wobei $\beta = (\beta_1, \dots, \beta_p)^T$ die festen Effekte und $u = (u_1, \dots, u_q)^T$ die zufälligen Effekte der Kovariablen $x(s_i, t) = (x_1(s_i, t), \dots, x_p(s_i, t))$ bzw. $x(s_i, t) = (x_1(s_i, t), \dots, x_q(s_i, t))$, gemessen am Ort s_i zur Zeit t, bezeichnen. Der Messfehler, auch als Nugget-Effekt bekannt, wird durch $\epsilon(s_i, t) \sim \mathcal{N}(0, \sigma_\epsilon^2)$ modelliert.

Unser Ziel ist es, an beliebigen Orten Vorhersagen zu machen, d.h. nicht nur an den Messorten, Dazu wird angenommen, dass dem beobachteten diskreten Prozess ein Gaussfeld $y(s, t) = \{y(s, t) : (s, t) \in \mathcal{D} \subseteq \mathbb{R}^2 \times \mathbb{R}\}$ mit stetiger Indexmenge zugrundeliegt.

Da die Kovarianzmatrizen dieses Gaussfelds im Allgemeinen sehr dicht sind, ist eine klassische Inferenz aufgrund des Rechenaufwands meist nicht umsetzbar. Eine Möglichkeit der Abbhilfe besteht darin, die Kovarianzmatrizen stark zu vereinfachen. Eine weitere Möglichkeit, welche hier verfolgt wird, besteht darin, das Gaussfeld mithilfe des beobachteten diskreten Gaussfelds zu repräsentieren.

Beim SPDE-Ansatz wird angenommen, dass das stetige latente Gaussfelds Lösung einer stochastischen partiellen Differentialgleichung ist, welche mit der Finiten-Elemente-Methode gelöst wird. Durch diese Verbindung kann gezeigt werden, dass das GF mithilfe eines GMRF reräsentiert werden kann. Dazu wird die stetige Indexmenge der räumlichen Koordinaten in aneinandergrenzende, getrennte, irreguläre Dreiecke unterteilt. Dieses Verfahren ist in der Geodäsie als Triangulation bekannt. Mithilfe der Triangulation kann das räumliche d-dimensionale GF η mit Matérn-Kovarianz zu einem n-dimensionalen GMRF $\tilde{\eta}$ mit Präzision Q_S modifiziert werden. Genauer wird das GMRF $\tilde{\eta}$ für jeden Zeitpunkt $t = 1, \ldots, T$ an den Knotenpunkten $k_1, \ldots, k_n$ der Dreicke mit Ausprägung $\tilde{\eta}_t(k_j)$, $j = 1, \ldots, n$, modelliert. Wenn η_t in (2) durch $\tilde{\eta}_t$ ersetzt wird (und entsprechend ξ_t durch das n-dimensionale $\tilde{\xi}_t$), erhalten wir damit ein Raum-Zeit-GMRF

$$\tilde{\xi} := (\tilde{\xi}_1, \ldots, \tilde{\xi}_T)^T \sim \mathcal{N}(0, Q^{-1}), \tag{5}$$

wobei

$$\tilde{\xi}_t = a\tilde{\xi}_{t-1} + \tilde{\eta}_t, \tag{6}$$

mit $\tilde{\xi}_t := (\tilde{\xi}_t(k_1), \ldots, \tilde{\xi}_t(k_n))$ und $\eta_t \sim \mathcal{N}(0, Q_S^{-1})$. Aufgrund der obigen Annahme der Interaktionsunabhängigkeit ist Q_S für alle Zeiptunkte identisch. Die Präzisionsmatrix des Raum-Zeit-Gaussfelds ergibt sich als $Q = Q_T \otimes Q_S$. Hierbei ist Q_T die T-dimensionale Präzisionsmatrix des autoregressiven GMRFs für einen festen Ort s.

Weiterhin kann eine Ausprägung des latenten, stetigen Gaussfelds (ohne Messfehler und mit EW 0) an einem beliebigen Ort s zur Zeit t als Basisexpansion $\sum_{j=1}^{n} \psi_j(s)\tilde{\xi}_t(k_j)$ approximiert werden. Somit entstammen für jeden Zeitpunkt t die Koeffizienten der Basisfunktionen dem räumlichen GMRF $\tilde{\xi}_t$. Mit einer geeigneten $d \times n$-Matrix $A_{\text{est},t}$ der Basisfunktionen (wobei $A_{\text{est},t}$ von t nur über den Ort, welcher theoretisch für verschiedene Zeitpunkte variieren kann, abhängt) kann das diskrete Gaussfeld, welches wir in (3) definiert haben, nun als

$$y_t = X_t\beta + Z_t u + A_{\text{est},t}\tilde{\xi}_t + \epsilon_t \tag{7}$$

geschrieben werden, wobei alle Messorte jeweils in $y_t = (y(s_1, t), \ldots, y(s_d, t))$, sowie in den Kovariablenmatrizen $X_t = (x(s_1, t)^T, \ldots, x(s_d, t)^T)$ und $Z_t = (z(s_1, t)^T, \ldots, z(s_d, t)^T)$ zusammengefasst wurden, und $\tilde{\xi}_t = (\tilde{\xi}_t(k_1), \ldots, \tilde{\xi}_t(k_n))$ das Raum-Zeit-GMRF zum Zeitpunkt t bezeichnet.

Für eine Vorhersage an beliebigen Orten $s = (s_1, \ldots, s_m)$ zum Zeitpunkt t kann statt $A_{\text{est},t}$ entsprechend eine $m \times n$-Matrix $A_{\text{pred},t}$ der Basisfunktionen verwendet werden.

