
Article:
Smith, M.T., Ross, M., Ssematimba, J. et al. (3 more authors) (2023) Modelling calibration uncertainty in networks of environmental sensors. Journal of the Royal Statistical Society Series C: Applied Statistics, 72 (5). pp. 1187-1209. ISSN 0035-9254

https://doi.org/10.1093/jrsssc/qlad075

Supplementary Material

| Algorithms for Inference and Prediction in the Calibration Pair Model

Algorithm 1 Variational Inference for calibration.

Note: For implementation we structure an input matrix $\boldsymbol{X}$ to hold the time, sensor and component as three columns. This matrix is $2C$ times the number of observation pairs (N, the length of Y). Split into C submatrices, each pair of rows in each submatrix is associated with one row of $\boldsymbol{Y}$. $\boldsymbol{f}$ now becomes a vector with each item associated with one row of $\boldsymbol{X}$. The parameters are selected using slice notiation.

Inputs

Observation time, sensor and parameter, $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{2CN}$;
Observation pair values, $\boldsymbol{Y} = \{[y_i^{(1)}, y_i^{(2)}]\}_{i=1}^{N}$;
Inducing point locations (time, sensor and parameter), $\boldsymbol{Z} = \{\boldsymbol{z}_i\}_{i=1}^{M}$;
Calibration function, $\phi(y, \boldsymbol{c})$;
Number of parameters used by function, C;
Kernel, $k(\cdot,\cdot)$;
Likelihood variance σ^2;
Reference sensors, $\boldsymbol{r} = \{0,1\}^S$;
Number of samples in MC approximation, P.

Outputs

Approximating Gaussian distribution parameters: mean $\boldsymbol{m}$ and factor $\boldsymbol{R}$ (where distribution covariance is $RR^\top$).

procedure VI($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \phi(\cdot), C, k(\cdot,\cdot), \sigma^2, \boldsymbol{r}$)
  while $\boldsymbol{m}$ or $\boldsymbol{R}$ not converged **do**
    ▷ $q(\boldsymbol{f})$ is the approximate posterior over all latent variables defined in $\boldsymbol{X}$.
    $q(\boldsymbol{f}) \sim N(K_{xz}K_{zz}^{-1}\boldsymbol{m}, K_{xx} - K_{xz}K_{zz}^{-1}K_{zx} + K_{xz}K_{zz}^{-1}RR^\top K_{zz}^{-1}K_{zx})$
    for $j = 1..P$ **do** ▷ Sample P times
      for $i = 1..N$ **do**
        ▷ Sample the latent variables relevant to the two sensors.
        $\boldsymbol{s}_i^{(1)}, \boldsymbol{s}_i^{(2)} = \text{sample}[q(\boldsymbol{f}_{2i-1::2N}), q(\boldsymbol{f}_{2i::2N})]$ ▷ $\boldsymbol{s}_i^{(1)}$ and $\boldsymbol{s}_i^{(2)}$ are each of length C.
        $L_{ij} = \log p\left(\begin{bmatrix} y_i^{(1)} \\ y_i^{(2)} \end{bmatrix} \middle| \middle| \begin{bmatrix} \boldsymbol{s}_i^{(1)} \\ \boldsymbol{s}_i^{(2)} \end{bmatrix}\right)$ ▷ Compute likelihood of sample, using (4)
      end for
    end for
    $\mathcal{L} \leftarrow \frac{1}{P}\sum_{j=1}^{P}\sum_{i=1}^{N}(L_{ij}) - D_{KL}\left(N(\boldsymbol{m}, RR^\top), N(0, K_{ZZ})\right)$ ▷ Compute ELBO
    compute $\frac{d\mathcal{L}}{\boldsymbol{m}}$ and $\frac{d\mathcal{L}}{\boldsymbol{R}}$ ▷ using automatic differentiation.
    update $\boldsymbol{m}$ and $\boldsymbol{R}$ using stochastic gradient descent (using Adam).
  end while
end procedure

