S4) Synthetic Case Studies: Heat Flow

This notebook contains the code of the paper "Bayesian Calibration of Imperfect Computer Models using Physics-Informed Priors". The models are fitted in rstan and the code is available in the folder "STAN/Heat_equation".

Load libraries

```r
# uncomment to install
# install.packages("rstan")
# install.packages("ggplot2")
# install.packages("ggpubr")
# install.packages("reshape2")
# install.packages("lhs")
library(rstan)
library(lhs)
library(ggplot2)
library(ggpubr)
library(reshape2)
theme_set(theme_classic()) # set ggplot theme
rstan_options(auto_write = TRUE)
options(mc.cores = 3) # allocate 3 cores (for each model we run 3 chains in parallel)
```

Section 4: Synthetic Case Studies: Heat Flow

The non-homogeneous Heat equation is given by the following space-time dependent differential equation

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = f. \tag{1}$$

In the 1D case the heat equation describes the distribution of heat, $u(t,x)$ in a thin metal rod and the differential equation reduces to

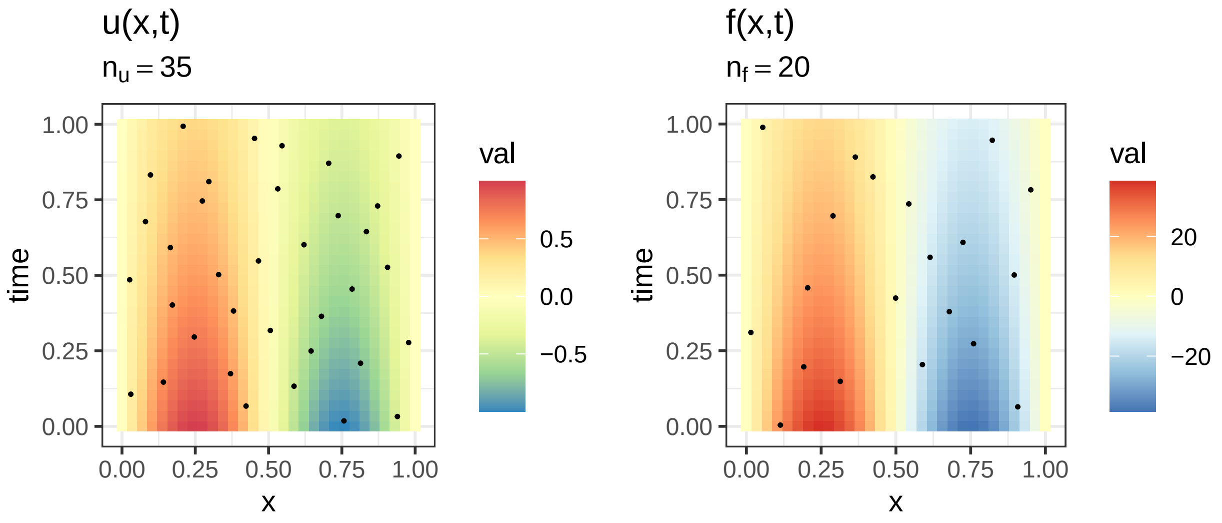
$$\frac{\partial u(t,x)}{\partial t} - \alpha \frac{\partial^2 u(t,x)}{\partial^2 x} = f(t,x). \tag{2}$$

For $\alpha = 1$, the functions $f(t,x) = \exp(-t)(4\pi^2 - 1)\sin(2\pi x)$ and $u(t,x) = \exp(-t)\sin(2\pi x)$ satisfy this equation. We will use this solution to simulate data for the synthetic case study and our aim is to estimate α and produce model predictions.

More specifically, we simulate 35 data points for $u(t,x)$ and 20 data points for $f(t,x)$ randomly on $[0,1]^2$ (see Figure below). We add Gaussian noise to the simulated u and f values and we obtain the observed data as follows, $y_u = u(t,x) + \varepsilon_u$, where $\varepsilon_u \sim N(0, 0.2^2)$ and $y_f = f(t,x) + \varepsilon_f$, where $\varepsilon_f \sim N(0, 1^2)$.

```r
f = function(x,t){
  (exp(-t))*(4*(pi^2)-1)*sin(2*pi*x)
}

u = function(x,t){
  (exp(-t))*sin(2*pi*x)
}

s=1111
set.seed(s)
nU=35
nF=20
# space and time locations according to
# a maximin experimental design
xtF=maximinLHS(nF,2)
colnames(xtF) = c("x","t")
set.seed(s*2)
xtU=maximinLHS(nU,2)
colnames(xtU) = c("x","t")
u_true = u(xtU[,1],xtU[,2])
f_true = f(xtF[,1],xtF[,2])
# add noise
set.seed(0)
yU = u_true+rnorm(nU,0,0.2)
set.seed(1)
yF = f_true+rnorm(nF,0,1)
# create prediction grid
nx = 20
xseq = seq(0,1,length.out = nx)
xtU_pred = xtF_pred = expand.grid(xseq,xseq)

data_noisy_pred = list(nU = nrow(xtU), nF = nrow(xtF), xtU = xtU, xtF = xtF, yU = yU, yF = yF
                       , xtU_pred = xtU_pred, nU_pred = nrow(xtU_pred)
                       , xtF_pred = xtF_pred, nF_pred = nrow(xtF_pred))
```

The corresponding values (val) of functions $u(x,t)$ and $f(x,t)$ in space and time, and the observed data (black dots) locations for each function.



Section 4.2 HF Case Study 1: Fully Bayesian analysis

To develop the physics-informed prior for the Heat equation, we assume that the heat follows a GP prior, $u(t,x) \sim GP(\mu, K((t,x),(t',x')))$ where we use an anisotropic squared exponential kernel,

