Supplementary Material

Supplement S1: Further Details of the Partial Least Squares approach

PLS operates by projecting **X** and **Y** into new spaces, determined by maximising the covariance between the projections of **X** and **Y**. The transformations of **X** and **X** are given by $\mathbf{X} = \mathbf{T}\mathbf{P}^{\mathbf{T}} + \mathbf{E}$ and $\mathbf{Y} = \mathbf{U}\mathbf{Q}^{\mathbf{T}} + \mathbf{F}$, where **T** and **U** are the projections of **X** and **Y** respectively, **P** and **Q** are orthogonal matrices, and **E** and **F** are independent and identically distributed error terms. It is straightforward to show that $\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{G}$, where $\mathbf{B} = \mathbf{X}^{-1}\mathbf{T}\mathbf{Q}^{\mathbf{T}}$ and $\mathbf{G} = (\mathbf{U} - \mathbf{T})\mathbf{Q}^{\mathbf{T}} + \mathbf{F}$. The **B** matrix stores the PLS-regression coefficients, which can be interpreted as sensitivity indices (Chang et al., 2015; Sobie, 2009).

Supplement S2: Calculating first order and total sensitivity indices using the Sobol method without dimension reduction

We run the following R script file:

```
X =read.csv('InputsNorm_TrainingData.csv', header=FALSE)
yALL =read.csv('Outputs_TrainingData.csv', header=FALSE)
SI = matrix(-9999,nrow=dim(inputs)[2],ncol= dim(outputs)[2])
SI.total = matrix(-9999,nrow=dim(inputs)[2],ncol= dim(outputs)[2])
for (j in 1: dim(outputs)[2]){y = as.matrix(yALL[,j],rownames.force=NA)
m = km(~ ., design = X, response = y, covtype = "matern3_2")
AB = randomLHS(N,16); A = AB[,1: dim(X)[2]]; B = AB[,(dim(X)[2]+1):(dim(X)[2]*2)]
yA = predict.km(m, A, "UK", se.compute = FALSE, checkNames = FALSE)$mean
yB = predict.km(m, B, "UK", se.compute = FALSE, checkNames = FALSE)$mean
yA_sum = mean(yA); yB_sum = mean(yB); yAyA_sum = mean(yA^2); yByB_sum = mean(yB^2)
```

```
for (i in 1: dim(X)[2]){print(c(j,i)); Ci=B; Ci[,i]=A[,i]; Di=A; Di[,i]=B[,i]
yCi = predict.km(m, Ci, "UK", se.compute = FALSE, checkNames = FALSE)$mean
yDi = predict.km(m, Di, "UK", se.compute = FALSE, checkNames = FALSE)$mean
yCi_sum = mean(yCi); yDi_sum = mean(yDi); yCiyCi_sum = mean(yCi^2);
yDiyDi_sum = mean(yDi^2); yAyCi_sum = mean(yA*yCi);
yAyDi_sum = mean(yA*yDi); yByCi_sum = mean(yB*yCi);
yByDi_sum = mean(yB*yDi); SI_temp = rep(-9999, 8)
SI_temp[1] = (yAyCi_sum - (yA_sum*yB_sum))/(yAyA_sum - (yA_sum*yB_sum))
SI_temp[2] = (yByDi_sum - (yA_sum*yB_sum))/(yByB_sum - (yA_sum*yB_sum))
SI_temp[3] = (yAyCi_sum - (yA_sum*yB_sum))/(yByB_sum - (yA_sum*yB_sum))
SI_temp[4] = (yAyCi_sum - (yCi_sum*yDi_sum))/(yCiyCi_sum - (yCi_sum*yDi_sum))
SI_temp[5] = (yAyCi_sum - (yCi_sum*yDi_sum))/(yDiyDi_sum - (yCi_sum*yDi_sum))
SI_temp[6] = (yByDi_sum - (yA_sum*yB_sum))/(yAyA_sum - (yA_sum*yB_sum))
SI_temp[7] = (yByDi_sum - (yCi_sum*yDi_sum))/(yCiyCi_sum - (yCi_sum*yDi_sum))
SI_temp[8] = (yByDi_sum - (yCi_sum*yDi_sum))/(yDiyDi_sum - (yCi_sum*yDi_sum))
SI[i,j] = mean(SI_temp)}}
write.table(SI*100, "SIs_Sobol_EmulatorOnly.csv", row.names=F, col.names=F, sep=",")
```