Ferne kann gezeigt werden, dass für alle Raum-Zeit-Orte (s, t) nur jeweils maximal drei Basisfunktionen $\psi_j(s), j = 1, \ldots, n$ ungleich Null sind (eine, falls s auf einem Knoten liegt; zwei, falls s auf der Kante eines Dreiecks liegt; drei sonst). Somit sind die Matrizen $A_{\text{est},t}$ und $A_{\text{pred},t}$ dünn besetzt, wodurch eine effiziente Berechnung ermöglicht wird.

Der SPDE-Ansatz kann in R mithilfe des Pakets INLA durchgeführt werden. Hierbei werden Laplace-Approximationen für die marginalen Posterioris des Modells verwendet.

Im Folgenden werden wir zunächst die erforderlichen Vorbereitungen für das beschriebene Verfahren durchführen, also insbesondere die Triangulation, sowie die Erzeugung der benötigten Matrizen der Basisfunktionen, um anschließend den Posteriori-EW von $y(s, t)$ auf einem zuvor definierten Gitter zu bestimmen. Zu schätzen

sind die festen und zufälligen Effekte (β, u), sowie das GMRF $\tilde{\xi}$ mit Präzisionsmatrix Q. Die Hyperparameter des Modells lauten $\theta := (\sigma_\eta^2, a, \kappa, \sigma_\epsilon^2)$ (ν ist fest und ergibt sich durch κ, für Details siehe Kapitel 2, Lindgren, Rue, and Lindström (2011)). Für die Prioris dieser Parameter verwenden wir die standardmäßig implementierten Prioris des Pakets INLA.

Daten einlesen

Zunächst laden wir die benötigten Pakete und lesen die zuvor aufbereiteten Daten ein:

```
library(tidyverse)
library(lubridate)
library(INLA)
library(sf)

rm(list = ls())

path_workfiles <- "/Users/patrickschulze/desktop/inla_research_project/data/workfiles"
path_results <- "/Users/patrickschulze/desktop/inla_research_project/data/results"

filename_in <- paste(path_workfiles, "pm_all.csv", sep ="/")

# Validierungs-/Trainingsdaten einlesen
filename_val <- paste(path_workfiles, "pm_all_val.csv", sep ="/")
filename_train <- paste(path_workfiles, "pm_all_train.csv", sep ="/")
(pm_data_val <- read_delim(filename_val, delim=",", col_types = "fiddddddffddd")%>%
  dplyr::select(c(-pm10,-lon,-lat, -hour)))
```

```
## # A tibble: 1,427 x 9
##    ID        t logpm10     X     Y day   temperature humidity pressure
##    <fct> <int>   <dbl> <dbl> <dbl> <fct>       <dbl>    <dbl>    <dbl>
##  1 3         1    2.67  682. 5337. 1            24.5     67.9   95629.
##  2 6         1    2.55  685. 5332. 1            21.3     80.7   95629.
##  3 12        1    3.60  686. 5331. 1            27.3     61.7   95629.
##  4 27        1    2.44  688. 5337. 1            34.0     35.9   95629.
##  5 35        1    2.21  688. 5337. 1            26.0     98.6   95629.
##  6 39        1    2.60  689. 5337. 1            22.7     99.7   95629.
##  7 41        1    2.49  689. 5335. 1            23.7     99.9   95629.
##  8 49        1    2.31  689. 5341. 1            22.6     99.9   95629.
##  9 71        1    2.20  692. 5334. 1            22.8     99.9   95629.
## 10 82        1    2.57  694. 5342. 1            23.5     99.9   95629.
## # ... with 1,417 more rows
```

```
(pm_data_train <- read_delim(filename_train, delim=",", col_types = "fiddddddffddd")%>%
  dplyr::select(c(-pm10,-lon,-lat, -hour)))
```

```
## # A tibble: 13,720 x 9
##    ID        t logpm10     X     Y day   temperature humidity pressure
##    <fct> <int>   <dbl> <dbl> <dbl> <fct>       <dbl>    <dbl>    <dbl>
##  1 1         1    2.25  678. 5339. 1            21.3     82.6   95629.
##  2 2         1    2.43  680. 5335. 1            23.2     99.9   95629.
##  3 4         1    2.74  683. 5340. 1            21.5     88.4   95629.
##  4 5         1    2.44  684. 5333. 1            22.7     69.3   95629.
##  5 7         1    0     685. 5333. 1            21.4     64.7   95629.
##  6 8         1    2.48  685. 5334. 1            20.6     99.9   95629.
```

```
##  7 9         1    2.97  685. 5335. 1            21.2     99.9   95629.
##  8 10        1    2.39  686. 5334. 1            24.2     99.9   95629.
##  9 13        1    2.25  686. 5337. 1            23.2     93.6   95629.
## 10 14        1    2.44  687. 5337. 1            21.7     99.9   95629.
## # ... with 13,710 more rows
```

Für die Vorhersage verbleiben $d = 90$ verschiedene Messorte und die Anzahl der diskreten Zeiptunkte ist $T = 24 \cdot 7 = 168$:

```
unique(pm_data_train$ID) %>% length
```

```
## [1] 90
```

```
(n_t <- max(pm_data_train$t))
```

```
## [1] 168
```