Algorithm 2 Prediction for Algorithm 1

Inputs

Test time, sensor and component, $\boldsymbol{X}_* = \{\boldsymbol{x}_{*i}\}_{i=1}^{N_*}$; $\boldsymbol{y}_* = \{y_{*i}\}_{i=1}^{N_*}$ raw observations (unlike normal regression we need to give uncalibrated observations at test time)

Inducing point locations (time, sensor and component), $\boldsymbol{Z} = \{\boldsymbol{z}_i\}_{i=1}^{M}$;

Calibration function, $\phi(y, \boldsymbol{c})$;

Number of components used by function, C;

Kernel, $k(\cdot,\cdot)$;

Approximating Gaussian distribution parameters: $\boldsymbol{m}$ and $\boldsymbol{R}$.

Number of samples for each test point, P.

Outputs

An $N_* \times P$ matrix of P samples for each of the N_* test points, $\boldsymbol{S}$

procedure Predict($\boldsymbol{X}_*, \boldsymbol{Y}_*, \boldsymbol{Z}, \phi(\cdot), C, k(\cdot,\cdot)$)

▷ $q(\boldsymbol{f})$ is the approximate posterior over all latent variables defined in $\boldsymbol{X}_*$.

$q(\boldsymbol{f}) \sim N(K_{x_*z}K_{zz}^{-1}\boldsymbol{m}, K_{x_*x_*} - K_{xz}K_{zz}^{-1}K_{zx_*} + K_{x_*z}K_{zz}^{-1}RR^\top K_{zz}^{-1}K_{zx_*})$

for $i = 1..N_*$ **do**

▷ Sample the latent variables relevant to each test point.

$\boldsymbol{s}_i = \phi(y_i, \text{sample}[q(\boldsymbol{f}_{i::N_*})])$ ▷ Sample K times.

end for

end procedure

| Algorithms for Training and Prediction in the Multi-hop 'graph' Algorithm

Algorithm 3 Multi-hop 'Graph' Algorithm

Inputs

Observation time, sensor id pair, $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}$, so $\boldsymbol{X}$ is $(N \times 3)$;

Observation pair values, $\boldsymbol{Y} = \{[y_{i1}, y_{i2}]\}_{i=1}^{N}$, so $\boldsymbol{Y}$ is $(N \times 2)$;

Window size, δ;

Edge 'distances', connect by colocation events and over time, $d_{\text{colocation}}$, d_{time};

Reference sensors, $\boldsymbol{r} = \{0, 1\}^{S}$.

Outputs

Dictionary of scaling factors for (sensor,window) tuples $\boldsymbol{F}$.

procedure BuildGraph($\boldsymbol{X}, \boldsymbol{Y}, \delta, \boldsymbol{r}$)
  for w in all time windows W of width δ **do**
    for s_i,s_j in all pairs of sensors from X, where $s_i \neq s_j$ **do**
      $Y' \leftarrow Y_{X_{:,2}=s_i \wedge X_{:,3}=s_j \wedge X_{:,1} \in w}$ ▷ Selects observation pairs that are in the time window and between sensors s_i and s_j
      if $|Y'| \geq 5$ **then** ▷ We don't add edges where there are fewer than five observations
        G.addEdge($(s_i, w) \rightarrow (s_j, w)$, value=mean($log(Y'_{:,1}) - log(Y'_{:,2})$, distance=$d_{colocation}$))
      end if
    end for
  end for
  for w in all time windows $W - 1$ of width δ **do**
    for s_i in all sensors from X **do**
      G.addEdge($(s_i, w) \rightarrow (s_i, w + 1)$, value=0, distance=$d_{time}$)
      G.addEdge($(s_i, w + 1) \rightarrow (s_i, w)$, value=0, distance=$d_{time}$)
    end for
  end for
  for w in all time windows $W - 1$ of width δ **do**
    for s_i in all sensors from X **do**
      $P \leftarrow$ Shortest path (using Dijkstra) from (s_i, w) to any reference sensor, specified in r.
      if Path P exists **then**
        $F[s_i, w] = \sum_{p \in P} p_{value}$ ▷ Sum log ratios over path
      end if
    end for
  end for
end procedure

