NUFEB 2.0: Gaussian process emulation procedure

Oluwole K. Oyebamiji

May 23, 2018

Contents

List of Tables

1. Introduction

The simulation of microbial communities has important application in wastewater treatment studies. Wastewater treatment plants are open systems that depend on many species of bacteria to form a microbial community for the transformation of waste into biomass [1]. To develop the tools to predict how bacteria grow, flow and change in a large-scale system such as wastewater treatment plants (WWTPs), there is a need to understand the interactions of microbes at a microscale level. In this study, the NUFEB 2.0 simulator was developed to study the growth and interactions of microbes at a fine scale using an individual-based modelling technique.

This model is computationally expensive, and due to computing constraints, a limited set of scenarios is often possible. The model can run up to order 10^7 bacteria, and we would like to scale up to 10^{18} bacteria. Emulators offer rapid and relatively quick alternatives for projection of model outputs from expensive computer models. An emulator is a statistical representation of an expensive computer model that gives fast probabilistic predictions of the output. In other words, emulation is a statistical technique for simplifying models that lead to reduced-form representations of complex models which are computationally much faster to run. A further benefit of emulation is the provision of a measure of uncertainty associated with the projections [2, 3].

The emulators describe in this document are based on the Gaussian process regression. A Gaussian process is a distribution over functions, and it is fully specified by a mean function $\boldsymbol{\mu}$ and a covariance function $\mathbf{K}$ such that $f \sim GP(\boldsymbol{\mu}, \mathbf{K})$. The points that are close together in input space are more highly correlated than points far away. There have been a significant number of research applications dealing with the statistical emulation of expensive computer models. This ranges from a univariate Gaussian process emulation to multi-output predictions [4]. The emulator can enhance our understanding of the common frameworks for theoretical modelling of multiscale problems. Therefore, emulators can serve as tools for coupling scales in multi-scale modelling. Here, we describe the process for emulating an IB model simulation outputs that represent different kinds of quantities. We use a multivariate GP emulator that models the outputs jointly using a separable covariance structure due to its mathematical tractability that simplifies our estimation procedure.

We give detail procedure for emulating essential outputs from the NUFEB 2.0 model such as the rate of change of nutrient concentrations and important derivatives of various biofilm components given its current biomass composition and environmental conditions. A multivariate GP emulator based on the simulation ensembles from the NUFEB 2.0 model is used in this study. The simulator modelled the interaction of microbes at a microscale while the emulator was built to transfer this microscale information to mesoscale process in a computationally efficient way. The emulators produced are designed to upscale and pass the fine-scale emerging properties to mesoscale. For instance, to develop a computationally intensive continuum model of a bench-scale experiment (where we have a rectangular channel of dimension $\sim 2cm \times 40cm$), many rates of change and relevant derivatives are often required, and this can be provided by emulators trained from the NUFEB 2.0 simulation data.

In training the emulator, the current biofilms mass composition and environmental conditions (e.g. nutrient concentrations) can be considered as the inputs to the emulator, and then the emulators can be constructed that predict the growth rate of each species in the biofilm and rates of change of nutrient concentrations etc. as the real outputs. The idea is to represent the simulator by an unknown function f that gives a matrix of outputs $\mathbf{Y}$ with an input matrix $\mathbf{X}$ such that $\mathbf{Y} = f(\mathbf{X})$. We wish to emulate this unknown function $f(.)$ that can predict the values of $\mathbf{Y}$ given input $\mathbf{X}$ as accurately as possible. To achieve this, we can choose a linear function of this form $f = H(.)B$, where $B = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_p]$ is a vector of unknown regression coefficients and $H(x) = [h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})]$ is a matrix of regression functions. See further details in section 3.

2. Simulation data

We describe the emulation of a Monod-based growth experiment in this study. The NUFEB 2.0 model Jayathilake et al. [5] was run with three functional groups of microorganism and two inert states to train the emulator. The microorganisms are heterotrophs (HET) which consume organic carbon source and oxygen, ammonia oxidizing bacteria (AOB) which convert ammonia and oxygen to nitrite, and nitrite oxidizing bacteria (NOB) which use nitrite and oxygen to produce nitrate. For the inert states, extracellular polymeric substance (EPS), secreted by some heterotrophs and dead agents are represented by soft spheres (labelled DEAD). Five different substrates namely carbon substrate (Sub), Oxygen (O2), Ammonia (NH4), Nitrite (NO2), and Nitrate (NO3) are considered. See further details in [5, 6] and https://github.com/shelllbw/nufeb.

We ran the IB model for a small sample of input parameters (see Table 1) which are generated using a Latin Hypercube Sampling (LHS). The LHS technique provides good coverage of the input space with a relatively small number of design points. We use the ”maximin” version of the LHS technique that optimises samples by maximising the minimum distance between design points. We generated an $n \times p$ variables Latin Hypercube sample matrix with values uniformly distributed on the interval [0,1]. We transform the generated sample to the quantile of a uniform distribution. The 11 model parameters are varied within the range of ±50% of the standard values given in Table 1 to cover a wide variety of the computer model outputs behaviour [7, 3]. We ran our simulation at $n = 200$ training points, and five replicates at each design point because of the expense of this computer model. The design matrix $\mathbf{X}_{200\times11}$ denotes the input values at which the NUFEB 2.0 model is run for every combination of $\mathbf{x}_i$. The other simulation parameters are held constant in Table 2. The simulation box for this experiment was run on a cubic volumes with a length of 200 μm on each side.