Kovariablen für Prädiktion & räumliches Vorhersage-Gitter spezifieren

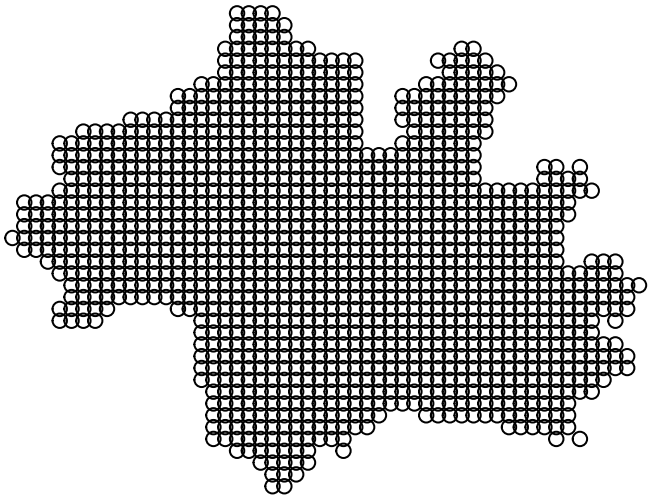
Als erstes werden die meteorologischen Kovariablen für die Vorhersage, sowie der Zeitpunkt der Vorhersage spezifiziert. Für die Vorhersage nehmen wir vereinfachend an, dass die meteorologischen Kovariablen an allen Orten gleich sind.

```
hour_pred <- 21 # Stunde der Vorhersage (von 1 bis 24)
day_pred <- 4 # Tag (25.07 = Tag 1, 26.07 = Tag 2, ..., 14.08 = Tag 21) der Vorhersage
temperature_pred <- 18 # Temperatur bei Vorhersage
humidity_pred <- 99 # Luftfeuchtigkeit (in % Wasserdampf an Luft) bei Vorhersage
pressure_pred <- 95000 # Luftdruck (in Pascal) bei Vorhersage
(t_pred <- hour_pred+24*(day_pred-1)) # Vorhersagezeitpunkt aus Stunde+Tag berechnen
```

```
## [1] 93
```

Zudem erstellen wir ein räumliches Vorhersagegitter für München; dies ist nicht mit dem Dreiecksgitter der Triangulation zu verwechseln.

```
# Dateiname des shape-files von München
filename_boundary <- paste(path_workfiles, "munich_pol.shp", sep="/")
# shape-file einlesen
munich_boundary <- sf::st_read(filename_boundary, quiet = TRUE)$geometry
# Projiziere WGS84 lat/lon Koordinaten zu metrischen Koordinaten (UTM Zone 32 für München)
munich_boundary <- sf::st_transform(munich_boundary,25832)
# Erzeuge die Gitterpunkte
munich_grid <- munich_boundary %>%
  sf::st_make_grid(cellsize = 500, what = "centers") %>% # Rechteckiges Gitter in Box erzeugen
  sf::st_intersection(munich_boundary) # Nur den Teil innerhalb der Stadtgrenzen behalten
# Projektion zu metrischem Referenzsystem bei Erzeugung des Gitters
# ermöglicht quadratische Anordnung der Punkte (ohne Projektion wäre Länge ungleich Breite
plot(munich_grid)
```



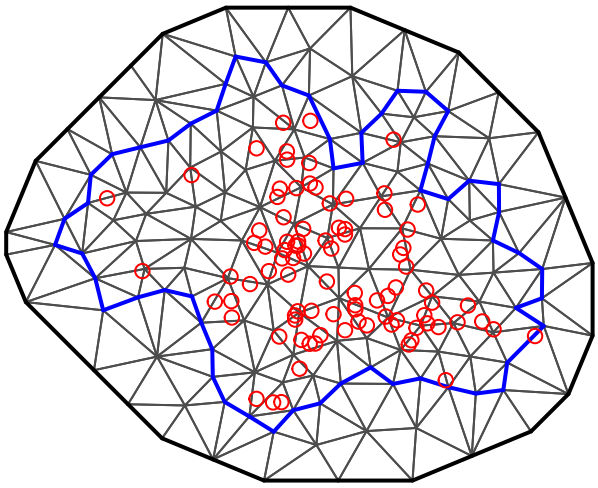
```
# Koordinaten der Gitterpunkte (in km) speichern
munich_grid <- sf::st_coordinates(munich_grid)
munich_grid <- matrix(unlist(munich_grid/1000), ncol = 2)
```

Triangulation

Für die Triangulation verwenden wir ebenfalls die Stadtgrenzen von München; alternativ könnten die Beobachtungspunkte verwendet werden, um das Gitter zu erzeugen, was jedoch zu Problemen führen kann, wenn eine Vorhersage auf dem gesamten Stadtgebiet erwünscht ist.

```
sp_munich_boundary <- as(munich_boundary/1000, 'Spatial')
inla_munich_boundary <- inla.sp2segment(sp_munich_boundary)
mesh <- inla.mesh.2d(boundary = inla_munich_boundary,
                     max.edge = c(3,6), cutoff = 1.5, offset = c(-0.1, -0.1))
plot(mesh, asp=1)
est_coord <- unique(cbind(pm_data_train$X, pm_data_train$Y))
points(est_coord, col = "red")
```

Constrained refined Delaunay triangulation



```
mesh$n
```

```
## [1] 155
```

Insgesamt haben wir $n = 155$ Knotenpunkte erzeugt. Wichtig ist, dass die Dreicke regelmäßig angeordnet sind und ähnlich groß sind. Zudem sollten zusätzliche Knoten um die Stadtgrenzen gesetzt werden, um eine hohe Varianz an den Stadtgrenzen zu vermeiden. Durch offset $= c(-0.1, -0.1)$ wird ein Rand von 10% des Durchmessers des Stadtgebiets verwendet. Mithilfe der Einstellung cutoff $= 1.5$ wird erreicht, dass die Knoten keinen zu kleinen Abstand haben, wohingegen max.edge $= c(3, 6)$ die maximale Kantenlänge festgelegt.