To compute the total indices, the eight SI_temp lines can be replaced with the following:

```
SI_temp[1] = (yAyDi_sum - (yA_sum*yB_sum))/(yAyA_sum - (yA_sum*yB_sum))
SI_temp[2] = (yByCi_sum - (yA_sum*yB_sum))/(yByB_sum - (yA_sum*yB_sum))
SI_temp[3] = (yAyDi_sum - (yCi_sum*yDi_sum))/(yAyA_sum - (yCi_sum*yDi _sum))
SI_temp[4] = (yByCi_sum - (yCi_sum*yDi _sum))/(yByB_sum - (yCi_sum*yDi _sum))
```

Supplement S3: Calculating first order and total sensitivity indices using the Sobol method, using principal component analysis to reduce the dimensionality of the output

We run the following R script file to compute the sensitivity indices for the emulator-PCA hybrid approach. The R script below can be modified to ensure that the number of principal components included accounts for a specified proportion of the variance (chosen here to be 99%).

```
X =read.csv('InputsNorm_TrainingData.csv', header=FALSE)
yALL =read.csv('Outputs_TrainingData.csv', header=FALSE)
S = var(Y); S.eig = eigen(S)
eig.value.cumul = cumsum((S.eig$values/sum(S.eig$values))*100)
yALL.dim.index = c(1:dim(yALL)[2]); Npca=min(yALL.dim.index[eig.value.cumul>99])
ptm <- proc.time()
AB <- randomLHS(N,(dim(X)[2]*2))
A <- AB[,1:dim(X)[2]];   B <- AB[,(dim(X)[2]+1):(dim(X)[2]*2)]
yA_PCA=matrix(-9999,N,Npixels); yB_PCA=matrix(-9999,N,Npixels)
for (j in 1:Npc){
  PC <- matrix(rep(S.eig$vectors[,j],dim(X)[1]),nrow=dim(X)[1],ncol=Npixels,byrow=TRUE)
  y_PCj <- as.matrix(rowSums(PC*Y),rownames.force=NA)
  assign(paste("m",j, sep=""),km(~ ., design = X, response = y_PCj, covtype = "matern5_2"))
  yA_PCA[,j] <- predict.km(eval(as.symbol(paste("m",j,sep=""))), A, "UK", se.compute =
FALSE, checkNames = FALSE)$mean
  yB_PCA[,j] <- predict.km(eval(as.symbol(paste("m",j,sep=""))), B, "UK", se.compute =
FALSE, checkNames = FALSE)$mean
```

```
}
for (j in (Npc+1):Npixels){
yA_PCA[,j]=matrix(0,nrow=N,ncol=1); yB_PCA[,j]=matrix(0,nrow=N,ncol=1)
}
PCs.inv <- solve(S.eig$vectors); yA <- yA_PCA%*%PCs.inv; yB <- yB_PCA%*%PCs.inv
yCi <- matrix(-9999,nrow=N,ncol=Npixels*dim(X)[2])
yDi <- matrix(-9999,nrow=N,ncol=Npixels*dim(X)[2])
for (i in 1:dim(X)[2]){
  print(c(i));  Ci=B; Ci[,i]=A[,i];  Di=A;  Di[,i]=B[,i]
  yCj_PCA <- matrix(-9999,nrow=N,ncol=Npixels)
  yDj_PCA <- matrix(-9999,nrow=N,ncol=Npixels)
  for (j in 1:Npc){
    yCj_PCA[,j] <- predict.km(eval(as.symbol(paste("m",j,sep=""))), Ci, "UK", se.compute =
FALSE, checkNames = FALSE)$mean
    yDj_PCA[,j] <- predict.km(eval(as.symbol(paste("m",j,sep=""))), Di, "UK", se.compute =
FALSE, checkNames = FALSE)$mean
  }
  for (j in (Npc+1):Npixels){
  yCj_PCA[,j] <- matrix(0,nrow=N,ncol=1);  yDj_PCA[,j] <- matrix(0,nrow=N,ncol=1)
}
  i2 <- i*Npixels;  i1 <- i2 - (Npixels-1)
  yCi[,i1:i2] <- yCj_PCA%*%PCs.inv;   yDi[,i1:i2] <- yDj_PCA%*%PCs.inv
}
```

```
SI <- matrix(-9999,nrow=8,ncol=Npixels)
for (i in 1:Npixels){
  yA_sum <- mean(yA[,i]);  yB_sum <- mean(yB[,i]);
  yAyA_sum <- mean(yA[,i]^2); yByB_sum <- mean(yB[,i]^2)
  SI_temp <- rep(-9999,8)
  for (j in 1:8){
    z <- ((j-1)*Npixels) + I;  yCi_sum <- mean(yCi[,z]);   yDi_sum <- mean(yDi[,z])
    yCiyCi_sum <- mean(yCi[,z]^2);   yDiyDi_sum <- mean(yDi[,z]^2)
    yAyCi_sum <- mean(yA[,i]*yCi[,z]);   yAyDi_sum <- mean(yA[,i]*yDi[,z])
    yByCi_sum <- mean(yB[,i]*yCi[,z]);    yByDi_sum <- mean(yB[,i]*yDi[,z])
    SI_temp[1] <- (yAyCi_sum - (yA_sum*yB_sum))/(yAyA_sum - (yA_sum*yB_sum))
    SI_temp[2] <- (yByDi_sum - (yA_sum*yB_sum))/(yByB_sum- (yA_sum*yB_sum))
    SI_temp[3] <- (yAyCi_sum - (yA_sum*yB_sum))/(yByB_sum - (yA_sum*yB_sum))
    SI_temp[4] <- (yAyCi_sum - (yCi_sum*yDi_sum))/(yCiyCi_sum - (yCi_sum*yDi_sum))
    SI_temp[5] <- (yAyCi_sum - (yCi_sum*yDi_sum))/(yDiyDi_sum - (yCi_sum*yDi_sum))
    SI_temp[6] <- (yByDi_sum - (yA_sum*yB_sum))/(yAyA_sum - (yA_sum*yB_sum))
    SI_temp[7] <- (yByDi_sum - (yCi_sum*yDi_sum))/(yCiyCi_sum - (yCi_sum*yDi_sum))
    SI_temp[8] <- (yByDi_sum - (yCi_sum*yDi_sum))/(yDiyDi_sum - (yCi_sum*yDi_sum))
    SI[j,i] <- mean(SI_temp)
  }
}
proc.time() – ptm    #Stop the clock
```

```
write.csv(data.frame(SI*100), file =
paste("SIs_Sobol_EmulatorPCA_",modelN,"_N",N,".csv",sep=""), row.names=FALSE)
```