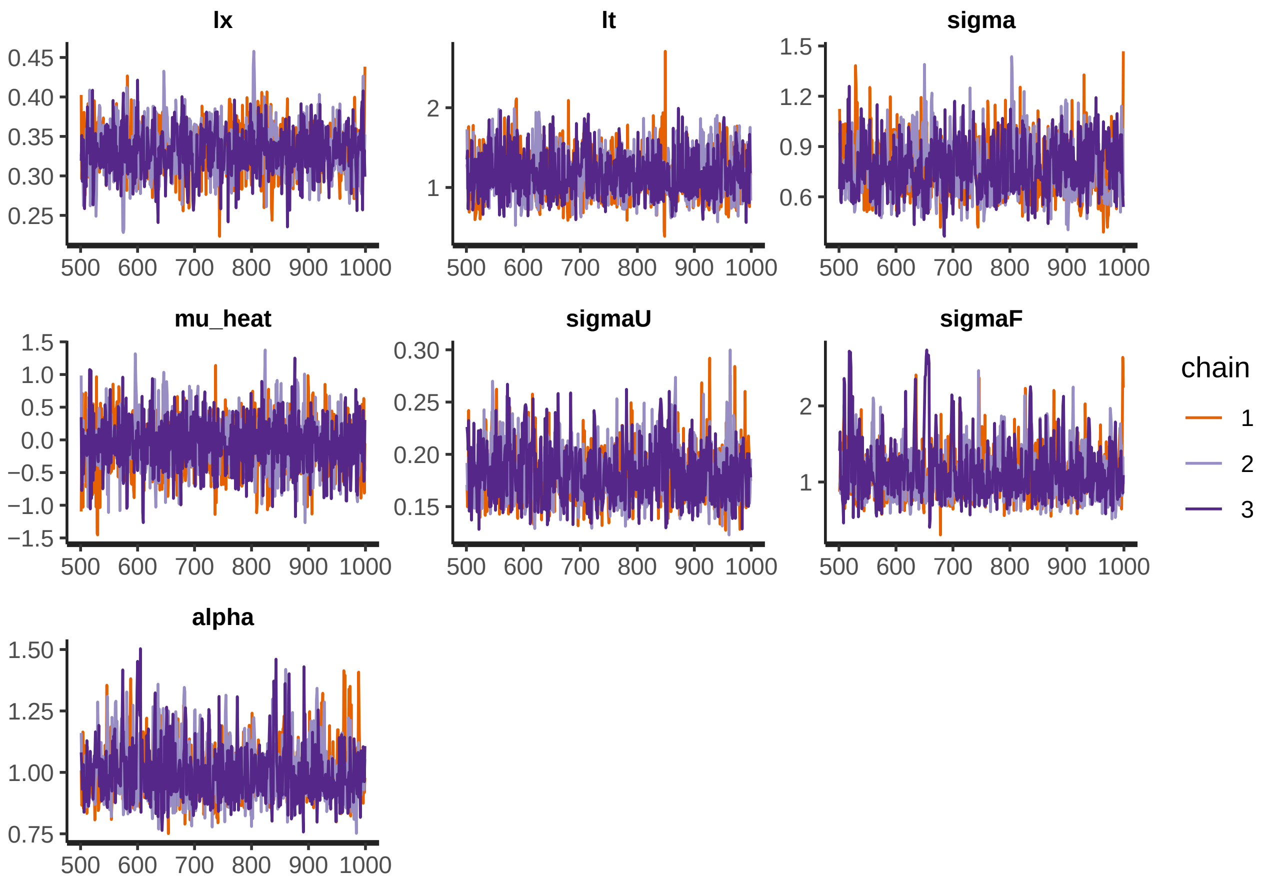
$$K_{uu}((t,x),(t',x')) = \sigma^2 exp\left(-\frac{1}{2l_t^2}(t-t')^2\right) exp\left(-\frac{1}{2l_x^2}(x-x')^2\right)$$

and μ is a constant. For more details see Appendix B.2 in the paper.

Stan code (select `eval=TRUE` in the code chunk to see the stan code):

```r
writeLines(readLines('STAN/Heat_equation/Heat_PI_prior.rds'))
```

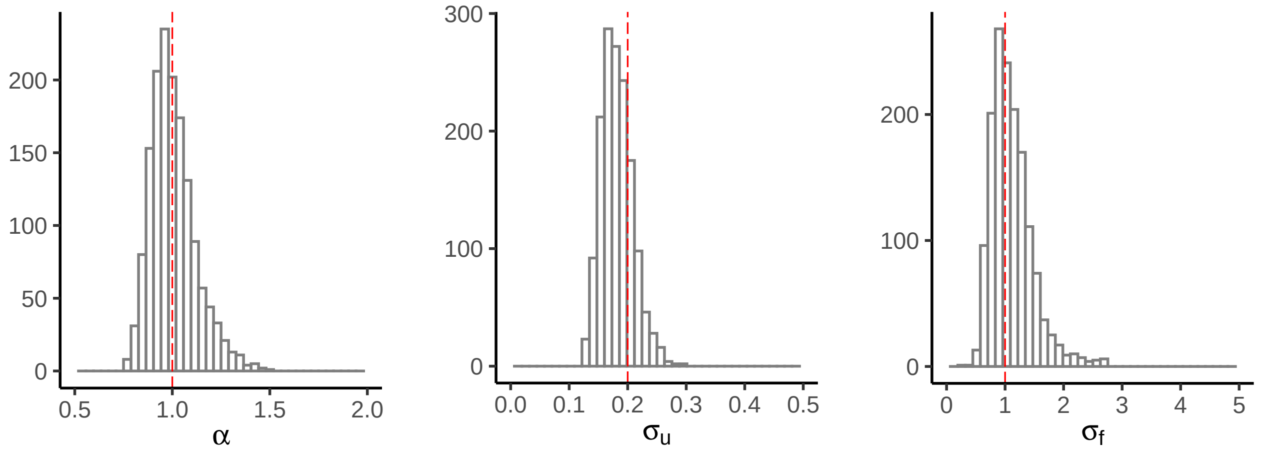
```r
fit_heat = stan(file  = 'STAN/Heat_equation/Heat_PI_prior.stan',
                data = data_noisy_pred,
                chains = 3,
                iter = 1000,
                seed = 123
)
stan_trace(fit_heat, pars = names(fit_heat)[1:7])
```



```r
post_fitHeat = as.data.frame(extract(fit_heat))
```

Posterior distributions for the parameters of interest (Figure 9 in the paper)

