Supplementary Material of article "Bayesian uncertainty analysis of inversion models applied to the inference of thermal properties of walls"

Séverine Demeyer et.al

2021

Contents

1 R codes

1.1 functions.R

```
MH<- function(n_sim,pars.init,scale,par_names){

  THETA<- matrix(0,nrow=(n_sim+1),ncol=length(pars.init))
  THETA[1,]<- pars.init
  p.val <- rep(NA, n_sim)
  p.val<- log.vrais(THETA[1,],Tsi)

  k<-0

  for(i in 2:(n_sim+1)){
    theta_c<- mvrnorm(1,THETA[i-1,],scale) #proposal
    p.val.prop<- log.vrais(theta_c,Tsi)
    tmp<- min(1, exp(p.val.prop - p.val[i - 1]))
    print(paste("iter=",i, " tmp=",tmp))
    u<- runif(1,0,1)
    if (!is.finite(tmp)) {tmp <- 0}
    if(u<tmp){ #accept
      THETA[i,]<- theta_c
      p.val[i]<- p.val.prop
      k<- k+1
```

```
    }
    if(u>=tmp){#reject
      THETA[i,]<-THETA[i-1,]
      p.val[i]<- p.val[i - 1]
    }
  }
  accept.rate<- k/n_sim


  return(list(THETA=THETA,p.val=p.val,accept.rate=accept.rate,par_names=par_names))
}

addMHsimulations<- function(MH_object,n_update,scale,par_names){
  end<-dim(MH_object$THETA)[1]
  pars.init<- MH_object$THETA[end,]
  samp.update <- MH(n_update,pars.init,scale,par_names)
  THETA<- rbind(MH_object$THETA,samp.update$THETA)
  p.val<- c(MH_object$p.val,samp.update$p.val)
  accept.rate <- (end * MH_object$accept.rate +
  n_update*samp.update$accept.rate ) / (end + n_update )

  return(list(THETA=THETA,p.val=p.val,accept.rate=accept.rate,par_names=par_names))
}

traceMH<-function (mcmc_object)
{
    num_params <- length(mcmc_object$par_names)
    par(mfrow = c(3,4))
    if (num_params > 1) {
        for (i in 1:num_params) {
            par <- mcmc_object$par_names[i]
            plot(mcmc_object$THETA[, i], type = "l", main =
            paste("Trace of ",
                par), xlab = "Iteration", ylab = par)
        }
    }
}

histMH<- function(mcmc_object){
    num_params <- length(mcmc_object$par_names)
    par(mfrow = c(3,4))
    if (num_params > 1) {
        for (i in 1:num_params) {
            par <- mcmc_object$par_names[i]
            hist(mcmc_object$THETA[, i], breaks = 30, probability
            = TRUE,
```

```
                xlab = par, ylab = "Posterior Density", main =
                 ↪  paste("Posterior distribution of ",
                  par))
        }
    }
}

shortestInt<- function(samples, p){
  y<- samples
  M<- length(y)
  q<- floor(p*M+1/2)
  r<-1:(M-q)
  yOrd=sort(y)
  intervalles<- cbind(yOrd[r],yOrd[q+r],yOrd[q+r]-yOrd[r])
  r_star<-which(intervalles[,3]==min(intervalles[,3]))
  return(intervalles[r_star,c(1,2)])
}

#Thermal resistance
post_trait<- function(TH_c.mcmc,burn_in=1000,end,thinning=30,
 ↪  intervalle, nb_breaks){
  out.thin6<- TH_c.mcmc[seq(burn_in,end,by = thinning),]
  Resistance<- L1/out.thin6[,5] + L2/out.thin6[,6] +
   ↪  L3/out.thin6[,7] + L4/out.thin6[,8]
  k2<- out.thin6[,6]
  return(list(val=Resistance,R.mean=mean(Resistance),
   ↪  R.sd=sd(Resistance), R.CI=shortestInt(Resistance,0.95),
              val_k2=k2, k2.mean=mean(k2), k2.sd=sd(k2),
               ↪  CI.k2=shortestInt(k2,0.95)))
}
```