Supplement S4: Basic emulator diagnostic checks

We run the followingR script file:

```
library(sensitivity); library(DiceKriging); library(DiceOptim); library(lhs);
X =read.csv('InputsNorm_TrainingData.csv', header=FALSE)
inputs_val_norm = maximinLHS(30,dim(X)[2])
write.table(inputs_val_norm,"InputsNorm_TrainingData.csv", row.names=F, col.names=F,
sep=",")
inputs_val = matrix(-9999,nrow=30,ncol=dim(X)[2]))
for (i in 1:dim(X)[2]){inputs[,i] = (inputs_val_norm[,i]*(max(X[,i])-min(X[,i])) + min(X[,i
write.table(inputs_val,"Inputs_ValidationRuns.csv", row.names=F, col.names=F, sep=",")
```

Each row of the *Inputs_ValidationRuns.csv* file provides the input values for a given model simulation, and we write the output to a corresponding row in the output file *Outputs_ValidationRuns.csv*. The diagnostic check compares the outputs of the simulator at the validation inputs with those predicted by the emulator. We run the following R script file to carry out this diagnostic check:

```
X=read.csv('InputsNorm_TrainingData.csv', header=FALSE)
yALL =read.csv('Outputs_TrainingData.csv', header=FALSE)
X_val=read.csv('InputsNorm_ValidationRuns.csv', header=FALSE)
yALL_val =read.csv('Outputs_ValidationRuns.csv', header=FALSE)
for (j in 1: dim(yALL)[2]){y = as.matrix(yALL[,j],rownames.force=NA)
```

```
m = km(~ ., design = X, response = y, covtype = "matern5_2")
yALL_val_emul[,j] = as.matrix(predict.km(m, X_val, "UK", se.compute=FALSE,
checkNames=FALSE)$mean)}
plot(c(yALL_val),c(yALL_val_emul),main="Emulator versus Simulator predictions at validation
inputs",xlab="simulator", ylab="emulator").
print(cor(c(x),c(y))^2)*100
```

This prints the value of the coefficient of determination, R^2.