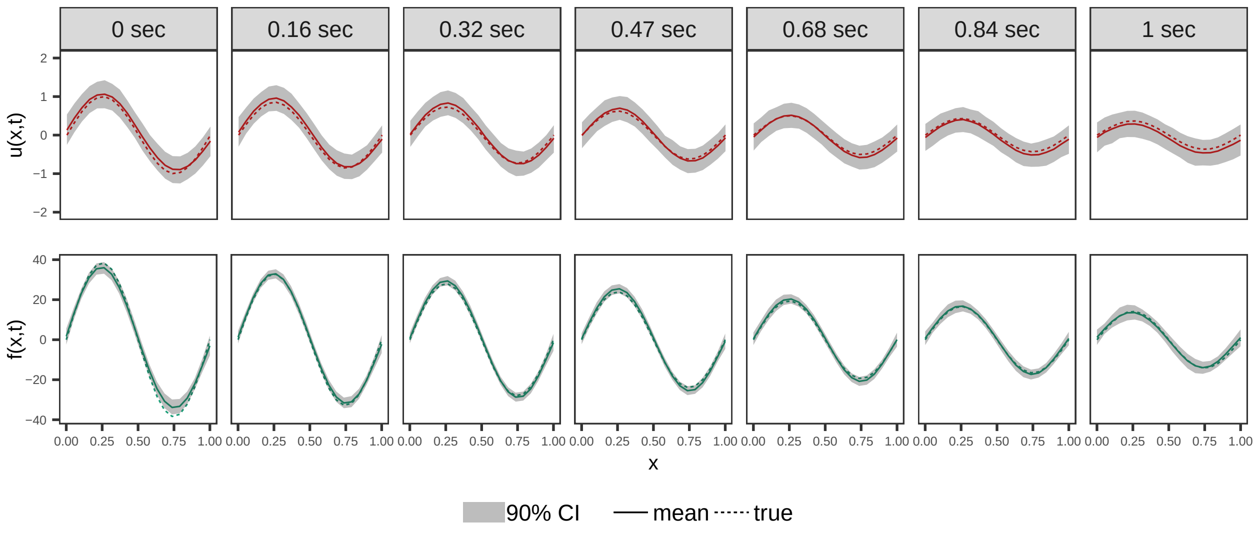
```r
post = post_fitHeat
pl_alpha = ggplot()+
  geom_histogram(data = post, aes(alpha), bins=40, fill="white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",size=0.3, colour = "red")+
  xlim(0.5,2)+
  ylab("")+xlab(expression(alpha))+
  theme(legend.position = c(0.8,0.7)
        ,legend.key.size = unit(0.35, 'cm')
  )

pl_unoise = ggplot()+
  geom_histogram(data = post, aes(sigmaU), bins=40, fill="white", colour = "grey50")+
  geom_vline(xintercept = 0.2, linetype = "longdash",size=0.3, colour = "red")+
  xlim(0,0.5)+
  ylab("")+xlab(expression(sigma[u]))+
  theme(legend.position = c(0,0.5)
        ,legend.key.size = unit(0.35, 'cm')
  )

pl_fnoise = ggplot()+
  geom_histogram(data = post, aes(sigmaF), bins=40, fill="white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",size=0.3, colour = "red")+
  xlim(0,5)+
  ylab("")+xlab(expression(sigma[f]))+
  theme(legend.position = c(0,0.5)
        ,legend.key.size = unit(0.35, 'cm')
  )

pl_post_Heat_Unbiased = ggarrange(pl_alpha,pl_unoise,pl_fnoise, nrow=1)
pl_post_Heat_Unbiased
```



```r
# ggsave("Figures/Post_Heat_Unbiased.pdf", plot = pl_post_Heat_Unbiased , width = 18, height = 4, units
```

Predictions for both u and f (Figure 10 in paper). Equations can be found in Section 2.1.

```r
# Unbiased Simulation case study
pred_true = data.frame(u = u(xtU_pred[,1], xtU_pred[,2])
                       , f = f(xtF_pred[,1], xtF_pred[,2])
                       , x = xtU_pred[,1], time = xtU_pred[,2]
                       )

post_df = post
cn = colnames(post_df)
Upred = post_df[,grep("y_U",cn)]
Fpred = post_df[,grep("y_F",cn)]
Umean = colMeans(Upred)
UCIs = apply(Upred, 2, quantile, probs = c(0.05,0.95))
Fmean = colMeans(Fpred)
FCIs = apply(Fpred, 2, quantile, probs = c(0.05,0.95))
rm(post_df)
df_pred_time = data.frame(
  Umean=Umean, Fmean=Fmean
  , Utrue = pred_true$u
  , Ftrue = pred_true$f
  , U_low = UCIs[1,], U_upper = UCIs[2,]
  , F_low = FCIs[1,], F_upper = FCIs[2,]
  , x = pred_true$x, time = pred_true$time
)

uniq_time = unique(df_pred_time$time)
n.unique=length(uniq_time)
ntimes=7
ind = c(round(seq(1,n.unique, length.out =  ntimes)),n.unique)
df_pred_time = df_pred_time[df_pred_time$time%in% uniq_time[ind], ]
df_pred_time$time_sec = paste(round(df_pred_time$time,2), " sec", sep = "")
cols = c("#1B9E77","#E41A1C", "#E69F00", "#377EB8")
scaleFUN = function(x) sprintf("%.1f", x)
t.size = 5
l.size = 5
size.line = 0.3

pl_Unbiased_pred_U=ggplot()+
  geom_line(data=df_pred_time,aes(x=x, y=Umean, linetype = "mean"),
            colour = cols[2], size=size.line)+
  geom_line(data=df_pred_time,aes(x=x, y=Utrue, linetype = "true"),
            colour = cols[2], size=size.line)+
  geom_ribbon(data=df_pred_time,aes(ymin=U_low, ymax=U_upper, x=x, fill = "95% CI"), alpha = 0.3)+
  scale_fill_manual("",values=c("95% CI" = "grey12"))+
  facet_wrap(~time_sec, nrow = 1) +
  theme_bw()+
  theme(legend.position = "none"
        , legend.title = element_blank()
        , axis.title.x = element_blank()
        , axis.text.x=element_blank()
        , axis.ticks.x=element_blank()
        , legend.spacing.y = unit(0.01, 'cm')
        , legend.direction = "horizontal"
        , legend.background = element_rect(fill='transparent')
        , legend.key.size = unit(0.6, 'cm')
        , legend.key.height = unit(0.05, 'cm')
        , legend.spacing.x = unit(0.01, 'cm')
        , axis.text = element_text(size = t.size)
        , axis.title = element_text(size = 1.5*l.size)
        , panel.grid.major = element_blank(), panel.grid.minor = element_blank()
  )+
  ylim(-2,2)+ylab("u(x,t)")+ xlab("")+
  guides(colour = guide_legend(nrow = 1))

pl_Unbiased_pred_F=ggplot()+
  geom_line(data=df_pred_time,aes(x=x, y=Fmean, linetype = "mean"),
            colour = cols[1], size=size.line)+
  geom_line(data=df_pred_time,aes(x=x, y=Ftrue, linetype = "true"),
            colour = cols[1], size=size.line)+
  geom_ribbon(data=df_pred_time,
              aes(ymin=F_low, ymax=F_upper, x=x, fill = "95% CI"),
              alpha = 0.3)+
  scale_fill_manual("",values=c("95% CI" = "grey12"))+
  facet_wrap(~time_sec, nrow = 1) +
  theme_bw()+
  theme(legend.position = "bottom"
        , legend.title = element_blank()
        , legend.spacing.y = unit(0.01, 'cm')
        , legend.direction = "horizontal"
        , legend.background = element_rect(fill='transparent')
        , legend.key.size = unit(0.6, 'cm')
        , legend.key.height = unit(0.05, 'cm')
        , legend.spacing.x = unit(0.01, 'cm')
        , axis.text = element_text(size = t.size)
        , axis.title = element_text(size = 1.5*l.size)
        , panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()
        ,strip.background = element_blank()
        , strip.text = element_blank()
        ,axis.title.y = element_text(
          margin = margin(t = 0, r = -0.3, b = 0, l = 0))
  )+
  ylab("f(x,t)")+
  guides(colour = guide_legend(nrow = 1))


common_leg=ggplot()+
  geom_line(data=df_pred_time,aes(x=x, y=Fmean, linetype = "mean"),
            colour = "black", size=size.line)+
  geom_line(data=df_pred_time,aes(x=x, y=Ftrue, linetype = "true"),
            colour = "black", size=size.line)+
  geom_ribbon(data=df_pred_time,
              aes(ymin=F_low, ymax=F_upper, x=x, fill = "95% CI"),
              alpha = 0.3)+
  scale_fill_manual("",values=c("90% CI" = "grey12"))+
  facet_wrap(~time_sec, nrow = 1) +
  theme_bw()+
  theme(legend.position = "bottom", legend.title = element_blank()
        , legend.spacing.y = unit(0.01, 'cm')
        , legend.direction = "horizontal"
        , legend.background = element_rect(fill='transparent')
        , legend.key.size = unit(0.6, 'cm')
        , legend.key.height = unit(0.05, 'cm')
        , legend.spacing.x = unit(0.01, 'cm')
  )+
  guides(colour = guide_legend(nrow = 1))
(pl_Unbiased_pred=ggarrange(pl_Unbiased_pred_U,
                            pl_Unbiased_pred_F, ncol=1
                            , legend = "bottom"
                            , common.legend = TRUE
                            , legend.grob = get_legend(common_leg, position = "bottom")))
```