1.2 mainMCMC.R

```
--
title: "ProtoLNE"
output:
  html_document: default
---


```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
#change to your directory
setwd("C:/Users/demeyer/Desktop/RESBATI/Proto_REBECCA/Code_article")
source('C:/Users/demeyer/Desktop/RESBATI/Proto_REBECCA/thermal_model.R',
 ↪  echo=TRUE)
```
```

```
source('C:/Users/demeyer/Desktop/RESBATI/Proto_REBECCA/functions.R', echo=TRUE)
library(mvtnorm) #multivariate student
library(coda)
library(ggplot2)
library(Hmisc)
library(MASS)

#wrapper for the thermal simulator
#theta=(cw1,cw2,cw3,cw4,k1,k2,k3,k4)
#X=(Tse,vectqint)
F_fun<- function(theta, Tse, vectqint, indice){#theta en dimension 8
  res<-
  modele_thermique_BCG(theta[1],theta[2],theta[3],theta[4],theta[5],theta[6],theta[7],thet

  dx,ndx,L1,L2,L3,L4,dtze,ndtze,vectqint,Tinie,Tse)
  if(indice==1)
  return(list(Tsim_si=res[1,],Tsim_se=res[(ndx+1),]))
  if(indice==2)
  return(res)
}

#layer thickness (m)
L1 = 0.013  #plasterboard
L2 = 0.118  #insulation layer (EPS)
L3 = 0.15   #cinderblock
L4 = 0.015  #exterior coating
Le = L1+L2+L3+L4 # =0.298
dx = 0.002 #Le/ndx : size of discretization in the wall 0.002 (m)
ndx = Le/dx # number of elements of the discretization of the wall

#read initial dataset from .csv file
DON<-read.table(file ="Data_03062019.csv",header = T,sep=";")
```

#Choice of duration for a time step (dtze) of 300 seconds

```{r}
#Choice of duration 12h or 24h
choice = "24h"
if(choice=="12h") temps_simus<- 43200
if(choice=="24h") temps_simus<- 86400

dtze = 300 #time step (dt=300s)
```

```
ndtze = temps_simus/dtze   #number of time steps for the simulation, temps_simus chosen as a multiple of dtze

Tinie1 = DON$TSI[1] # initial temperature in the wall on the internal face (?C)
Tiniendxp1 = DON$TSE[1] # initial temperature in the wall on the external face (?C)
# vector of initial temperatures in the wall at t=0s
Tinie =seq(Tinie1,Tiniendxp1,(Tiniendxp1-Tinie1)/ndx)

#Dataset for analysis
DON1<- DON[seq(1,which(DON$t.s==temps_simus),by=150),] #remember that data are recorded every 2s in the initial dataset
N<- dim(DON1)[1] #number of measurements for analysis

#internal surface measurements
vectqint = DON1$phy_abs
Tsi<- DON1$TSI

#external surface measurements, solar flux set to zero
Tse<- DON1$TSE

#Uncertainty level: low / medium (recommended)
u_level<- "medium"
if(u_level=="low"){u_Tsi=u_Tse=0.1 ; u_qint<- 0.01*vectqint}
if(u_level=="medium"){u_Tsi=u_Tse=0.5 ; u_qint<- 0.03*vectqint}
```

#Bounds for uniform distributions

```{r}
cw1<- c(7e5,8e5)
cw2<- c(1e4,3e4)
cw3<- c(8.5e5,2e6)
cw4<- c(1e6,2e6)
k1<- c(0.2,0.4)
k2<- c(0.02,0.04)
k3<- c(0.7,1.2)
k4<- c(0.5,1.2)
```

#Plot prior distribution of the thermal resistance

```{r}
Nsim_p<- 50000
Rprior<- L1/runif(Nsim_p,k1[1],k1[2])+L2/runif(Nsim_p,k2[1],k2[2])+L3/runif(Nsim_p,k3[1],k3[2])+I
out.prior<- density(Rprior)
plot(out.prior)
```

```

#Log-posterior
```{r}
log.prior<- function(pars){
  s02<- 2
  nu0<- 2
  Xsim<- pars[c(9:(8+N))]
  Qsim<- pars[(9+N):(length(pars))]
  log_prior<- dunif(pars[1],cw1[1],cw1[2],log=T) +
  dunif(pars[2],cw2[1],cw2[2],log=T)+
  dunif(pars[3],cw3[1],cw3[2],log=T)+
  dunif(pars[4],cw4[1],cw4[2],log=T) +
  dunif(pars[5],k1[1],k1[2],log=T) +
  dunif(pars[6],k2[1],k2[2],log=T) +
  dunif(pars[7],k3[1],k3[2],log=T) +
  dunif(pars[8],k4[1],k4[2],log=T)+
  dmvnorm(Xsim, Tse, diag(u_Tse^2,N),log=T) +
  dmvnorm(Qsim, vectqint, diag((u_qint)^2),log=T)
  return(log_prior)
}