Das oben beschriebende SPDE-Modell wird durch den folgenden Befehl initialisiert:

```
spde <- inla.spde2.matern(mesh=mesh)
```

Inla-Objekte für Schätzdaten erzeugen

Basierend auf der Triangulation und den beobachteten Daten kann nun A_{est} erzeugt werden, wobei alle Matrizen $A_{\text{est},t}, t = 1, \ldots, T$ in A_{est} zusammengefasst werden.

```
A_est <- inla.spde.make.A(mesh, loc = cbind(pm_data_train$X, pm_data_train$Y),
                          group = pm_data_train$t, n.group = n_t)
nrow(A_est) # = 13691 (Anzahl Beobachtungen; hängt von gesampelten Validierungsorten ab)
```

```
## [1] 13720
```

```
ncol(A_est) # 26040 = 155*168 Spalten
```

```
## [1] 26040
```

```
head(rowSums(A_est!=0), n=100) # pro Spalte nur drei Einträge ungleich 0
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

```
head(rowSums(A_est), n=100) # Summe ist immer 1
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Hier erkennt man, wie bereits oben beschrieben, dass pro Spalte nur drei Einträge ungleich null sind. Die Summe dieser Ausprägungen der Basisfunktionen ist eins. Die Anzahl der Zeilen der Matrix A_{est} enstpricht der Anzahl der Beobachtungen (diese ist ungleich $d \cdot T$, da für manche Zeitpunkte keine Messungen vorhanden sind). Die Spaltenanzahl ergibt sich als $n \cdot T = 155 \cdot 168$ (hier werden die potentiellen Orte zu allen Zeitpunkten in einer Matrix zusammenfasst, weshalb sich die Dimension als nT ergibt).

Abschließend werden alle Objekte mit inla.stack in einem Objekt kombiniert; inla.stack vereinfacht zudem die gegebenen Matrizen, z.B. wird jede Nullzeile von A eliminiert:

```
# Generiere örtliche und zeitliche Indices
field_indices <- inla.spde.make.index("field",n.spde=mesh$n,n.group=n_t)
# Kombiniere die Objekte
stack_est <- inla.stack(
  data=list(logPM10=pm_data_train$logpm10),
  A=list(A_est,1), # Kovariablen werden 1-zu-1 gemappt
  effects=list(c(field_indices, list(Intercept=1)),
```

```
              list(pm_data_train[,c("X", "Y", "day", "temperature",
                                    "humidity", "pressure")])), tag="est")
```

Inla-Objekte für Vorhersage- und Validierungsdaten erzeugen

Nun werden noch die Matrizen A_{pred} und A_{val} für die Vorhersage- und Validierungsdaten erzeugt und alle Objekte kombiniert.

```
# ------------------------------------------------------------------------------------------
# ------------ Inla-Objekte für Vorhersage-Gitter erzeugen ----------------------------
# ------------------------------------------------------------------------------------------
# Kovariablenwerte welche für Vorhersage verwendet werden
covariate_matrix_std <- tibble(
  X = munich_grid[,1], Y = munich_grid[,2], day = day_pred,
  temperature = temperature_pred, humidity = humidity_pred, pressure = pressure_pred)

# Erzeuge Projektionsmatrix für Vorhersagegitter (diesmal nicht Inla-Gitter)
A_pred <- inla.spde.make.A(mesh, loc=as.matrix(munich_grid),
                           group=t_pred, n.group=n_t)
# wie zuvor, alle Objekte kombinieren; setze für Prädiktion Response auf NA
stack_pred <- inla.stack(
  data=list(logPM10=NA),
  A=list(A_pred,1),
  effects=list(c(field_indices, list(Intercept=1)),
               list(covariate_matrix_std)),
  tag="pred")
# ------------------------------------------------------------------------------------------
# ------------ Inla-Objekte für Validierungsdaten erzuegen ----------------------------
# ------------------------------------------------------------------------------------------
A_val = inla.spde.make.A(mesh, loc = cbind(pm_data_val$X, pm_data_val$Y),
                         group = pm_data_val$t, n.group = n_t)
stack_val <- inla.stack(
  data = list(logPM10 = pm_data_val["logpm10"]),
  A=list(A_val,1),
  effects=list(c(field_indices, list(Intercept=1)),
               list(pm_data_val[,c("X", "Y", "day", "temperature", "humidity", "pressure")])),
  tag="val")
# ------------------------------------------------------------------------------------------
# Füge Inla-Objekte für Daten & Vorhersage-Gitter zusammen
stack <- inla.stack(stack_est, stack_pred, stack_val)
```

Modellgleichungen spezifizieren, Modelle fitten & Vorhersagen machen

Im Folgenden werden verschiedene Kovariablenspezifizierungen gefittet. Im einfachsten Modell nehmen wir an, dass die geographische Lage und der Tag der Erhebung - abgesehen vom indirekten Effekt über das latente Raum-Zeit-Gaussfeld - keinen zusätzlichen Einfluss auf den messbaren Feinstaub haben.

```
# Modellgleichung 1
formula1<-(logPM10 ~ -1 + Intercept + temperature + humidity + pressure +
```

```
            f(field, model=spde, group=field.group, control.group=list(model="ar1")))
# Modell 1 fitten
# Zuerst Modell nur mit Datengitter laufen lassen, um Startwerte zu finden
mod_mode1 <- inla(formula1, data=inla.stack.data(stack_est, spde=spde), family="gaussian",
                  control.predictor=list(A=inla.stack.A(stack_est), compute=FALSE))
# Nun mit Startwerten von erstem Durchlauf fitten & und auf Vorhersage-Gitter anwenden
mod1 <- inla(formula1, data = inla.stack.data(stack, spde = spde), family = "gaussian",
             control.predictor = list(A = inla.stack.A(stack), compute = TRUE),
             control.mode = list(theta = mod_mode1$mode$theta, restart = FALSE),
             quantiles = NULL,
             control.results = list(return.marginals.random = FALSE,
                                    return.marginals.predictor = FALSE))
# ------------------------------------------------------------------------------
```