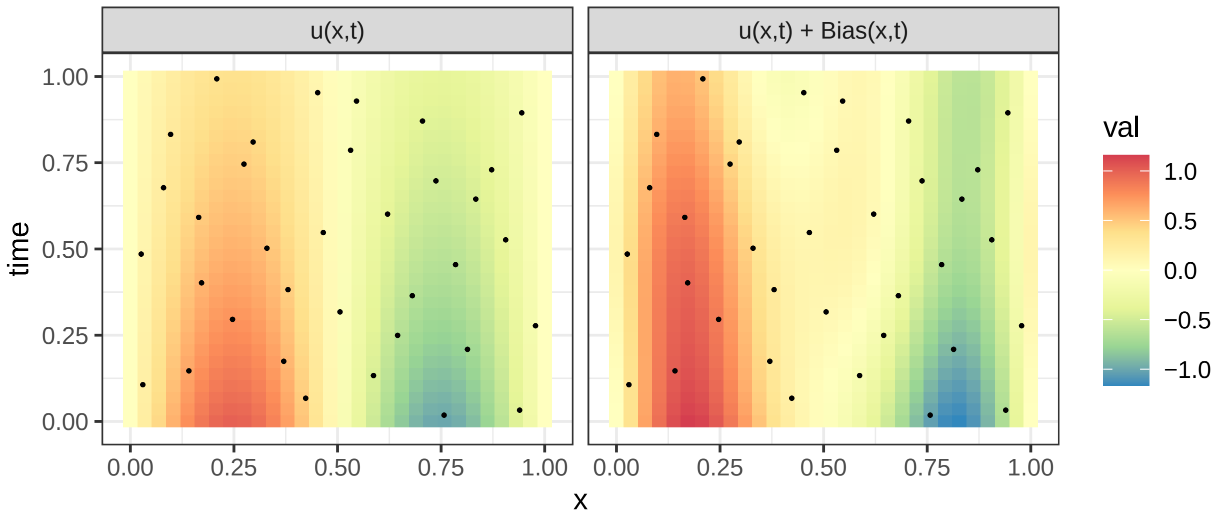


```
# ggsave("Figures/Heat_Unbiased_pred.pdf", plot = pl_Unbiased_pred, width = 18, height = 8, units = "cm
```

Section 4.3 HF Case Study 2: Biased sensor observations

To demonstrate a synthetic case study, we generate bias in the observational process by the following non-linear function, $b(t, x) = \sin(4\pi x)/3 + 2t^2(1-t)^2$. We add this bias to the previously observed data as follows

```
# Bias function
u_biasfn = function(x,t){
  sin(4*pi*(x))/3+2*t^2*(1-t)^2
}

U_Bias = u_biasfn(xtU[,1], xtU[,2])
data_Bias = data_noisy_pred
data_Bias$yU = data_noisy_pred$yU+U_Bias
```



The values of u have now increased in absolute value towards the boundaries of the spatial domain.

We fit two models to the biased data, the model that does not account for biased measurements and is the same as above and the models that accounts for bias in the model formulation as follows

$$
\begin{aligned}
y_u &= u(t,x) + \mathrm{Bias}(t,x) + \varepsilon_u, \text{ where } \mathrm{Bias}(t,x) \sim GP(0, K_{\mathrm{Bias}}((t,x),(t',x'))) \\
y_f &= f(t,x) + \varepsilon_f
\end{aligned}
$$