Table 1: **IB model parameters**

Index	Parameters	Symbols	Values	Units	References
1	Bulk substrate concentration	Sub	30×10^{-3}	kgm^{-3}	Lardon et al. [8]
2	Oxygen concentration	O_2	10×10^{-3}	kgm^{-3}	Lardon et al. [8]
3	Nitrite concentration	NO_2	10×10^{-3}	kgm^{-3}	-
4	Nitrate concentration	NO_3	10×10^{-3}	kgm^{-3}	-
5	Ammonia concentration	NH_4	6×10^{-3}	$kgNm^{-3}$	Lardon et al. [8]
6	Max. specific growth rate (HET)	$\mu_{m,HET}$	6.944×10^{-5}	s^{-1}	Lardon et al. [8]
7	Max. specific growth rate (AOB)	$\mu_{m,AOB}$	1.158×10^{-5}	s^{-1}	Lardon et al. [8]
8	Max. specific growth rate (NOB)	$\mu_{m,NOB}$	1.158×10^{-5}	s^{-1}	-
9	Yield coefficient (HET)	Y_{HET}	0.63	$gCOD/gCOD$	Lardon et al. [8]
10	Yield coefficient (AOB)	Y_{AOB}	0.24	$gCOD/gCOD$	Lardon et al. [8]
11	Yield coefficient (NOB)	Y_{NOB}	0.24	$gCOD/gCOD$	-
	Simulation box dimension	$L_x \times L_y \times L_z$	$200 \times 200 \times 200$	μm^3	-
	Cartesian grid cells	$N_x \times N_y \times N_z$	$20 \times 20 \times 20$	-	

3. Method for emulation

3.1. Multivariate Gaussian process (GP)

A popular method for constructing a metamodel is Gaussian process regression. GP emulation is based on Bayesian updating and experimental design of computer experiments for predicting model outputs at test input points. A GP emulator assumes that a simulator output is an unknown function $f(.)$ with a given prior distribution for $f(.)$, updated using data obtained from the simulator runs [9, 7]. We shall briefly describe the multivariate GP regression. Suppose we have k outputs $\mathbf{Y}(x) = (Y_1(x), \ldots, Y_k(x))$, which has a joint matrix normal distribution for n simulator runs, such that for any matrix $\mathbf{Y}_{n\times k}$ of outputs, we have

$$\mathbf{Y}|\mathbf{B}, \mathbf{\Sigma}, \mathbf{R} \sim MN(\mathbf{HB}, \mathbf{\Sigma} \otimes \mathbf{A}), \tag{1}$$

where $\mathbf{H}_{n\times m}$ is the model matrix with i^{th} row denoted as $h(.)^T$ and defined as a vector of regression functions. The matrix $\mathbf{B}_{m\times k}$ is a matrix of unknown regression coefficients and $\mathbf{R}_{n\times n}$ is a diagonal matrix of scale parameters. We have assumed a separable covariance structure because it is relatively easy to perform. The assumption of separability of covariance function implies that output variance can be decomposed such that $cov(f(\mathbf{x}), f(\mathbf{x}')) = \mathbf{\Sigma}_Y = \mathbf{\Sigma} \otimes \mathbf{A}$, where $\mathbf{\Sigma}$ is an $k \times k$ positive definite matrix of cross-covariance between the outputs at any input and $\mathbf{A}_{n\times n}$ is a correlation matrix across the input space and $\otimes$ is a kronecker product operator [10]. We use linear mean and exponential correlation functions in this study. We estimate the unknown parameters $\mathbf{B}$, $\mathbf{\Sigma}$ and $\mathbf{R}$ of the mean and covariance functions. These parameters are estimated under a Bayesian framework using a non-informative priors [3]. Let $\mathbf{Y} = f(\mathbf{x}) = h(\mathbf{x})^T\mathbf{B} + \epsilon$, we can obtain the conditional posterior distribution of the computer model $f(.)|\mathbf{B}, \mathbf{\Sigma}, \tilde{\mathbf{R}}, \mathbf{Y}$ often called the emulator such that

$$f(.)|\mathbf{B}, \mathbf{\Sigma}, \mathbf{A}, \mathbf{Y} \propto MN(\boldsymbol{\mu}^*, \mathbf{c}^*(.,.)\mathbf{\Sigma}), \tag{2}$$

We use an exponential correlation function of the form

$$c(\mathbf{x}, \mathbf{x}') = \exp\Big\{-(\mathbf{x}-\mathbf{x}')^T\mathbf{R}(\mathbf{x}-\mathbf{x}')\Big\} + \delta I, \tag{3}$$

where $\mathbf{A}$ is a $(n\times n)$ positive definite matrix of correlations at the experimental design points (i.e $\mathbf{A} = c(\mathbf{x}, \mathbf{x}')$), $\boldsymbol{\mu}^*$ and $\mathbf{c}^*$ are given below.

$$\boldsymbol{\mu}^*(\mathbf{x}) = \mathbf{H}(\mathbf{x})^T\hat{\boldsymbol{B}} + t^T(\mathbf{x})\mathbf{A}^{-1}(\mathbf{Y} - \mathbf{H}\hat{\boldsymbol{B}}), \tag{4}$$

$$\mathbf{c}^*(\mathbf{x}, \mathbf{x}') = c^*(\mathbf{x}, \mathbf{x}') + \Big[h(\mathbf{x}) - \mathbf{H}^T\mathbf{A}^{-1}t(\mathbf{x})\Big]^T(\mathbf{H}\mathbf{A}^{-1}\mathbf{H})^{-1}\Big[h(\mathbf{x}') - \mathbf{H}^T\mathbf{A}^{-1}t(\mathbf{x}')\Big], \tag{5}$$

where

$$\hat{\boldsymbol{B}} = \Big((\mathbf{H}^T\mathbf{A}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{A}^{-1}\mathbf{Y}\Big), \qquad \hat{\mathbf{\Sigma}} = \Big(\frac{(\mathbf{Y}-\mathbf{H}\hat{\boldsymbol{B}})^T\mathbf{A}^{-1}(\mathbf{Y}-\mathbf{H}\hat{\boldsymbol{B}})}{(n-m)}\Big)$$

and $t(\mathbf{x}^{new}) = \Big[c(\mathbf{x}_1, \mathbf{x}^{new}), \ldots, c(\mathbf{x}_n, \mathbf{x}^{new})\Big]^T$ denotes the $(n \times 1)$ vector of cross correlations between the $\mathbf{x}$'s at the design points and new input points $\mathbf{x}^{new}$.