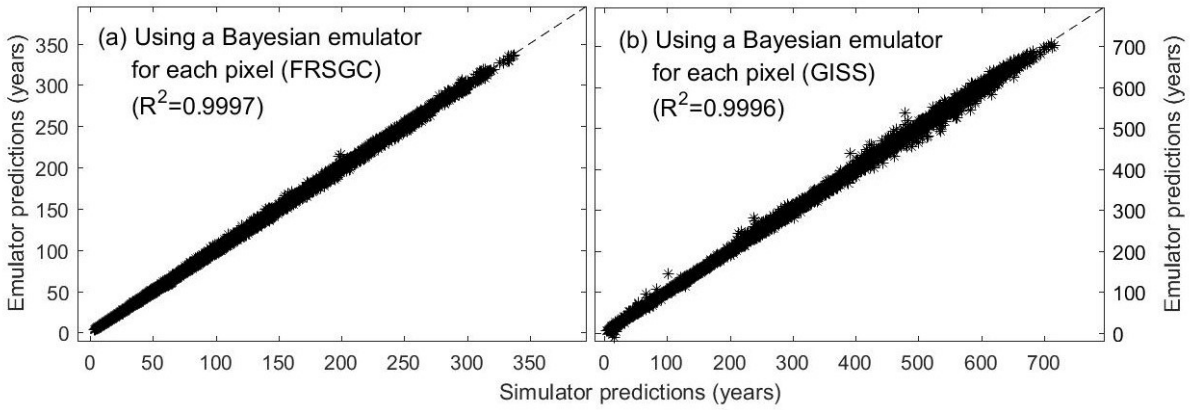

Figure S1. Annual column mean CH_4 lifetime calculated by the FRSGC and GISS chemistry models (x-axis) versus predicted by the Gaussian Process emulator (y-axis).