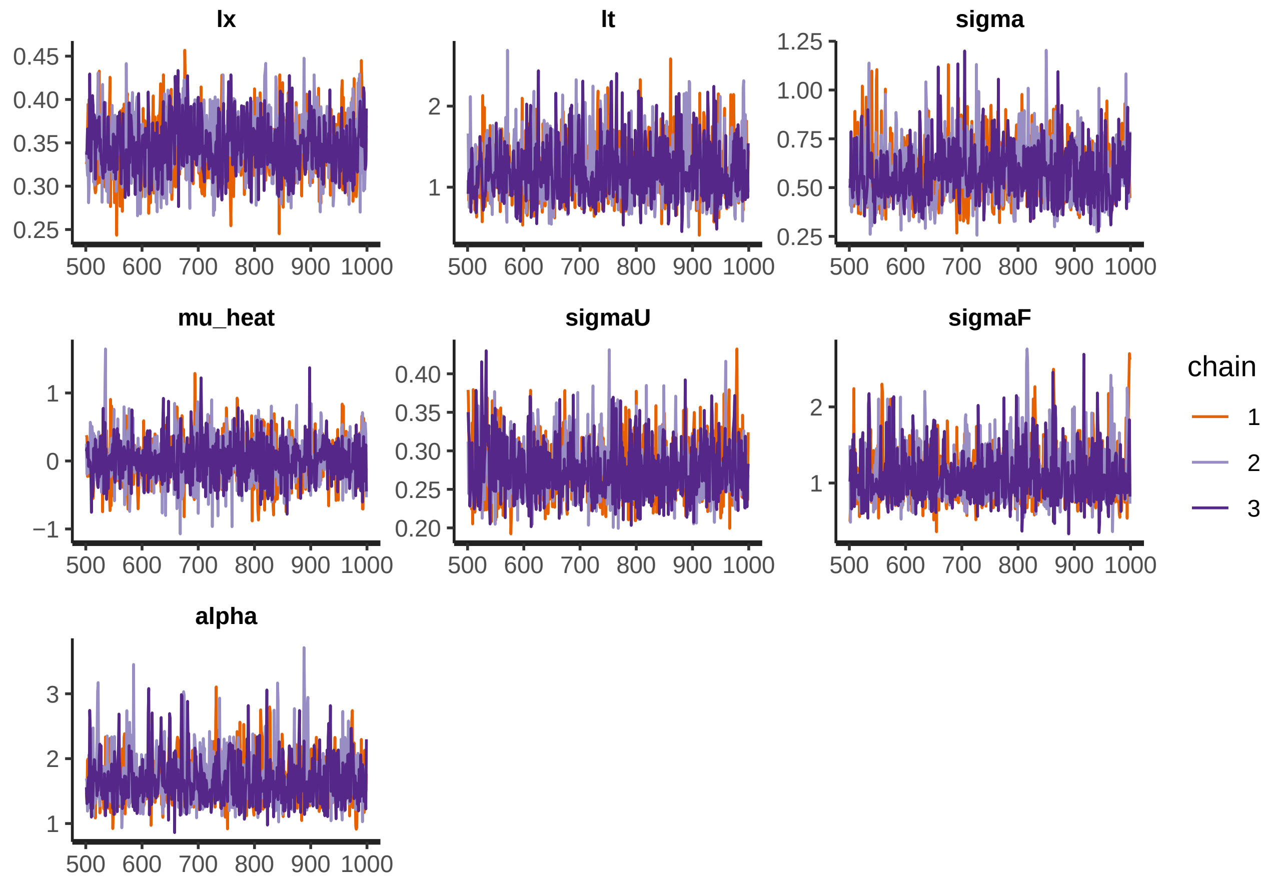
and $K_{\mathrm{Bias}}((t,x),(t',x'))) = \sigma_B^2 \exp\left(-\frac{1}{2lB_t^2}(t-t')^2\right) exp\left(-\frac{1}{2lB_x^2}(x-x')^2\right)$. So we introduce to the model three additional hyper-parameters (σ_B, lB_t and lB_x).

Stan code (select `eval=TRUE` in the code chunk to see the stan code):

```
writeLines(readLines('STAN/Heat_equation/Heat_UBias_PI_prior.stan'))
```

- Fist we fit the model that does not account for bias

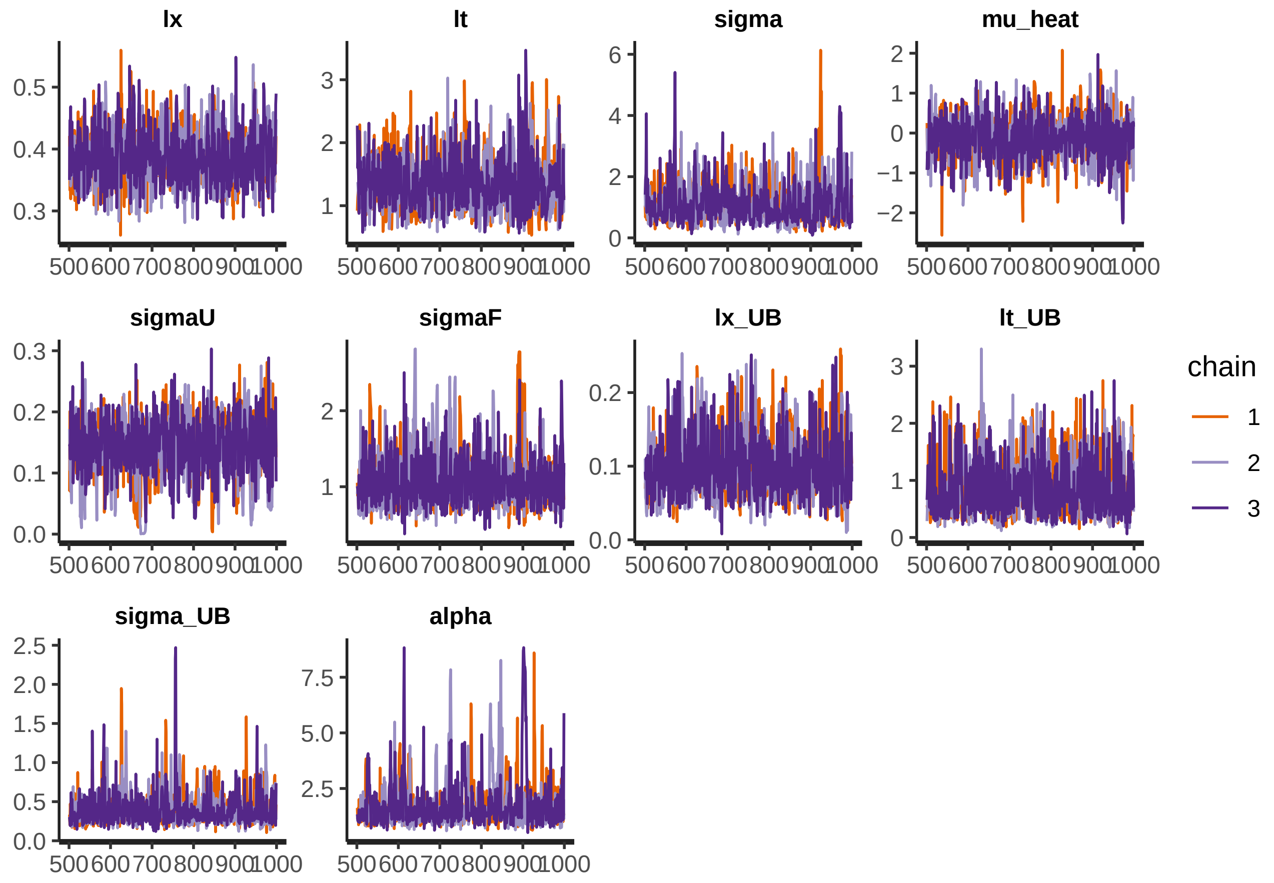
```
fit_noBias = stan(file  = 'STAN/Heat_equation/Heat_PI_prior.stan',
                  data = data_Bias,
                  chains = 3,
                  iter = 1000,
                  seed = 123
)
stan_trace(fit_noBias, pars = names(fit_noBias)[1:7])
```



```
post_noBias = as.data.frame(extract(fit_noBias))
```

- Now we fit the model that accounts for bias

```
fit_UBias = stan(file  = 'STAN/Heat_equation/Heat_UBias_PI_prior.stan',
                 data = data_Bias,
                 chains = 3,
                 iter = 1000,
                 seed = 123
)
stan_trace(fit_UBias, pars = names(fit_UBias)[1:10])
```