To estimate $\mathbf{R}$ and δ, we use a plug in approach based on a posterior mode of $\tilde{\mathbf{R}}$, derived by maximizing equation (6) below.

$$\pi_{\tilde{\mathbf{R}}}(\tilde{\mathbf{R}}|\mathbf{Y}) \propto \pi_{\tilde{\mathbf{R}}}(\tilde{\mathbf{R}})|\mathbf{A}|^{-k/2}|\mathbf{H}^T\mathbf{A}^{-1}\mathbf{H}|^{-k/2}|\mathbf{Y}^T\mathbf{G}\mathbf{Y}|^{-(n-m)/2}, \tag{6}$$

where $\mathbf{G} = \tilde{\mathbf{R}}^{-1} - \tilde{\mathbf{R}}^{-1}\mathbf{H}(\mathbf{H}^T\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\tilde{\mathbf{R}}^{-1}$, see further details in Oyebamiji et al. [3].

3.2. GP emulation summary

We summarised the detailed procedure for emulating the rate of change of biomass and nutrient concentrations as a function of current nutrient conditions and biomass concentrations using a multivariate GP regression. The main steps taken for the emulation are given here:

(i) Design of an experiment to identify important input parameters to be varied ($p = 11$ in the present study).

(ii) Assign a uniform probability density to each input variable (we have little knowledge about these parameters) - from literature.

(iii) Generate LHS of 200 design points on each of the 11 input parameters and transform into the quantile of uniform distribution using parameter ranges in Table 1. The R scripts "rnumber.R" and "stoch.R" for implementing item $(i - iii)$ are given in the Appendices B and C.

(iv) Evaluate the NUFEB 2.0 model for various input combinations to obtain the training data (including 5 replicates at each sample design points). A shell script "main.sh" to perform this action on a Rocket machine is given in the Appendix D.

(v) Compute the interesting outputs from the simulation ensembles. The simulation model is a time evolving dynamical system therefore we can calculate the rate of change of nutrient concentrations (S, O_2, NO_2, NO_3, NH_4) and biomass concentration for each species of biofilms (HET, AOB, NOB, EPS) such that $\mathbf{Y} = \left[\frac{\partial S}{\partial t}, \frac{\partial O_2}{\partial t}, \frac{\partial NO_2}{\partial t}, \frac{\partial NO_3}{\partial t}, \frac{\partial NH_4}{\partial t}, \frac{\partial HET}{\partial t}, \frac{\partial AOB}{\partial t}, \frac{\partial NOB}{\partial t}, \frac{\partial EPS}{\partial t}\right]$. The corresponding input matrix is denoted as $\tilde{\mathbf{X}}$=[S, O_2, NO_2, NO_3, NH_4, HET, AOB, NOB, EPS]. These represent the current nutrient conditions and biomass concentration. The R script "prep1d.R" for implementing this is given in the Appendix E.

(vi) Scale the input matrices to the range [0, 1] such that

$$\mathbf{X} = \frac{\tilde{\mathbf{X}} - \tilde{\mathbf{X}}_{min}}{\tilde{\mathbf{X}}_{max} - \tilde{\mathbf{X}}_{min}}.$$

(vii) Sample 2900 design points randomly from both $\mathbf{X}$ and $\mathbf{Y}$ matrices as training data and use the left-out as test data for cross-validation.

(viii) Fit GP emulation as described in subsection 3 and use linear mean and exponential covariance functions. The multivariate GP parametes $\Theta = (\mathbf{B}, \mathbf{\Sigma}, \mathbf{R}, \delta)$ are estimated using the Bayesian inference with non informative priors. For further details see Oyebamiji et al. [3].

(ix) Compute the posterior distribution of multivariate GP $\left(f(.)|\mathbf{Y}, \hat{\Theta}\right) \sim N\left(\boldsymbol{\mu}(\mathbf{x}), \mathbf{c}^{*}(\mathbf{x}, \mathbf{x}')\right)$ which the emulator where $\boldsymbol{\mu}(\mathbf{x})$ and $\mathbf{c}^{**}(\mathbf{x}, \mathbf{x}')$ are defined in equations (4, 5) respectively). The R script "emulator.R" for implementing item $(vi - ix)$ is given in the Appendix F.

(x) The final emulator is written in a C^{++} (emu.cpp) code, and the "readme" file that gives full detailed instruction for compiling and running the codes are given in the Appendices G and H respectively.

4. Appendix

The R codes and bash scripts for implementing the GP emulator describe above are given in this section.

4.1. Appendix A: nufeb.sh

```
#Design of experiments
R CMD BATCH rnumber.R res1.txt
#Generate 200 design 5 replicates inputscripts for NUFEB 2.0
for num in {1..200}
do
layi=$num
echo $layi | sed s/"num"/$layi/< main.sh >v$num.sh
chmod +x v$num.sh
done
#
for num in {1..200}
do
for fold in {1..5}
do
layi=$fold
echo $layi | sed s/"fold"/$layi/< v$num.sh >v$num.$fold.sh
chmod +x v$num.$fold.sh
done
done
cp v*.sh run
rm v*.sh
#bash script for submission on Rocket machine
for num in {1..200}
do
for fold in {1..5}
do
sbatch run/v$num.$fold.sh
done
done
#preprocessing the simulation data
R CMD BATCH prep1d.R res2.txt
#train the emulator
R CMD BATCH emulator.R res3.txt
```