Für das nächste Modell nehmen wir die geographische Lage als zufälligen Effekt (entspricht einem sensorspezifischen Effekt) mit in das Modell auf:

```
# ------------------------------------------------------------------------------
# Modellgleichung 2
formula2<-(logPM10 ~ -1 + Intercept + f(X, model = "rw1") + f(Y, model = "rw1") +
             temperature + humidity + pressure +
             f(field, model=spde, group=field.group, control.group=list(model="ar1")))
# Modell 2 fitten
# Zuerst Modell nur mit Datengitter laufen lassen, um Startwerte zu finden
mod_mode2 <- inla(formula2, data=inla.stack.data(stack_est, spde=spde), family="gaussian",
                  control.predictor=list(A=inla.stack.A(stack_est), compute=FALSE))
# Nun mit Startwerten von erstem Durchlauf fitten & und auf Vorhersage-Gitter anwenden
mod2 <- inla(formula2, data = inla.stack.data(stack, spde = spde), family = "gaussian",
             control.predictor = list(A = inla.stack.A(stack), compute = TRUE),
             control.mode = list(theta = mod_mode2$mode$theta, restart = FALSE),
             quantiles = NULL,
             control.results = list(return.marginals.random = FALSE,
                                    return.marginals.predictor = FALSE))
# ------------------------------------------------------------------------------
```

Nun nehmen wir zusätzlich noch den Tageseffekt als zufälligen Effekt auf:

```
# ------------------------------------------------------------------------------
# Modellgleichung 3
formula3<-(logPM10 ~ -1 + Intercept + f(X, model = "rw1") + f(Y, model = "rw1") +
             + f(day, model = "rw2") + temperature + humidity + pressure +
             f(field, model=spde, group=field.group, control.group=list(model="ar1")))
# Modell 3 fitten
# Zuerst Modell nur mit Datengitter laufen lassen, um Startwerte zu finden
mod_mode3 <- inla(formula3, data=inla.stack.data(stack_est, spde=spde), family="gaussian",
                  control.predictor=list(A=inla.stack.A(stack_est), compute=FALSE))
# Nun mit Startwerten von erstem Durchlauf fitten & und auf Vorhersage-Gitter anwenden
mod3 <- inla(formula3, data = inla.stack.data(stack, spde = spde), family = "gaussian",
             control.predictor = list(A = inla.stack.A(stack), compute = TRUE),
             control.mode = list(theta = mod_mode3$mode$theta, restart = FALSE),
             quantiles = NULL,
             control.results = list(return.marginals.random = FALSE,
                                    return.marginals.predictor = FALSE))
# ------------------------------------------------------------------------------
```

Für alle Modelle speichern wir den Posteriori-Erwartungswert des linearen Prädiktors für jeden Ort des

Vorhersage-Gitters. Außerdem transformieren wir die Response zurück auf die ursprüngliche Skala:

```
# Prädiktion
index_pred <- inla.stack.index(stack = stack, tag = "pred")$data
pm10_mean_mod1 <- exp(mod1$summary.fitted.values[index_pred, "mean"]-1)
pm10_mean_mod2 <- exp(mod2$summary.fitted.values[index_pred, "mean"]-1)
pm10_mean_mod3 <- exp(mod3$summary.fitted.values[index_pred, "mean"]-1)

# Ergebnisse der Prädiktion speichern
pm_pred_mod1 <- data.frame(pm10_mean = pm10_mean_mod1, X = munich_grid[,1],
                           Y = munich_grid[,2], day = day_pred, hour = hour_pred,
                           temperature = temperature_pred, humidity = humidity_pred,
                           pressure = pressure_pred)
filename_pred1 <- paste(path_results,"d4h21_pred_mod1.csv", sep = "/")
write.table(pm_pred_mod1, file = filename_pred1, sep = ",", row.names = FALSE)
pm_pred_mod2 <- data.frame(pm10_mean = pm10_mean_mod2, X = munich_grid[,1],
                           Y = munich_grid[,2], day = day_pred, hour = hour_pred,
                           temperature = temperature_pred, humidity = humidity_pred,
                           pressure = pressure_pred)
filename_pred2 <- paste(path_results,"d4h21_pred_mod2.csv", sep = "/")
write.table(pm_pred_mod2, file = filename_pred2, sep = ",", row.names = FALSE)
pm_pred_mod3 <- data.frame(pm10_mean = pm10_mean_mod3, X = munich_grid[,1],
                           Y = munich_grid[,2], day = day_pred, hour = hour_pred,
                           temperature = temperature_pred, humidity = humidity_pred,
                           pressure = pressure_pred)
filename_pred3 <- paste(path_results,"d4h21_pred_mod3.csv", sep = "/")
write.table(pm_pred_mod3, file = filename_pred3, sep = ",", row.names = FALSE)
```