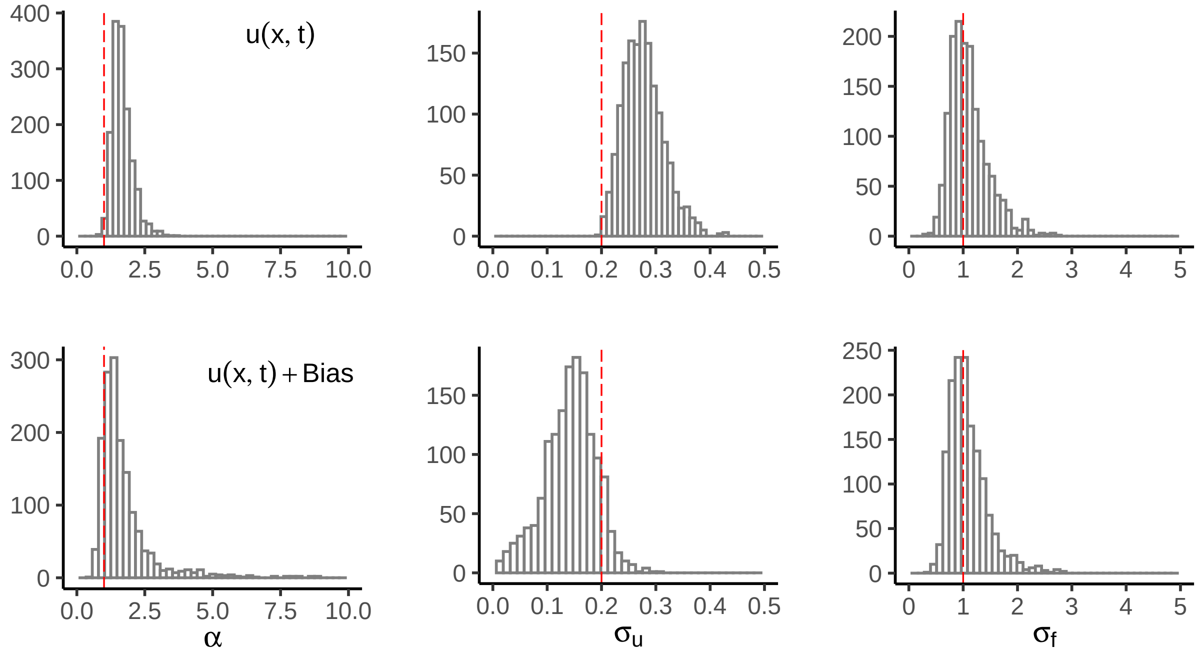
```
post_UBias = as.data.frame(extract(fit_UBias))
```

Posterior distributions of α, σ_u and σ_f (Figure 12 in the paper)

```
pred_true_Bias = pred_true
true_val = data.frame(alpha=1, sd_unoise = 0.2, sd_fnoise=1)
pl_Heat_eq = list( true_val = true_val,
  Unbiased=list(post = post_fitHeat, data=data_noisy_pred,
                pred_true = pred_true)
  , Biased=list(post_noBias=post_noBias,
                post_UBias = post_UBias,
                data=data_Bias,
                pred_true = pred_true_Bias)
  )
#---------------------
post = post_noBias
post$model = "u(x,t)"

post_alpha = post[,c("model", "alpha")]
post_alpha$param = "alpha"

post_sd_noiseU = post[,c("model", "sigmaU")]
post_sd_noiseU$param = "sigmaU"

pl_alpha = ggplot()+
  geom_histogram(data = post, aes(alpha), bins=50,
```

```r
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,10)+
  ylab("")+xlab("")+
  annotate('text', x = 7.5, y = 360, fontface =2,
           label = "u(x,t)",parse = TRUE,size=3.5)

pl_unoise = ggplot()+
  geom_histogram(data = post, aes(sigmaU), bins=50,
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 0.2, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,0.5)+
  ylab("")+xlab("")

pl_fnoise = ggplot()+
  geom_histogram(data = post, aes(sigmaF), bins=50,
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,5)+
  ylab("")+xlab("")
pl_noBias = ggarrange(pl_alpha,pl_unoise,pl_fnoise, nrow=1)

### Bias
post = post_UBias
post$model = "u(x,t)+Bias"
post_alpha = post[,c("model", "alpha")]
post_alpha$param = "alpha"

post_sd_noiseU = post[,c("model", "sigmaU")]
post_sd_noiseU$param = "sigmaU"

pl_alpha = ggplot()+
  geom_histogram(data = post, aes(alpha), bins=45,
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,10)+
  ylab("")+xlab(expression(alpha))+
  annotate('text', x = 7.5, y = 280, fontface =2,
           label = "u(x,t) + Bias",parse = TRUE,
           size=3.5)


pl_unoise = ggplot()+
  geom_histogram(data = post, aes(sigmaU), bins=40,
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 0.2, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,0.5)+
  ylab("")+xlab(expression(sigma[u]))
```

```
pl_fnoise = ggplot()+
  geom_histogram(data = post, aes(sigmaF), bins=45,
                 fill = "white", colour = "grey50")+
  geom_vline(xintercept = 1, linetype = "longdash",
             size=0.3, colour="red")+
  xlim(0,5)+
  ylab("")+xlab(expression(sigma[f]))


pl_UBias = ggarrange(pl_alpha,pl_unoise,pl_fnoise, nrow=1)
post_noBias_UBias = ggarrange(pl_noBias, pl_UBias,ncol=1)
post_noBias_UBias
```



```
# ggsave("Figures/Post_Heat_Biased.pdf", plot = post_noBias_UBias, width = 18, height = 8, units = "cm"
rm(post)
```