4.2. Appendix B: rnumber.R for experimental design

```
nvar=11#number of parameters to be varied
npoints=200#nunber of experimental design points
npoints2=5#number of stochastic repetitions
 val=c("30e-3","10e-3","6e-3","6e-3","6e-3","0.00006944444","0.00001158333",
 "0.00001158333","0.63","0.24","0.24")
 vall=as.numeric(val)
 ppara=c("1␣sub␣l","2␣o2␣l","3␣no2␣l","4␣no3␣l","5␣nh4␣l","het␣0.00006944444",
 "aob␣0.00001158333","nob␣0.00001158333","het␣0.63","aob␣0.24","nob␣0.24")
dirname=getwd()
```

```
#set.seed(3)
library(lhs)
 hyper=maximinLHS(npoints,nvar, 2)
 design= matrix(NA,nrow=npoints, ncol=nvar)
 for(j in 1:nvar){
design[,j]=round(qunif(hyper[,j], min=0.5*vall[j], max=1.5*vall[j]),6)#+-50%
}
text <- readLines("atom.in",encoding="UTF-8")
nrep=npoints
save(design,file=paste(dirname,"design",sep="/"))
id=rep(NA,nvar)
for (i in 1:nvar){
id[i]=grep(ppara[i],text,value=FALSE,fixed=TRUE)
}
 for(rep in 1:nrep){
 for(j in 1:length(val)){
 text[id[j]]=gsub(val[j],design[rep,j],text[id[j]])
 }
 writeLines(text,sub("rep",rep,paste("atom", "rep.in",sep="")))
 text <- readLines("atom.in",encoding="UTF-8")
 }
 #options(scipen=0)
 for( rep in 1: nrep){
 dir.create(sub("rep",rep,paste("input", "rep",sep="")))
 }
 for( rep in 1: nrep){
 file.copy(c(sub("rep",rep,paste("atom","rep.in",sep="")),"Inputscript.lammps")
 ,to=sub("rep",rep,paste("input", "rep",sep="")))
 }
for( rep in 1: nrep){
 setwd(paste(sub("rep",rep,"inputrep")))
 file.rename(from=sub("rep",rep,paste("atom", "rep.in",sep="")),
 to=sub("rep",rep,paste("atom", ".in",sep="")))
setwd("..")
 }
#stochasticity
for(fold in 1:npoints){
 source("stoch.R",echo=TRUE)
 }
```

4.3. Appendix C: stoch.R for generating replicate simulations

```
#NOTE: npoints2 is the number of stochastic repetitions
nvar2=2
setwd(paste(getwd(),"/",gsub("fold",fold,"inputfold"),sep=""))
para2=c("fix␣d1␣all␣divide␣1␣v_EPSdens␣v_divDia␣978595␣demflag␣0",
"fix␣e1␣HET␣eps_extract␣1␣v_EPSratio␣v_EPSdens␣6274")
val2=c(978595,6274)
nre2=npoints2
hyper2=maximinLHS(npoints2,nvar2, 2)
design2= matrix(NA,nrow=npoints2, ncol=nvar2)
```

```
for(j in 1:nvar2){
design2[,j]=round(qunif(hyper2[,j], min=0.50*val2[j], max=1.5*val2[j]))
}
text2 <- readLines( gsub("fold",fold,paste("Inputscript", ".lammps",sep="")),
encoding="UTF-8")
save(design2,file=paste(getwd(),"design2",sep="/"))
design2=rbind(val2,design2)
index=c(rep(NA,length(para2)))
for(i in 1:length(para2)){
index[i]=grep(para2[i],text2,value=FALSE,fixed=TRUE)
}
for(re2 in 1:nre2){
for(j in 1:nvar2){
text2[index[j]]=gsub(design2[re2,j] ,design2[(re2)+1,j], text2[index[j]])
}
writeLines(text2,gsub("re2",re2,paste("Inputscriptre2", ".lammps",sep="")))
}
for(re2 in 1: nre2){
dir.create(sub("re2",re2,paste("input", "re2",sep="")))
}
for( re2 in 1: nre2){
file.copy(c(sub("re2",re2,paste("Inputscriptre2", ".lammps",sep="")),"atom.in"),
 sub("re2",re2,paste("input", "re2",sep="")))
}
for(re2 in 1: nre2){
file.remove(sub("re2",re2,paste("Inputscriptre2", ".lammps",sep="")))
}
for(re2 in 1: nre2){
setwd(sub("re2",re2,paste("inputre2")))
file.rename(sub("re2",re2,paste("Inputscriptre2", ".lammps",sep="")),"Inputscript.lammps")
setwd("../")
}
setwd("..")
```

4.4. *Appendix D: main.sh for running the simulation on the Rocket machine*

```
#!/bin/bash
#SBATCH -A tcnufeb
# SLURM defaults to the directory you were working in when you submitted the job.
#Output files are also put in this directory. To set a different working directory add:
#SBATCH --workdir=/mnt/storage/nobackup/noo11/Andrew/inputnum/inputfold
#SBATCH --ntasks=4
#SBATCH -c 1
#BATCH --mail-type=ALL
module load GCC/6.4.0-2.28
module load OpenMPI/2.1.1-GCC-6.4.0-2.28
export PATH=$PATH:$HOME/nufeb2/nufeb/code/lammps5Nov16/src
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/local/lib
#module load intel
#module load OpenMPI/1.8.4-GCC-4.9.3-2.25
#module load OpenMPI/1.10.3-GCC-6.1.0-2.27
```

```
#module load GCC/6.4.0-2.28
srun lmp_rocket_gnu -in Input*
```