Validierung

Um die Güte der Modelle zu ermitteln und miteinander zu vergleichen wird nun noch die Posteriori an den Validierungsmessorten mit den tatsächlichen Messungen verglichen. Dazu berechnen wir zunächst Posteriori-Erwartungswert und -Standardabweichung des Gaussfelds y an den Validierungsorten. Damit können die standardisierten, normalverteilten Residuen berechnet werden; für diese wird dann bestimmt, ob sie sich innerhalb des 95%-Kredibilitätsintervalls befinden. Die relative Anzahl der Validierungsdaten für die dies zutrifft, wird jeweils in der Variable validation$cover gespeichert. Zudem kann bspw. der RMSE der Modelle (hier auf der log-Skala) geschätzt und verglichen werden.

```
index_val <- inla.stack.index(stack = stack, tag = "val")$data
# Posteriori-Erwartungswert und -Standardabweichung berechnen und speichern
# Modell 1
mean_val1 <- mod1$summary.linear.predictor[index_val,"mean"]
sd_val1 <- mod1$summary.linear.predictor[index_val,"sd"]
pm_val_mod1 <- bind_cols(pm_data_val, tibble(mean = mean_val1, sd = sd_val1,
                                             hp_mean = mod1$summary.hyperpar[1,"mean"]))
filename_val1 <- paste(path_results,"d4h21_val_mod1.csv", sep = "/")
write.table(pm_val_mod1, file = filename_val1, sep = ",", row.names = FALSE)
# Modell 2
mean_val2 <- mod2$summary.linear.predictor[index_val,"mean"]
sd_val2 <- mod2$summary.linear.predictor[index_val,"sd"]
pm_val_mod2 <- bind_cols(pm_data_val, tibble(mean = mean_val2, sd = sd_val2,
                                             hp_mean = mod2$summary.hyperpar[1,"mean"]))
filename_val2 <- paste(path_results,"d4h21_val_mod2.csv", sep = "/")
```

```
write.table(pm_val_mod2, file = filename_val2, sep = ",", row.names = FALSE)
# Modell 3
mean_val3 <- mod3$summary.linear.predictor[index_val,"mean"]
sd_val3 <- mod3$summary.linear.predictor[index_val,"sd"]
pm_val_mod3 <- bind_cols(pm_data_val, tibble(mean = mean_val3, sd = sd_val3,
                                             hp_mean = mod3$summary.hyperpar[1,"mean"]))
filename_val3 <- paste(path_results,"d4h21_val_mod3.csv", sep = "/")
write.table(pm_val_mod3, file = filename_val3, sep = ",", row.names = FALSE)
```

```
# Relative Anzahl in Kredibilitätsintervall und RMSE bestimmen
# Modell 1
filename_val1 <- paste(path_results,"d4h21_val_mod1.csv", sep = "/")
data_val1 <- read_delim(filename_val1, delim=",")
validation1 <- list()
validation1$res <- data_val1$logpm10 - data_val1$mean
validation1$res_std = validation1$res /
  sqrt(data_val1$sd^2 + 1/data_val1$hp_mean)
validation1$p = pnorm(validation1$res_std)
validation1$cover = mean((validation1$p>0.025) & (validation1$p<0.975), na.rm=TRUE)
validation1$rmse = sqrt(mean(validation1$res^2, na.rm=TRUE))
# Modell 2
filename_val2 <- paste(path_results,"d4h21_val_mod2.csv", sep = "/")
data_val2 <- read_delim(filename_val2, delim=",")
validation2 <- list()
validation2$res <- data_val2$logpm10 - data_val2$mean
validation2$res_std = validation2$res /
  sqrt(data_val2$sd^2 + 1/data_val2$hp_mean)
validation2$p = pnorm(validation2$res_std)
validation2$cover = mean((validation2$p>0.025) & (validation2$p<0.975), na.rm=TRUE)
validation2$rmse = sqrt(mean(validation2$res^2, na.rm=TRUE))
# Modell 3
filename_val3 <- paste(path_results,"d4h21_val_mod3.csv", sep = "/")
data_val3 <- read_delim(filename_val3, delim=",")
validation3 <- list()
validation3$res <- data_val3$logpm10 - data_val3$mean
validation3$res_std = validation3$res /
  sqrt(data_val3$sd^2 + 1/data_val3$hp_mean)
validation3$p = pnorm(validation3$res_std)
validation3$cover = mean((validation3$p>0.025) & (validation3$p<0.975), na.rm=TRUE)
validation3$rmse = sqrt(mean(validation3$res^2, na.rm=TRUE))
# ------------------------------------------------------------------------------
```

Modell 1 schneidet deutlich schlechter ab als Modelle 2 und 3:

```
c(validation1$cover, validation2$cover, validation3$cover)
```

```
## [1] 0.4162088 0.8598901 0.8598901
```

```
c(validation1$rmse, validation2$rmse, validation3$rmse)
```

```
## [1] 3.601778 1.052497 1.052330
```

Das Aufnehmen des Tages bringt keinen Mehrwert, da durch den stündlichen AR(1)-Prozess bereits die gesamte zeitliche Information im Modell enthalten ist. Daher entscheiden wir uns für Modell 2. Die Validierung ist abhängig davon, welche Sensoren zu Beginn für die Validierung gesampelt werden. In unseren Versuchen bewegte sich validation$cover zwischen 80 und 97%.

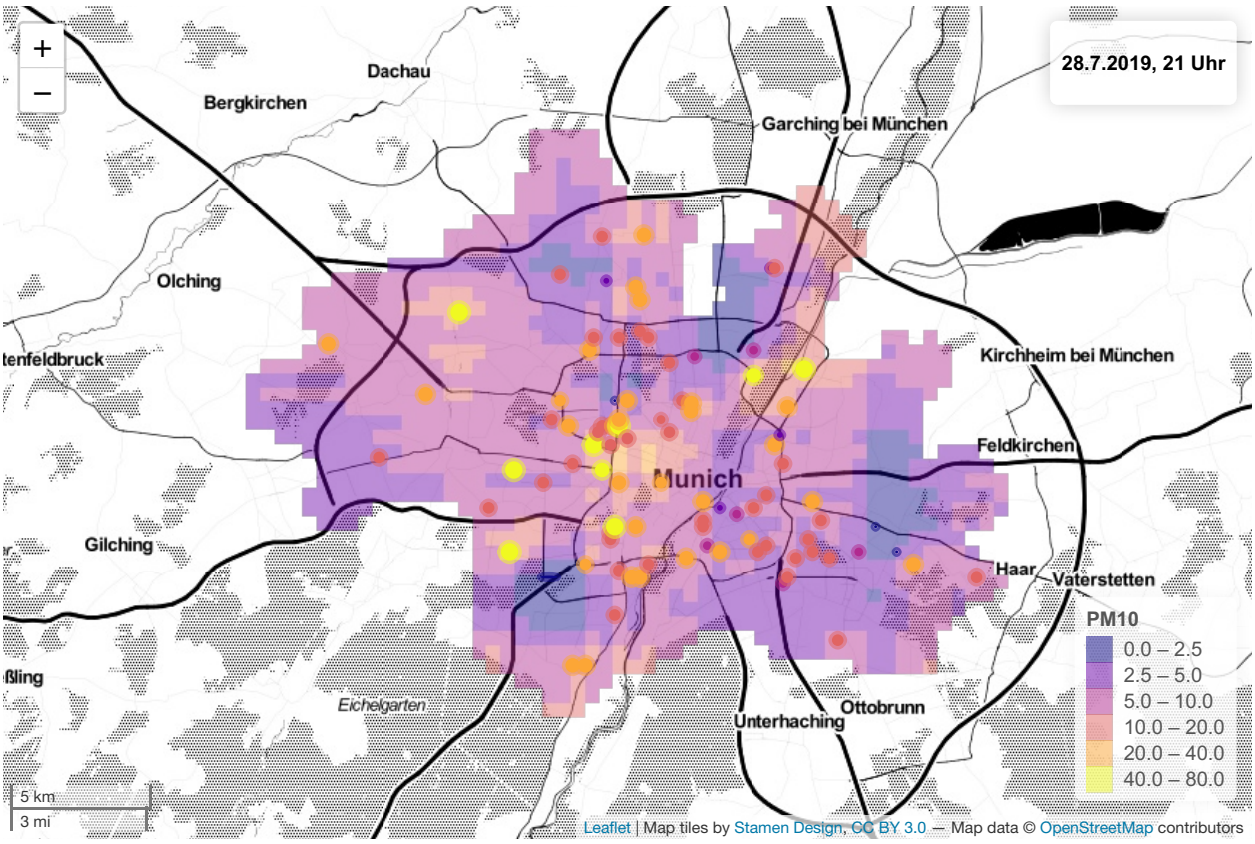
Modell 2 haben wir noch für einen weiteren Zeitpunkt, den 26.07.2019 um 13 Uhr, gefittet.