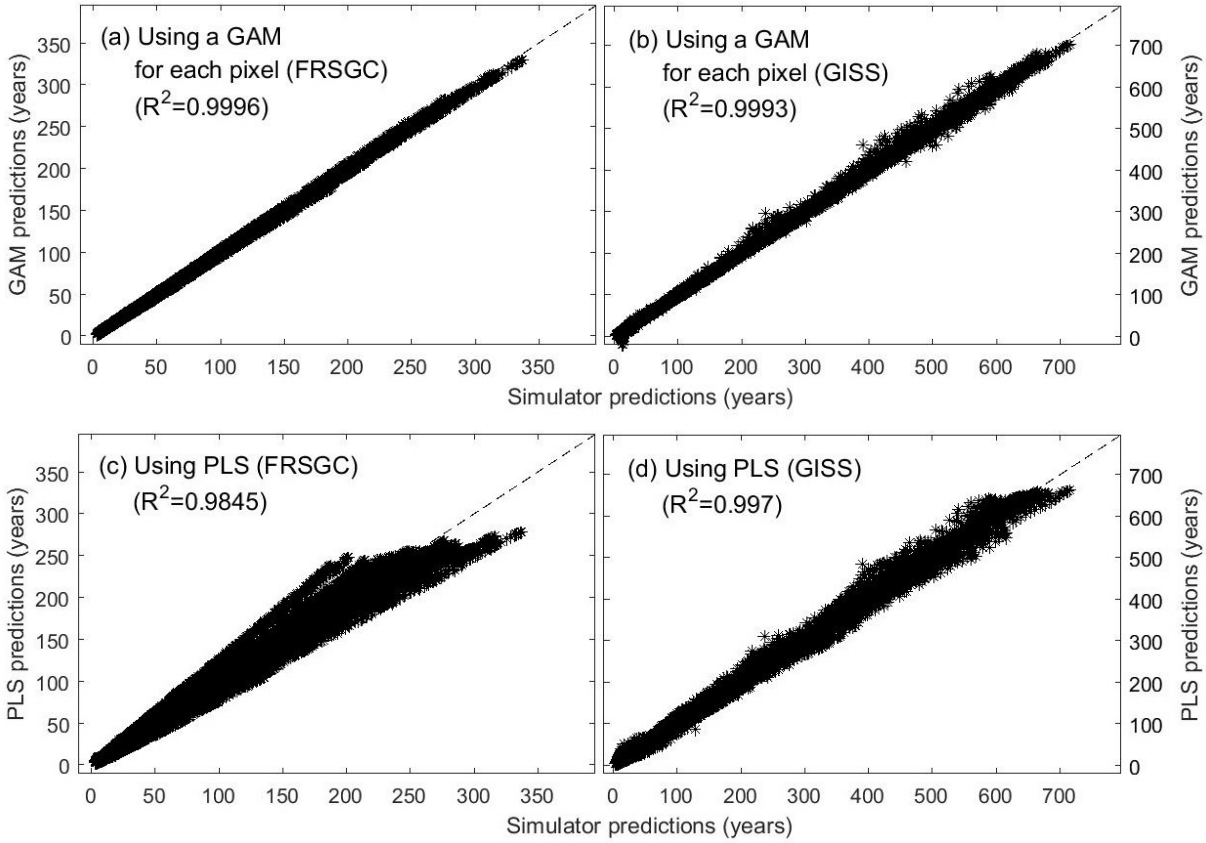

Figure S2. Annual column mean CH_4 lifetime calculated by the FRSGC and GISS chemistry models (x-axis) versus predictions by a separate generalized additive model (panels a and b) or partial least squares model (panels c and d) for each pixel (y-axis).