4.5. Appendix E: prep1d.R for preprocessing the NUFEB simulation data

```
path2=getwd()
CONC=CONC0=CONC1=list()
CONN=CON0=CON1=list()
npoints2=5
npoints=200
for(j in 1:npoints){
setwd(sub("j",j,paste(getwd(),"inputj",sep="/")))
path=getwd()
for (num in 1:npoints2){
setwd(sub("num",num,paste(getwd(),"inputnum",sep="/")))
con=as.matrix(read.csv("Results/ave_concentration.csv",header=TRUE))[,1:5]
ll=nrow(con)
bio=as.matrix(read.csv("Results/biomass.csv",header=TRUE,nrows=ll))[,1:4]
height=as.matrix(read.csv("Results/ave_height.csv",header=FALSE,nrows=ll))[,2]
rough=as.matrix(read.csv("Results/roughness.csv",header=FALSE,nrows=ll))[,2]
div=as.matrix(read.csv("Results/diversity.csv",header=FALSE,nrows=ll))[,2]
types=as.matrix(read.csv("Results/ntypes.csv",header=TRUE,nrows=ll))[,1:4]
#a=rbind(con[1,],con)
#b=rbind(types[1,],types)
#biom=matrix(as.numeric(bio),dim(bio))[-nrow(bio),-ncol(bio)]
mass=rowSums(bio)##total biomass
colnames(bio)=c("bio_het","bio_aob","bio_nob","bio_eps")
res=cbind(con,bio,types,height,rough,div)
#res=cbind(con,mass)
CONC[[num]]=res
setwd("../")
}
library(reshape2)
b1=do.call(rbind,lapply(CONC,melt))
b2=acast(b1,b1$Var1~b1$Var2,mean)
b3=acast(b1,b1$Var1~b1$Var2,sd)
CONC0=b2
CONC1=b3
#setwd("../")
#}
print(j)
print(num)
save(CONC,file=paste(path,"CONC",sep="/"))
save(CONC0,file=paste(path,"CONC0",sep="/"))
save(CONC1,file=paste(path,"CONC1",sep="/"))
###
CONN[[j]]=CONC
CON1[[j]]=CONC1
CON0[[j]]=CONC0
setwd("../")
}
```

```r
save(CONN,file=paste(path2,"CONN",sep="/"))
save(CON0,file=paste(path2,"CON0",sep="/"))
save(CON1,file=paste(path2,"CON1",sep="/"))
```

4.6. Appendix F: emulator.R for training the emulator

```r
load("CON0")
library(abind)
ndes=200
g=rep(NA,ndes)
for(i in 1:ndes){
g[i]=nrow(CON0[[i]])
}
dt=500
gg=ini=list()
for(i in 1:ndes){
ll=nrow(CON0[[i]])
gg[[i]]=(CON0[[i]][-1,-c(10:16)]-CON0[[i]][1:(ll-1),-c(10:16)])/dt##
#compute rate of change
ini[[i]]=(CON0[[i]][1:(ll-1),-c(10:16)])##compute present nutrient concentrations
}
np=9##number of outputs for emulation
drdt=abind(gg,along=1)
input1=abind(ini,along=1)
s=sample(1:nrow(drdt),2900)# number of training points
input=input1[s,1:np]
output=drdt[s,]
X=input##train inputs
Y=output##train outputs
Xstar=input1[-s,1:np]##validation/left-out
Ystar=drdt[-s,]##validation/left-out
write.table(X,file="X.csv",row.names=FALSE)
write.table(Y,file="Y.csv",row.names=FALSE)
write.table(Xstar,file="Xstar.csv",row.names=FALSE)
write.table(Ystar,file="Ystar.csv",row.names=FALSE)
#######################
###fitting
library(MASS)
library(emulator)
X=as.matrix(read.table('X.csv',header=TRUE))##training data
Y=as.matrix(read.table('Y.csv',header=TRUE))##training data
##scale input matrix to [0,1]
s1=apply(X,2,min)
s2=apply(X,2,max)
X=scale(X,scale=(s2-s1),center=s1)
############
H<- X#model.matrix(form,data=as.data.frame(X))
k<-dim(Y)[2]
m<-dim(H)[2]
n<-dim(Y)[1]
ninp=dim(X)[2]
```

```
id2=ninp+1
mycor=function(X1,X2,betai){
pdf<- diag(betai, nrow = length(betai))
tx1<- t(X1)
tx2 <- t(X2)
R1 <- X1%*%pdf%*%tx1
R2 <- X2%*%pdf%*%tx2
S1 <- t(as.matrix(diag(R1)))%x%rep(1,nrow(X2))
S2 <- as.matrix(diag(R2))%x%t(rep(1,nrow(X1)))
a1 <- t(tx1)%*%pdf%*%tx2
a2 <- t(tx2)%*%pdf%*%tx1
return(exp(Re(t(a1)+a2-S1-S2)))
}
## Function giving the log marginal likelihood
theta=runif(ninp+1)#rep(0.05,ninp+1)
likfun2<-function(theta){
R=mycor(X,X,betai=theta[-id2])
A=R+(theta[id2]*diag(1,dim(R)))##add random nugget/noise
Ainv=solve(A)
Omega=t(H)%*%Ainv%*%H
dA=(determinant(A)$modulus[1])###return log ofdeterminant
dOmega=(determinant(Omega)$modulus[1])##return log of determinant
Q1=t(Y)%*%Ainv%*%Y
Q3=t(solve(A,H))
Q4=t(Q3)%*%solve(Omega)%*%Q3
sig=Q1-t(Y)%*%Q4%*%Y
S=determinant(sig)$modulus[1]
-0.5*k*dA-0.5*k*dOmega-.5*(n-m)*S
}
##############
initial_scale=theta
lower=rep(1e-4,length(initial_scale))
upper=3*(apply(X,2,max)-apply(X,2,min))
#opt<-optim(fn=likfun2,par=initial_scale,method="L-BFGS-B",
control=list(fnscale= -1,trace=3),lower=lower,upper=upper)
 opt<-optim(fn=likfun2,par=initial_scale,method="L-BFGS-B",
 control=list(fnscale= -1,trace=3),lower=lower)

save(opt,file=paste(getwd(),"opt",sep="/"))
R=mycor(X,X,betai=opt$par[-id2])#corr matrix for output#2
A=R+(opt$par[id2]*diag(1,dim(R)))
Ainv=solve(A)#chol2inv(chol(A))###
model=list(X=X,Y=Y,theta=opt$par,Ainv=Ainv)
dirname=getwd()
save(model,file=paste(dirname,"model",sep="/"))###spa
#########Validation
pred=function(model,newdata){
X<- model$X
Y<- model$Y
theta1=model$theta[id2]
theta<- model$theta[-id2]
#form<- model$form
Ainv<- model$Ainv
A=solve(Ainv)
n2<- dim(newdata)[1]
n<- dim(X)[1]
X0<- newdata
#colnames(X0)=model$nam
######model matrix
H<- X#model.matrix(form,data=as.data.frame(X))
H0<- X0#model.matrix(form,as.data.frame(X0))
A01=mycor(X0,X,betai=theta)###cross correlation
A00=mycor(X0,X0,betai=theta)##test point correlation
Omega=t(H)%*%Ainv%*%H
betahat=solve(t(H)%*%Ainv%*%H)%*%(t(H)%*%Ainv%*%Y)
mu_star=H0%*%betahat+t(A01)%*%Ainv%*%(Y-H%*%betahat)
c_star=A00-(t(A01)%*%Ainv%*%A01)+(((H0-(t(A01)%*%
Ainv%*%H))%*%solve(Omega)%*%t(H0-(t(A01)%*%Ainv%*%H))))
Sigma=(t(Y-H%*%betahat)%*%Ainv%*%(Y-H%*%betahat))/(n-m)
svar=list()
for(i in 1:n2){
svar[[i]]=(diag(c_star)[i]*diag(Sigma))
}
K0=abind(svar,along=2)
out=list(mu=mu_star,K=t(K0))
return(out)
}
###Validation
Xstar=as.matrix(read.table('Xstar.csv',header=TRUE))##test data
X0=scale(Xstar,scale=(ss2-ss1),center=ss1)
Ystar=as.matrix(read.table('Ystar.csv',header=TRUE))##test data
gap=pred(model,newdata=X0)
mu=gap[[1]]
#proportion of variance explained
1-(colSums((Ystar-mu)^2)/colSums((Ystar-colMeans(Ystar))^2))
#diag(cor(Ystar,mu)^2)
#Calculate RMSE
sqrt(colMeans(((Ystar-mu)^2)))
##training
gap=pred(model,newdata=(X))
mu=gap[[1]]
diag(cor(Y,mu)^2)
#Calculate RMSE
sqrt(colMeans(((Y-mu)^2)))
#proportion of variance explained
1-(colSums((Y-mu)^2)/colSums((Y-colMeans(Y))^2))
##prepare the C^{++} auxilliary files
iOmega=solve(Omega)
input=(X[1,]*(s2-s1))+s1 ##unscaled
write.table(X,file="X.txt",row.names=FALSE,col.names=FALSE)
write.table(Y,file="Y.txt",row.names=FALSE,col.names=FALSE)
write.table(Ainv,file="Ainv2.txt",row.names=FALSE,col.names=FALSE)
write.table(iOmega,file="iOmega.txt",row.names=FALSE,col.names=FALSE)
write.table(betahat,file="betahat.txt",row.names=FALSE,col.names=FALSE)
write.table(Sigma,file="Sigma.txt",row.names=FALSE,col.names=FALSE)
write.table(input,file="input.txt",row.names=FALSE,col.names=FALSE)
```