Plots

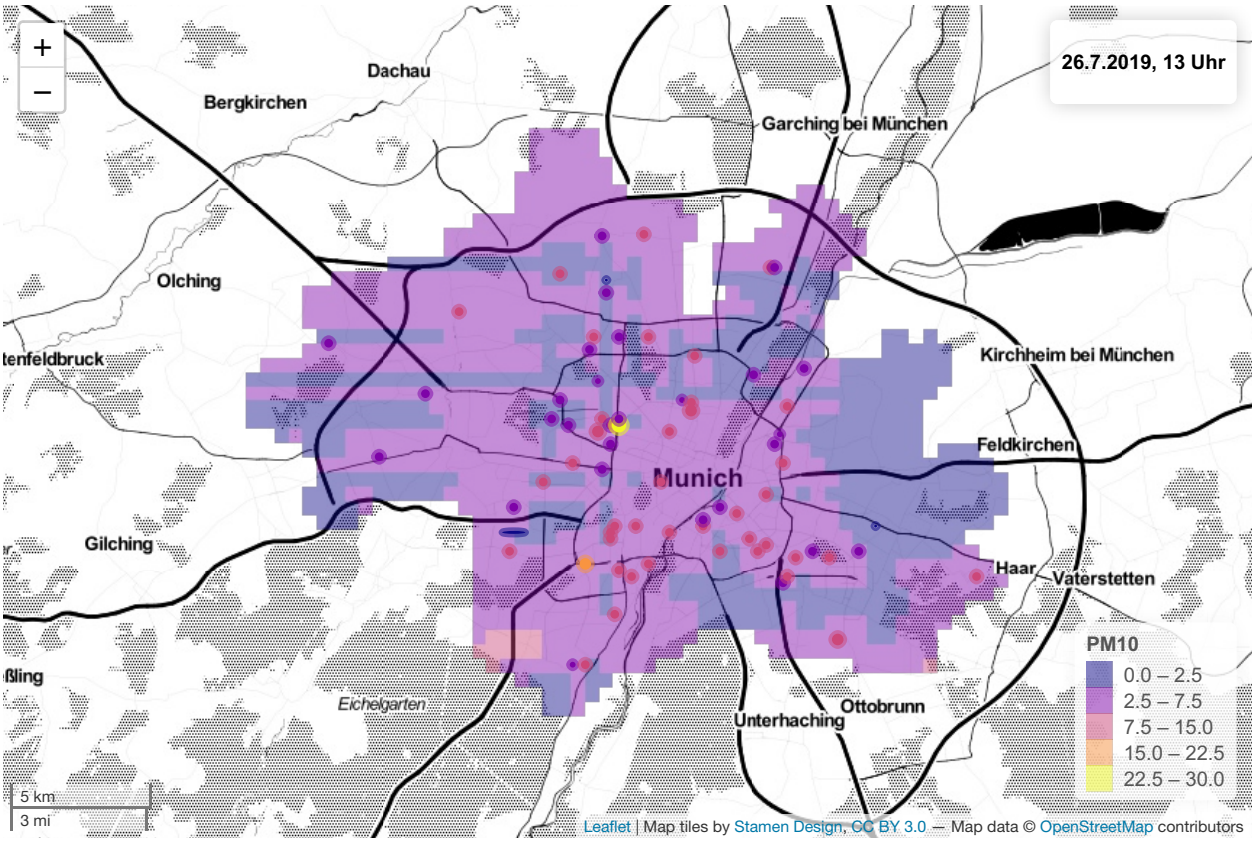
Für beide Zeitpunkte plotten wir abschließend die tatsächlichen Messungen und die Vorhersagen. Hierfür eignet sich das Paket leaflet in Kombination mit dem Paket Raster.

```
library(tidyverse)
library(raster)
library(leaflet)
library(sp)
library(sf)
library(fasterize)
library(lubridate)
library(htmltools)

rm(list = ls())

path_workfiles <- "/Users/patrickschulze/desktop/inla_research_project/data/workfiles"
path_results <- "/Users/patrickschulze/desktop/inla_research_project/data/results"

filename_all_data <- paste(path_workfiles,"pm_all.csv", sep = "/")
pm_all <- read_delim(filename_all_data, delim=",") %>%
  dplyr::select(c(-ID,-t,-logpm10,-X,-Y))
filename_pred <- paste(path_results,"d4h21_pred_mod2.csv", sep = "/")
pm_pred <- read_delim(filename_pred, delim=",")
# ------------------------------------------------------------------------------------
# Mithilfe der Stadtgrenzen und des Vorhersage-Gitters Raster zum Plotten erzeugen
# Stadtgrenzen einlesen
filename_boundary <- paste(path_workfiles, "munich_pol.shp", sep="/")
munich_boundary <- sf::st_read(filename_boundary, quiet = TRUE)
# Mithilfe des Vorhersage-Gitters Anzahl der lat- und lon-Koordinaten berechnen
ncols_template <- length(seq(min(pm_pred["X"]), max(pm_pred["X"]), by = 0.5))
nrows_template <- length(seq(min(pm_pred["Y"]), max(pm_pred["Y"]), by = 0.5))
# Raster erzeugen
template <- raster::raster(munich_boundary, nrows = nrows_template, ncols = ncols_template)
munich_raster <- fasterize::fasterize(munich_boundary, template)
raster::crs(munich_raster) <- sp::CRS("+proj=longlat +datum=WGS84")
# Vorhersage-Gitter von UTM Zone 32 (metrisch) zurück zu WGS84 (lat/lon) transformieren
coord_latlon <- pm_pred[,c("X", "Y")] %>%
  mutate(X = 1000*X, Y = 1000*Y) %>%
  sf::st_as_sf(coords = c("X", "Y"), crs = 25832) %>%
  sf::st_transform(4326) %>%
  sf::st_coordinates()
pm_pred <- pm_pred %>% dplyr::rename(lon = "X", lat = "Y") %>%
  dplyr::mutate(lon = coord_latlon[,1], lat = coord_latlon[,2])
# Vorhersagen auf Gitter zu Raster mappen
rast <- raster::rasterize(
  x = pm_pred[,c("lon", "lat")], y = munich_raster, field = pm_pred$pm10_mean
)
# Koordinatenreferenzsystem für Raster setzen
raster::crs(rast) <- sp::CRS("+proj=longlat +datum=WGS84")
```

```r
# PM10-Mittelwert je Tag und Stunde der Vorhersage pro Sensor berechnen
pm_hourly_h <- pm_all %>% filter(day==pm_pred$day[1], hour==pm_pred$hour[1]) %>%
  group_by(lon, lat) %>% summarise(pm10=mean(pm10))