log.vrais<- function(pars,data){
  s02<- 2
  nu0<- 2
  pred<- F_fun(pars[c(1:8)], pars[c(9:(8+N))],
  pars[(9+N):length(pars)], indice=1)$Tsim_si
  log_prior<- log.prior(pars)
  varcov_y<- diag(u_Tsi^2,N)
  log_likelihood<-  dmvt(data, delta = pred, sigma =
  s02*varcov_y, df = nu0, log = TRUE)
  return(log_likelihood + log_prior)
}

```

#MH algorithm
```{r}
n_sim<-50000 #number of MCMC simulations
par_names=c('cw1','cw2','cw3','cw4','k1','k2','k3','k4',paste("X",seq(1,length(Tse),1),
  sep="."),paste("Q",seq(1,length(vectqint),1), sep="."))
  #vector of parameters names
sig2.cw1<- 1000^2 ;sig2.cw2<- 100^2 ; sig2.cw3<- 1000^2 ;
  sig2.cw4<- 1000^2
sig2.k1<- 0.001^2 ; sig2.k2<- 0.0003^2 ; sig2.k3<- 0.001^2 ;
  sig2.k4<- 0.001^2
```
```

```
sig2.X<- 0.01^2 ; sig2.Q<- 0.01^2
#set the covariance matrix of the multivariate proposal distribution
scale<-
 diag(c(sig2.cw1,sig2.cw2,sig2.cw3,sig2.cw4,sig2.k1,sig2.k2,sig2.k3,sig2.k4,

 rep(sig2.X,length(Tse)),rep(sig2.Q,length(vectqint))))
#set initial values of parameters
Tse.init<- mvrnorm(1,Tse,diag((0.01*Tse/sqrt(3))^2))
Q.init<- mvrnorm(1,vectqint,diag((0.01*vectqint/sqrt(3))^2))
pars.init<- c(7.517902e+05, 2.062815e+04, 1.655038e+06,
 1.195574e+06, 0.3, 3.312235e-02, 0.950, 0.6, Tse.init,Q.init)

#run the Metropolis-Hastings algorithm
out.MH<- MH(n_sim,pars.init,scale,par_names=par_names)
str(out.MH)#print the structure of out.MH object containing the elements of the Markov chains generated by the MCMC algorithm

accept.rate<- out.MH$accept.rate #acceptance rate, ideally falls between 15% and 35%
print(paste("Acceptance rate =" , 100*accept.rate,"%"))
out.pt<- post_trait(out.MH$THETA,10000,50000,10,c(3.9,4.2),20) #post treatment of MCMC chains to get posterior point estimates
print(paste("R=",out.pt$R.mean,"u(R)=",out.pt$R.sd,"R 95% CI =","[",out.pt$R.CI[1],",", out.pt$R.CI[2],"]"))
print(paste("k2=",out.pt$k2.mean,"u(k2)=",out.pt$k2.sd,"k2 95% CI =","[",out.pt$CI.k2[1],",",out.pt$CI.k2[2],"]"))
```