$u(t, x)$ and $f(t, x)$ predictions

The predictions can be obtained using the prediction equations in Sections 2.1 and 2.3.

```
# noBias
post_df = post_noBias
cn = colnames(post_df)
Upred = post_df[,grep("y_U",cn)]
Fpred = post_df[,grep("y_F",cn)]
Umean = colMeans(Upred)
UCIs = apply(Upred, 2, quantile, probs = c(0.05,0.95))
Fmean = colMeans(Fpred)
FCIs = apply(Fpred, 2, quantile, probs = c(0.05,0.95))
rm(post_df)
df_pred_time_noBias = data.frame(
  Umean=Umean, Fpred=Fmean
 , Utrue = pred_true$u
 , Ftrue = pred_true$f
 , U_low = UCIs[1,], U_upper = UCIs[2,]
 , F_low = FCIs[1,], F_upper = FCIs[2,]
 , x = pred_true$x, time = pred_true$time
 , model=rep("u(x,t)", length(Umean)))


# Bias
post_df = post_UBias
cn = colnames(post_df)
Upred = post_df[,grep("y_U",cn)]
Fpred = post_df[,grep("y_F",cn)]
Umean = colMeans(Upred)
UCIs = apply(Upred, 2, quantile, probs = c(0.05,0.95))
Fmean = colMeans(Fpred)
FCIs = apply(Fpred, 2, quantile, probs = c(0.05,0.95))
rm(post_df)
df_pred_time_UBias = data.frame(
  Umean=Umean, Fpred=Fmean
  , Utrue = pred_true$u
  , Ftrue = pred_true$f
  , U_low = UCIs[1,], U_upper = UCIs[2,]
  , F_low = FCIs[1,], F_upper = FCIs[2,]
  , x = pred_true$x, time = pred_true$time
  , model=rep("u(x,t) + Bias", length(Umean)))

df_pred_time = rbind(df_pred_time_noBias, df_pred_time_UBias)
uniq_time = unique(df_pred_time$time)
n.unique=length(uniq_time)
ind = c(round(seq(1,n.unique, length.out =  ntimes)),n.unique)
df_pred_time = df_pred_time[df_pred_time$time%in% uniq_time[ind], ]
df_pred_time$time_sec = paste(round(df_pred_time$time,2), " sec", sep = "")
cols = c("#1B9E77","#E41A1C", "#E69F00", "#377EB8")
scaleFUN = function(x) sprintf("%.1f", x)
t.size = 5
l.size = 5
size.line = 0.3
pl=ggplot()+
  geom_line(data=df_pred_time,aes(x=x, y=Umean, linetype = "mean")
            , colour = cols[2], size=size.line)+
  geom_line(data=df_pred_time,aes(x=x, y=Utrue, linetype = "true")
            , colour = cols[2], size=size.line)+
  geom_ribbon(data=df_pred_time,aes(ymin=U_low, ymax=U_upper, x=x, fill = "95% CI")
              , alpha = 0.3)+
  scale_fill_manual("",values=c("90% CI" = "grey12"))+
  facet_grid(model~time_sec)+
  theme_bw()+
  theme(legend.position = "bottom"
        , legend.title = element_blank()
        , legend.spacing.y = unit(0.01, 'cm')
        , legend.direction = "horizontal"
        , legend.background = element_rect(fill='transparent')
        , legend.key.size = unit(0.6, 'cm')
        , legend.key.height = unit(0.05, 'cm')
        , legend.spacing.x = unit(0.01, 'cm')
        , axis.text = element_text(size = t.size)
        , axis.title = element_text(size = 1.5*l.size)
        , panel.grid.major = element_blank()
        , panel.grid.minor = element_blank()
  )+
  ylim(-2,2)+ylab("")+
  guides(colour = guide_legend(nrow = 1))
pl
```

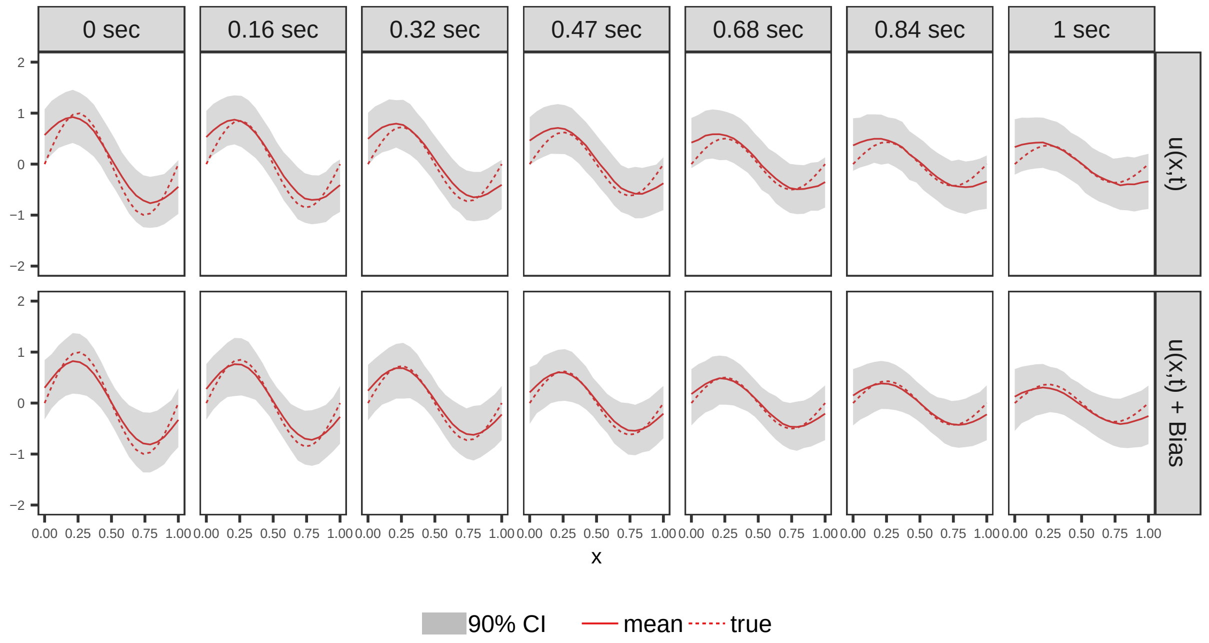
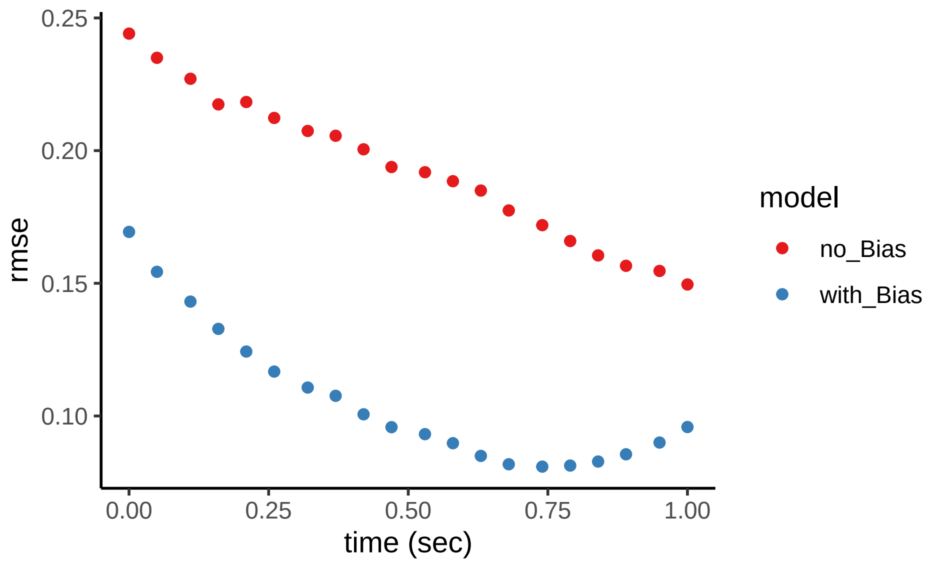