# Farbpalette für Pm10 Werte kreieren
max_value <- round(max(c(pm_hourly_h$pm10, pm_pred$pm10_mean)), digits = -1)
pal.bins <- unique(c(0,2.5,5,10,20,40,max_value))
pal <- colorBin("plasma", bins=pal.bins, na.color = "transparent")

# Datum erzeugen
dy <- (pm_pred$day[1]+24)%%30
mn <- ifelse(dy<25,8,7)
dt <- paste(dy,mn,"2019",sep=".")
hr <- paste(pm_pred$hour[1], "Uhr")
# Vorhersage- und Messwerte für gegebenen Tag und Stunde plotten
# Punkte korrespondieren zu Messwerten, Rasterzellen zu Vorhersagewerten
path_plots <- paste(path_results,"plots",sep="/")
my_title <- tags$p(tags$style("p {color: black; font-size:14px}"),
                   tags$b(paste(dt, hr, sep=", ")))

m <- leaflet()
m %>% addProviderTiles(providers$Stamen) %>%
  leaflet::addRasterImage(rast, colors = pal, opacity = 0.45) %>%
  leaflet::addCircles(lng = pm_hourly_h[["lon"]], lat = pm_hourly_h[["lat"]],
                      color = pal(pm_hourly_h$pm10), fillOpacity = 1,
                      stroke = TRUE, radius = log(pm_hourly_h$pm10)*75) %>%
  leaflet::addLegend("bottomright",
                     pal = pal, values = values(rast),
                     title = "PM10") %>%
  leaflet::addControl(my_title, position = "topright" )%>%
  leaflet::addScaleBar(position = c("bottomleft"))
```


28.7.2019, 21 Uhr
Dachau
Bergkirchen
Garching bei München
Olching
Kirchheim bei München
Feldkirchen
Munich
Gilching
Haar
Vaterstetten
Eichelgarten
Unterhaching
Ottobrunn
PM10
0.0 – 2.5
2.5 – 5.0
5.0 – 10.0
10.0 – 20.0
20.0 – 40.0
40.0 – 80.0
5 km
3 mi
Leaflet | Map tiles by Stamen Design, CC BY 3.0 — Map data © OpenStreetMap contributors

Mit einem identischen Code kann der Plot für den anderen Zeitpunkt erzeugt werden:



Literatur

Cameletti, Michela, Finn Lindgren, Daniel Simpson, and Håvard Rue. 2013. "Spatio-Temporal Modeling of Particulate Matter Concentration Through the Spde Approach." *AStA Advances in Statistical Analysis* 97 (2). Springer: 109–31.

Krainski, Elias T, Virgilio Gómez-Rubio, Haakon Bakka, Amanda Lenzi, Daniela Castro-Camilo, Daniel Simpson, Finn Lindgren, and Håvard Rue. 2018. *Advanced Spatial Modeling with Stochastic Partial Differential Equations Using R and Inla*. CRC Press.

Lindgren, Finn, Håvard Rue, and Johan Lindström. 2011. "An Explicit Link Between Gaussian Fields and Gaussian Markov Random Fields: The Stochastic Partial Differential Equation Approach." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73 (4). Wiley Online Library: 423–98.