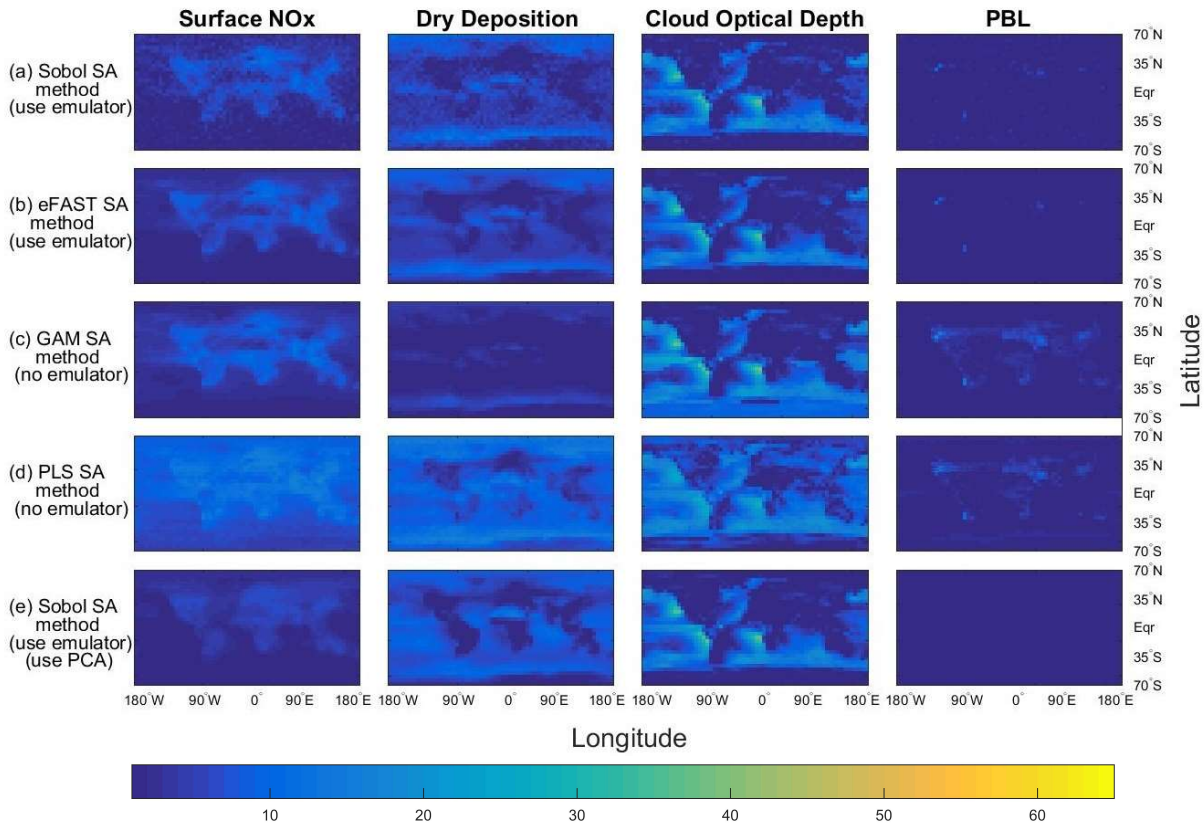

Figure S3. Sensitivity indices for the four minor inputs for the FRSGC chemistry transport model showing the inputs not already included in Figure 3.

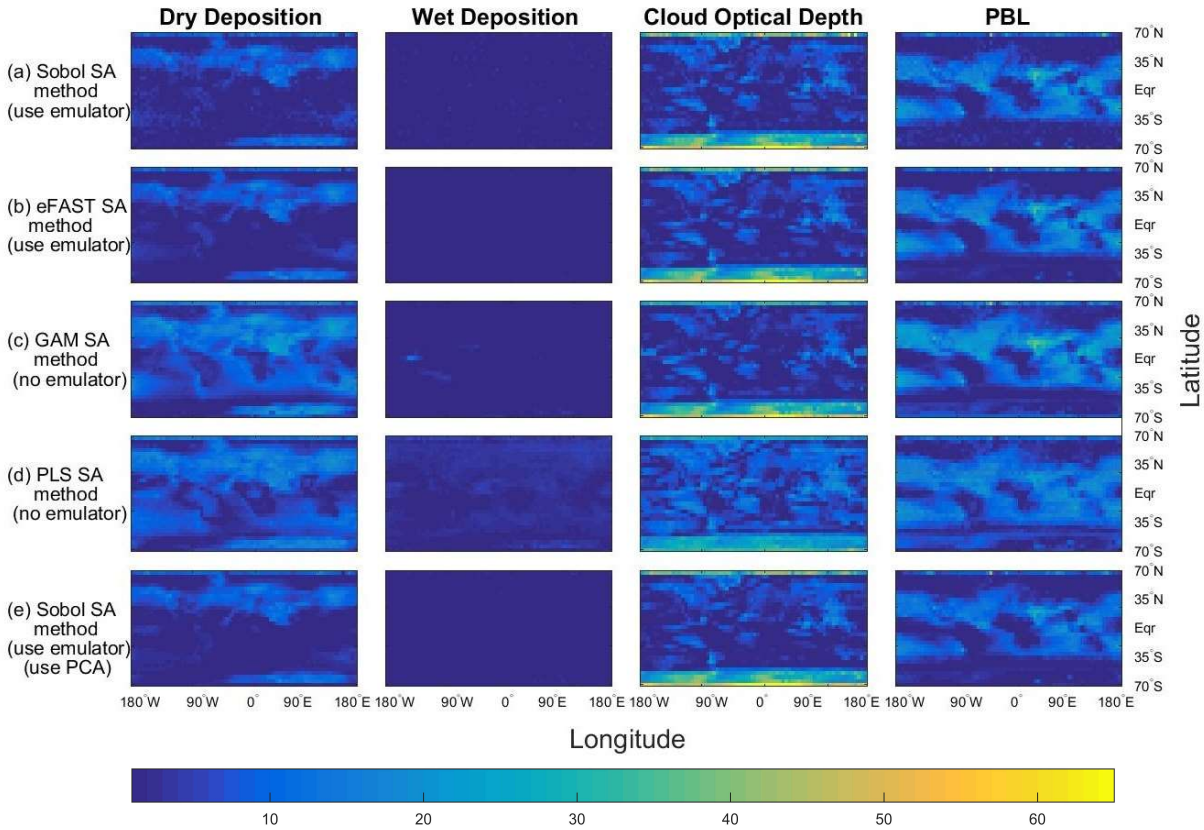

Figure S4. Sensitivity indices for the four minor inputs for the GISS chemistry transport model showing the inputs not already included in Figure 4.