Figure 1: We observe that the model not accounting for Bias does not predict well the heat close to the boundaries of the spatial domain, x. While accounting for bias produce more accurate model predictions.

and the prediction rmses

```
rmse.fn = function(actual, predicted) sqrt(mean((actual - predicted)^2))
rmses = matrix(NA, nrow = 2, ncol = length(uniq_time))
for(j in seq_along(uniq_time)){
  rmses[,j] = c(
    rmse.fn(df_pred_time_noBias$Umean[df_pred_time_noBias$time == uniq_time[j]],
  df_pred_time_noBias$Utrue[df_pred_time_UBias$time == uniq_time[j]]),
  rmse.fn(df_pred_time_UBias$Umean[df_pred_time_UBias$time == uniq_time[j]],
  df_pred_time_UBias$Utrue[df_pred_time_UBias$time == uniq_time[j]])
  )
}
rownames(rmses) = c("no_Bias", "with_Bias")
rmses_df=data.frame(t(rmses))
rmses_df$time=round(uniq_time,2)
rmses_df=melt(rmses_df, id="time")
colnames(rmses_df)[2] = "model"
```

```
# rmses_df$time = as.factor(rmses_df$time)
ggplot()+
  geom_point(data=rmses_df, aes(x=time, y=value, colour=model))+
  ylab("rmse") + xlab("time (sec)")+
  scale_color_brewer(palette="Set1")
```



The code below creates an animation for both models and works for .html but not for .pdf. To run change in the code chunk `eval=FALSE` to `TRUE`.

```
pl_animU.fn = function(post_df,  t.size = 2*5, l.size = 2*7, lt.size = 10, size.line = 0.3, title_U='Mo
  ### Takes as input the posterior distriibution of the stan fit
  library(gganimate)
  cn = colnames(post_df)
  Upred = post_df[,grep("y_U",cn)]
  Fpred = post_df[,grep("y_F",cn)]

  Umean = colMeans(Upred)
  UCIs = apply(Upred, 2, quantile, probs = c(0.05,0.95))

  Fmean = colMeans(Fpred)
  FCIs = apply(Fpred, 2, quantile, probs = c(0.05,0.95))

  df_pred_time = data.frame(Umean=Umean, Fpred=Fmean
                          , Utrue = pred_true$u
                          , Ftrue = pred_true$f
                          , U_low = UCIs[1,], U_upper = UCIs[2,]
                          , F_low = FCIs[1,], F_upper = FCIs[2,]
                          , x = xtU_pred$Var1, time = xtU_pred$Var2)

  uniq_time = unique(df_pred_time$time)

  cols = c("#1B9E77","#E41A1C", "#E69F00", "#377EB8")
  scaleFUN = function(x) sprintf("%.1f", x)
  plU=ggplot()+
    geom_line(data=df_pred_time,aes(x=x, y=Umean, linetype = "mean"), colour = cols[2], size=size.line)
    geom_line(data=df_pred_time,aes(x=x, y=Utrue, linetype = "true"), colour = cols[2], size=size.line)
    geom_ribbon(data=df_pred_time,aes(ymin=U_low, ymax=U_upper, x=x, fill = "90% CI"), alpha = 0.3)+
    scale_fill_manual("",values=c("90% CI" = "grey12"))+
    theme(legend.position = c(0.8, 0.8)
          , legend.title = element_blank()
          , legend.spacing.y = unit(3*0.01, 'cm')
          , legend.direction = "horizontal"
          , legend.background = element_rect(fill='transparent')
          , legend.key.size = unit(3*0.3, 'cm')
          , legend.spacing.x = unit(3*0.01, 'cm')
          , legend.text = element_text(size = lt.size)
          , axis.text = element_text(size = t.size)
          , axis.title = element_text(size = l.size)
          , plot.title = element_text(size=10)
          , plot.subtitle = element_text(size=7)
    )+
    ylab("u") + ylim(-3,3)+
    guides(colour = guide_legend(nrow = 1))

  plU_anim= plU+
    labs(title = title_U,
         subtitle = 'Heat distribution at {round(frame_time,2)} sec.', x = 'x', y = 'u') +
    transition_time(time) +
    ease_aes("linear")+
    enter_fade() +
    exit_fade()
  return(plU_anim)
}


anim_noBias=pl_animU.fn(post_df = post_noBias,  t.size = 2*5, l.size = 2*7, lt.size = 10, size.line = 0
anim_UBias=pl_animU.fn(post_df = post_UBias,  t.size = 2*5, l.size = 2*7, lt.size = 10, size.line = 0.3
```

```
anim_noBias
anim_UBias
```

Total run time

```
Time difference of 6.221286 mins
```

Session information

```
sessionInfo()
```

```
R version 4.0.3 (2020-10-10)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
```

```
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] reshape2_1.4.4       ggpubr_0.4.0         lhs_1.1.3
[4] rstan_2.21.3         ggplot2_3.3.5        StanHeaders_2.21.0-7

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.8          tidyr_1.2.0         prettyunits_1.1.1  ps_1.6.0
 [5] assertthat_0.2.1    digest_0.6.29       utf8_1.2.2         R6_2.5.1
 [9] plyr_1.8.6          backports_1.4.1     stats4_4.0.3       evaluate_0.14
[13] pillar_1.7.0        rlang_1.0.0         rstudioapi_0.13    car_3.0-12
[17] callr_3.7.0         rmarkdown_2.11      labeling_0.4.2     stringr_1.4.0
[21] loo_2.4.1           munsell_0.5.0       broom_0.7.12       compiler_4.0.3
[25] xfun_0.29           pkgconfig_2.0.3     pkgbuild_1.3.1     htmltools_0.5.2
[29] tidyselect_1.1.1    tibble_3.1.6        gridExtra_2.3      codetools_0.2-18
[33] matrixStats_0.61.0 fansi_1.0.2         crayon_1.4.2       dplyr_1.0.7
[37] withr_2.4.3         grid_4.0.3          gtable_0.3.0       lifecycle_1.0.1
[41] DBI_1.1.2           magrittr_2.0.2      scales_1.1.1       RcppParallel_5.1.5
[45] cli_3.1.1           stringi_1.7.6       carData_3.0-5      farver_2.1.0
[49] ggsignif_0.6.3      ellipsis_0.3.2      generics_0.1.2     vctrs_0.3.8
[53] cowplot_1.1.1       RColorBrewer_1.1-2 tools_4.0.3        glue_1.6.1
[57] purrr_0.3.4         processx_3.5.2      abind_1.4-5        parallel_4.0.3
[61] fastmap_1.1.0       yaml_2.2.2          inline_0.3.19      colorspace_2.0-2
[65] rstatix_0.7.0       knitr_1.37
```