1.3 thermalModel.R

```
modele_thermique_BCG<-function(cw1,cw2,cw3,cw4,k1,k2,k3,k4,

                               dx,ndx,Le1,Le2,Le3,Le4,dtze,ndtze,vectqint,Tinie,Tse)
{

  # Construction of matrices C and K
  ###################################

  #Elementary matrix matCelem
  Mtemp1<-matrix(c(1/3,1/6,1/6,1/3),nrow=2)
  matCelem1 = cw1*dx*Mtemp1
  matCelem2 = cw2*dx*Mtemp1
  matCelem3 = cw3*dx*Mtemp1
  matCelem4 = cw4*dx*Mtemp1
```

```
14
15
16    #Elementary matrix matKelem
17    Mtemp2<- matrix(c(1,-1,-1,1),nrow=2)
18    matKelem1 = k1/dx*Mtemp2
19    matKelem2 = k2/dx*Mtemp2
20    matKelem3 = k3/dx*Mtemp2
21    matKelem4 = k4/dx*Mtemp2
22
23    #Matrix Cww associated with the enveloppe
24    matCww = matrix(0,nrow=ndx+1,ncol=ndx+1)
25    matCww1 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 1
26    matCww2 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 2
27    matCww3 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 3
28    matCww4 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 4
29
30
31    #Matrix Kww associated with the enveloppe
32    matKww =  matrix(0,nrow=ndx+1,ncol=ndx+1)
33    matKww1 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 1
34    matKww2 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 2
35    matKww3 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 3
36    matKww4 = matrix(0,nrow=ndx+1,ncol=ndx+1) # associe a couche 4
37
38    #Preliminary calculus of the assembly
39    ndx1 = Le1/dx # number of elements in layer 1
40    ndx2 = Le2/dx # number of elements in layer 2
41    ndx3 = Le3/dx # number of elements in layer 3
42    ndx4 = Le4/dx # number of elements in layer 4
43
44
45    #Assembly
46    i=1 # initialisation
47
48    for(j in 1:ndx1)# layer 1
49    {
50      matCww1[i:(i+1),i:(i+1)] = matCww1[i:(i+1),i:(i+1)] +
        ↪  matCelem1 ;
51      matKww1[i:(i+1),i:(i+1)] = matKww1[i:(i+1),i:(i+1)] +
        ↪  matKelem1 ;
52      i=i+1
53    }
54
55    for(j in 1:ndx2)# layer 2
56    {
```

```
57      matCww2[i:(i+1),i:(i+1)] = matCww2[i:(i+1),i:(i+1)] +
        ↪  matCelem2 ;
58      matKww2[i:(i+1),i:(i+1)] = matKww2[i:(i+1),i:(i+1)] +
        ↪  matKelem2 ;
59      i=i+1
60    }
61
62    for(j in 1:ndx3)# layer 3
63    {
64      matCww3[i:(i+1),i:(i+1)] = matCww3[i:(i+1),i:(i+1)] +
        ↪  matCelem3 ;
65      matKww3[i:(i+1),i:(i+1)] = matKww3[i:(i+1),i:(i+1)] +
        ↪  matKelem3 ;
66      i=i+1
67    }
68
69    for(j in 1:ndx4)# layer 4
70    {
71      matCww4[i:(i+1),i:(i+1)] = matCww4[i:(i+1),i:(i+1)] +
        ↪  matCelem4 ;
72      matKww4[i:(i+1),i:(i+1)] = matKww4[i:(i+1),i:(i+1)] +
        ↪  matKelem4 ;
73      i=i+1
74    }
75
76    matCww = matCww + matCww1 + matCww2 + matCww3 + matCww4 #
      ↪  matrix heat capacity enveloppe
77    matKww = matKww + matKww1 + matKww2 + matKww3 + matKww4 #
      ↪  matrix thermal conductivity enveloppe
78
79    # Matrix C
80    matCcd = matCww # ndx+1 unknown in the enveloppe
81
82    # Matrix K
83    matKcd = matKww #ndx+1 unknown in the enveloppe
84
85    # Construction of matrix A
86    # ndx+1 unknown in the enveloppe
87    matAcd = (1/dtze)*matCcd + matKcd
88
89    matBlag<- matrix(0,nrow=1,ncol=ndx+1)
90    matBlag[1,ndx+1]<- 1
91
92    matAtot<- matrix(0,nrow=ndx+2,ncol=ndx+2)
93    matAtot[1:(ndx+1),1:(ndx+1)] <- matAcd
```

```
94    matAtot[(ndx+2),1:(ndx+1)] <- matBlag # for Dirichlet limit
      ↪  conditions for x=L
95    matAtot[1:(ndx+1),(ndx+2)] <- matBlag
96
97    #.....................................
98    # Resolution at each time step
99    #.....................................
100
101   # Matrix space-time of the temperature in the enveloppe
102   matTcd = matrix(0,nrow=ndx+1,ncol=ndtze+1) #ndx+1 points,
      ↪  ndtze+1 times
103
104   # vector of temperatures at t_n and t_n-1
105   vectTn = rep(0,ndx+1) # ndx+1 unknown in the enveloppe
106   vectTnm1 = rep(0,ndx+1) # ndx+1 unknown in the enveloppe
107
108
109   # CASE n=1 : initial condition
110   #..............................
111
112   # Time : n=1
113   vectTn[1:(ndx+1)] = Tinie[1:(ndx+1)] # initial condition in the
      ↪  enveloppe
114
115   # Storing result at n=1
116   matTcd[,1] = vectTn ;
117
118   # CASE n>1 :
119   #.................
120   vectFn = rep(0,ndx+1) # vector thermal load at instant n
121   vectBnnm1lag = rep(0,ndx+2) # second member of the thermal
      ↪  problem
122
123
124   for(n in 2:(ndtze+1)) # time steps
125   {
126     # temperature at instant n-1
127     vectTnm1 = vectTn ;
128
129     # thermal load at instant n
130     vectFn[1] = vectqint[n] # imposed flux
131
132     vectBnnm1lag[1:(ndx+1)] = (1/dtze)*matCcd%*%vectTnm1 + vectFn
        ↪  # second member
133     vectBnnm1lag[ndx+2]<- Tse[n]
134
```

```
135       # resolution of linear system
136       vectTnlag = solve(matAtot,vectBnnm1lag)
137       vectTn<- vectTnlag[1:(ndx+1)]
138
139
140       # Storing results at n
141       matTcd[,n] = vectTn[1:(ndx+1)]
142
143     }
144
145     return(Temp=matTcd)
146
147   }
```