4.7. Appendix G: emu.cpp for preparing the final emulator

```
//to compile g++ -I/usr/include/eigen3/ emu.cpp -o emu
#include <iostream>
#include <fstream>
#include <Eigen/Dense>
#include <Eigen/Core>
using namespace std;
using namespace Eigen;
//this function computes distance correlation
MatrixXd mycor(const MatrixXd& X1,const MatrixXd& X2,const VectorXd& betai)
{
MatrixXd ones = MatrixXd::Ones(1,X2.rows());
MatrixXd ones2 = MatrixXd::Ones(1,X1.rows());
MatrixXd pdf = betai.asDiagonal();//turn vector to digaonal matrix
MatrixXd R1 = X1*pdf*X1.transpose();
MatrixXd R2 = X2*pdf*X2.transpose();
MatrixXd S1 = R1.diagonal()*ones;
MatrixXd S2 = R2.diagonal()*ones2;
MatrixXd a1 = X1*pdf*X2.transpose(); //t(tx1)%*%pdf%*%tx2
MatrixXd a2 = X2*pdf*X1.transpose();
MatrixXd res = (a1.transpose()+a2-S1.transpose()-S2);
ArrayXXd res2 = res.array();
return(res2.exp());
}
//////////////////////
//main function
int main()
{
    int nrows=2900;
    int ncols=9;  //number of outputs change from 9 to 14
    int ncols2=9;
    double tau0=1;  //modify
    double tau=0.0001000;
VectorXd betai(9);
betai << 3.4938999,3.4186248,3.8043138,0.8883166,1.1512325,2.9979449,
2.5629414,0.8038899,1.1486920;

//read training input dat matrix X
MatrixXd X = MatrixXd::Zero(nrows,ncols2);
ifstream fin1 ("./X.txt");
if (fin1.is_open())
{
for (int row = 0; row < nrows; row++)
for (int col = 0; col < ncols2; col++)
{
float item;
fin1 >> item;
X(row, col) = item;
}
fin1.close();
}
//read in output matrix Y
MatrixXd Y = MatrixXd::Zero(nrows,ncols);
ifstream fin2 ("./Y.txt");
if (fin2.is_open())
{
for (int row = 0; row < nrows; row++)
for (int col = 0; col < ncols; col++)
{
float item;
fin2 >> item;
Y(row, col) = item;
}
fin2.close();
}

//read in new data point for prediction
MatrixXd newdata = MatrixXd::Zero(1,ncols2);
ifstream fin3 ("./input/input.txt");
if (fin3.is_open())
{
for (int row = 0; row < 1; row++)
for (int col = 0; col < ncols2; col++)
{
float item;
fin3 >> item;
newdata(row,col) = item;
}
fin3.close();
}
//read in Ainv matrix
MatrixXd Ainv = MatrixXd::Zero(nrows,nrows);
ifstream fin4 ("./Ainv2.txt"); //modify
if (fin4.is_open())
{
for (int row = 0; row < nrows; row++) //modify
for (int col = 0; col < nrows; col++) //modify
{
float item;
fin4 >> item;
Ainv(row, col) = item;
}
fin4.close();
```

```
}
//read in iOmega matrix
MatrixXd iOmega = MatrixXd::Zero(ncols2,ncols2);
ifstream fin5 ("./iOmega.txt");
if (fin5.is_open())
{
for (int row = 0; row < ncols2; row++)
for (int col = 0; col < ncols2; col++)
{
float item;
fin5 >> item;
iOmega(row, col) = item;
}
fin5.close();
}
//read in betahat matrix
MatrixXd betahat = MatrixXd::Zero(ncols2,ncols);
ifstream fin6 ("./betahat.txt");
if (fin6.is_open())
{
for (int row = 0; row < ncols2; row++)
for (int col = 0; col < ncols; col++)
{
float item;
fin6 >> item;
betahat(row, col) = item;
}
fin6.close();
}
//normalization data
MatrixXd s1(1,9);
MatrixXd s2(1,9);
s2 << 4.131475e-02,1.308324e-02,8.995006e-03,8.995000e-03,8.995000e-03,2.275674e-10
,2.159337e-10,3.977501e-14,3.987490e-14;
s1 << 3.593248e-03,1.885759e-03,2.622689e-119,5.293362e-118,4.234222e-116,1.080608e-14
,4.539888e-15,2.840420e-15,2.840420e-15;
VectorXd s3(9);
s3 << -6.245067,
 -6.39793,
 -6.948577,
  -8.282121,
   -5.484406,
    -32.23061,
     4.382027,
      -13.79393,
       -15.26332;
MatrixXd H = X.array();// training inputs
MatrixXd H0 = (newdata.array()-s1.array())/s2.array();//scale the test input
int m=H.cols();
int n=Y.rows();
int n2=H0.rows();
MatrixXd A00, A01;

//MatrixXd gaga = MatrixXd::Zero(2900,9);
// gaga = (H.colwise()-s3);//.rowwise().squaredNorm();
A01.noalias() = mycor(H0,H,betai);//###cross correlation//dont evaluate yet
 A00.noalias() = mycor(H0,H0,betai);//##test point correlation
//MatrixXd A = mycor(H,H,betai);//##very big matrix precomputed in R
MatrixXd temp = MatrixXd::Identity(A00.rows(), A00.cols());
 A00 += ((tau0*tau)*temp);
//Ainv.noalias() = A.inverse();
 /**
//MatrixXd iOmega = (H.transpose()*Ainv*H).inverse();//can be precomputed in R
//MatrixXd betahat= iOmega*(H.transpose()*Ainv*Y);// can be precomputed in R
* */

MatrixXd mu_star = H0*betahat+A01.transpose()*Ainv.selfadjointView<Upper>()
*(Y-H*betahat);
MatrixXd r1 = H0-(A01.transpose()*Ainv.selfadjointView<Upper>()*H);
MatrixXd c_star = A00-(A01.transpose()*Ainv.selfadjointView<Upper>()*A01)
+(((r1)*(iOmega)*r1.transpose()));
//MatrixXd Sigma = ((Y-H*betahat).transpose()*Ainv.selfadjointView<Upper>()
*(Y-H*betahat))/(n-m); precomputed in R
//MatrixXd Sigma(4,4);
//Sigma << 3.273639283317, 33.2402996825996, 22.3515408957194, 7.52106455280949,
//33.2402996826023, 33.3135878103714, 22.3313232827605, 7.51532055692301,
//22.3515408957016, 22.3313232827417, 31.1848209695244, -5.26625223699458,
//7.52106455282121, 7.51532055693379, -5.26625223699611, 33.5773444066967;
MatrixXd rate = mu_star;//(mu_star.array()*s2.array())+s1.array();
//output results
ofstream a_file ("./output/Rate.txt");
a_file << rate;
a_file.close();
//cout << H0 << endl;
}
```

4.8. *Appendix H: readme.txt for compilation instructions*

```
############LAMMPS EMULATOR version V1i####################
(A) DESCRIPTION
A coupled-emulator which produces rates of change of nutrient concentrations namely
(Sub,O2,NO2,NO3,NH4) and rates of change of the following characterized outputs
(rate of change of nutrient concentration and Biomass for each species) respectively.
For instance, dS/dt=(S_2-S_1)/(T_2-T_1)

#9 outputs
(i) The rate of change of SUB/Carbon substrate concentration (kg/m^3)
(ii) The rate of change of O2 concentration (kg/m^3)
(iii) The rate of change of NO2 concentration (kg/m^3)
(iv) The rate of change of NO3 concentration (kg/m^3)
(v) The rate of change of NH4 concentration (kg/m^3)
```

```
(vi) The rate of change of biomass for HET (kg)
(vii) The rate of change of biomass for AOB (kg)
(viii) The rate of change of biomass for NOB (kg)
(ix) The rate of change of biomass for EPS (kg)

(B) All required data and C++ scripts for the emulator are in
the directory "LAMMPS_emulator_V1i".
The main c++ source code is "emu.cpp".

(C) Input file description:
A sample "input.txt" file is placed in the "input" subfolder which
 is a required input parameter for running the emulator.
This contain 9 different values
The input values [1-5] represent "SUB","O2","NO2","NO3","NH4"
local nutrient substrates (kg/m^3) respectively.
The input values [6-9] represent current values of Biomass
concentration for each species.

(D) RUN the code:
The main source code is "emu.cpp". The emulator can be run from either
 the Window or Linux platform.
Put the input file "input.txt" to be run in the subfolder "input".
 These files contain the parameter values and initial conditions.
To run any case, the input parameters must be converted to the
units stated above to get correct results.

To run from Linux terminal:
$ cd path to /LAMMPS_emulator_V1i
/** compile .cpp file
$ g++ -O3 -I/usr/include/eigen3/ emu.cpp -o emu
/** run the code as
$ ./emu

(E) Obtain the simulation results from subfolder "output".
(F) The valid ranges of the nutrient concentrations (kg/m^3) and other parameters:
["SUB","O2","NO2","NO3", "NH4","Biomass_HET",
"Biomass_AOB","Biomass_NOB","Biomass_EPS"]
min=[0.003593248 0.001885759  2.622689e-119 5.293362e-118 4.234222e-116
 1.046317e-14  4.196982e-15    2.840420e-15   2.840420e-15]
max=[0.044765000 0.014969000  8.995006e-03  8.995000e-03  8.995000e-03
 2.275782e-10  2.159382e-10    4.261543e-14   4.271532e-14]

NOTE: This emulator will produce output for any specified time-steps and
any parameter combinations.
CONTACT: oluwole.oyebamiji@ncl.ac.uk or wolemi2@yahoo.com for further details.
```

Table 2: **List of parameters that remain unchanged for all simulation experiments**

Index	Parameters	Values	Units	References
1	$K_{Sub,HET}$	0.004	kgm^{-3}	Ofiţeru et al. [11]
2	$K_{O_2,HET}$	0.0002	kgm^{-3}	Ofiţeru et al. [11]
3	$K_{NO_2,HET}$	0.0003	kgm^{-3}	Ofiţeru et al. [11]
4	$K_{NO_3,HET}$	0.0003	kgm^{-3}	Ofiţeru et al. [11]
5	$K_{O_2,AOB}$	0.0005	kgm^{-3}	Bruce and Perry [12]
6	$K_{NH_3,AOB}$	0.001	kgm^{-3}	Bruce and Perry [12]
7	$K_{O_2,NOB}$	0.0005	kgm^{-3}	Bruce and Perry [12]
8	$K_{NO_2,NOB}$	0.001	kgm^{-3}	Bruce and Perry [12]
	Diffusion coef			
9	D_{Sub}	1.1574×10^{-9}	$m^2 s^{-1}$	Alpkvist et al. [13]
10	D_{O_2}	2.3148×10^{-9}	$m^2 s^{-1}$	Alpkvist et al. [13]
11	D_{NO_2}	1.967×10^{-9}	$m^2 s^{-1}$	Alpkvist et al. [13]
12	D_{NO_3}	1.967×10^{-9}	$m^2 s^{-1}$	Alpkvist et al. [13]
13	D_{NH_4}	1.967×10^{-9}	$m^2 s^{-1}$	Alpkvist et al. [13]
	Decay coef			
14	HET	9.166×10^{-7}	s^{-1}	-
15	AOB	3.472×10^{-7}	s^{-1}	-
16	NOB	3.472×10^{-7}	s^{-1}	-
	Other parameters			
17	Biomass density	150	kg/m^3	Lardon et al. [8]
18	EPS density	30	kg/m^3	Lardon et al. [8]

References

[1] O. K. Oyebamiji, D. J. Wilkinson, P. G. Jayathilake, S. P. Rushton, B. Bridgens, B. Li, P. Zuliani, A Bayesian approach to modelling the impact of hydrodynamic shear stress on biofilm deformation, PLoS ONE 13 (4) (2018) e0195484, doi:doi.org/10.1371/journal.pone.0195484.

[2] O. K. Oyebamiji, N. R. Edwards, P. B. Holden, P. H. Garthwaite, S. Schaphoff, D. Gerten, Emulating global climate change impacts on crop yields, Statistical Modelling 15 (6) (2015) 499–525, doi:https://doi.org/10.1177/1471082X14568248.

[3] O. Oyebamiji, D. Wilkinson, P. Jayathilake, T. Curtis, S. Rushton, B. Li, P. Gupta, Gaussian process emulation of an individual-based model simulation of microbial communities, Journal of Computational Science 22 (2017) 69–84.

[4] S. Conti, A. O'Hagan, Bayesian emulation of complex multi-output and dynamic computer models, Journal of statistical planning and inference 140 (3) (2010) 640–651.

[5] P. G. Jayathilake, P. Gupta, B. Li, C. Madsen, O. Oyebamiji, R. González-Cabaleiro, S. Rushton, B. Bridgens, D. Swailes, B. Allen, et al., A mechanistic individual-based model of microbial communities, PloS one 12 (8) (2017) e0181965.

[6] P. G. Jayathilake, S. Jana, S. Rushton, D. Swailes, B. Bridgens, T. Curtis, J. Chen, Extracellular Polymeric Substance Production and Aggregated Bacteria Colonization Influence the Competition of Microbes in Biofilms, Frontiers in microbiology 8 (2017) 1865.

[7] T. J. Santner, B. J. Williams, W. I. Notz, The design and analysis of computer experiments, Springer Science & Business Media, 2013.

[8] L. A. Lardon, B. V. Merkey, S. Martins, A. Dötsch, C. Picioreanu, J.-U. Kreft, B. F. Smets, iDynoMiCS: next-generation individual-based modelling of biofilms, Environmental Microbiology 13 (9) (2011) 2416–2434.

[9] J. Sacks, W. J. Welch, T. J. Mitchell, H. P. Wynn, Design and analysis of computer experiments, Statistical science (1989) 409–423.

[10] T. E. Fricker, J. E. Oakley, N. M. Urban, Multivariate Gaussian process emulators with nonseparable covariance structures, Technometrics 55 (1) (2013) 47–56.

[11] I. D. Ofiţeru, M. Bellucci, C. Picioreanu, V. Lavric, T. P. Curtis, Multi-scale modelling of bioreactor–separator system for wastewater treatment with two-dimensional activated sludge floc dynamics, water research 50 (2014) 382–395.

[12] E. R. Bruce, L. M. Perry, Environmental biotechnology: principles and applications, New York: McGrawHill 400.

[13] E. Alpkvist, C. Picioreanu, M. van Loosdrecht, A. Heyden, Three-dimensional biofilm model with individual cells and continuum EPS matrix, Biotechnology and bioengineering 94 (5) (2006) 961–979.