Algorithm 4 Multi-hop 'Graph' prediction algorithm

Inputs

Observation time, sensor id, $\boldsymbol{X}_* = \{\boldsymbol{x}_i\}_{i=1}^{N}$, so $\boldsymbol{X}_*$ is $(N_* \times 2)$;

Observed raw values, $\boldsymbol{y}_{\text{raw}}$, so $\boldsymbol{y}_{\text{raw}}$ is an $(N_* \times 1)$ vector;

Window size, δ;

Dictionary of scaling factors for (sensor,window) tuples $\boldsymbol{F}$, from Algorithm 3.

Outputs

Predicted calibrated values, $\boldsymbol{y}_*$, so $\boldsymbol{y}_*$ is an $(N_* \times 1)$ vector.

procedure Predict($\boldsymbol{X}_*, \boldsymbol{y}_{\text{raw}}, \delta, \boldsymbol{r}, G$)
  for $\boldsymbol{x}_*$ in $\boldsymbol{X}_*$; y_* in $\boldsymbol{y}_*$; and y_{raw} in $\boldsymbol{y}_{\text{raw}}$ **do**
    $w \leftarrow$ window computed for$[\boldsymbol{x}_*]_1$using window size δ
    if $([\boldsymbol{x}_*]_2, w) \notin F$ **then**
      Raise exception: No path to a reference sensor.
    end if
    $y_* \leftarrow e^{F[[\boldsymbol{x}_*]_2, w]} \times y_{\text{raw}}$
  end for
end procedure

Errors for predicting reference sensor without synthetic drift added

Figure 10 shows the errors in predictions for the low-cost sensor. Errors computed using a nearby reference sensor, while the predictions were made by relying on the network of colocations from a distance reference sensor.

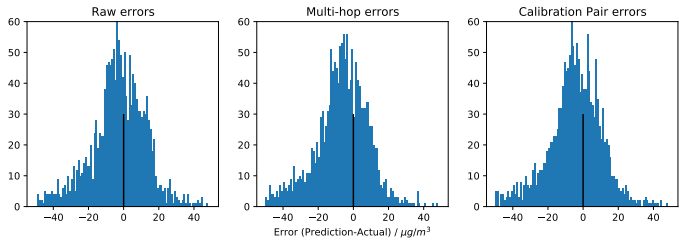

FIGURE 10 Errors in the predictions for the reference sensor in Kampala without any drift added.

Example Using Two Parameters

We provide a toy example to illustrate a two-parameter calibration function (the scaling and offset). Figure 11 shows the toy data (a simple sinusoid of observations: the low cost sensor is periodically offset, but not scaled).

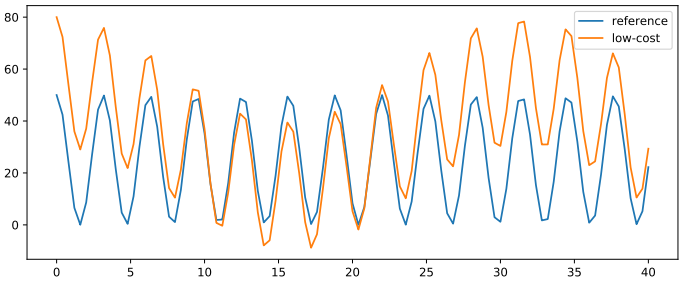

FIGURE 11 Demo data: The 'low-cost' sensor equals the reference measurements, except for a varying offset.

To show how one might use the available python module to fit this, we include a code sample in Algorithm 5. The key features are,

1. The simplicity of defining a calibration function. One just needs to replace the `transform_fn` and the associated log-gradient function `transform_fn_loggrad`. These take as inputs, samples of the parameters (`samps`), the raw measurements (Y) and other observations (`sideY`, e.g. humidity) that we might wish to use in our calibration function.
2. The choice of kernels for the two parameters. We assume here we have some prior information that the offset changes more quickly than the scaling. We use this to define a shorter lengthscale for the former, which means this is largely what the model uses for explaining the discrepancy between the two sensors.

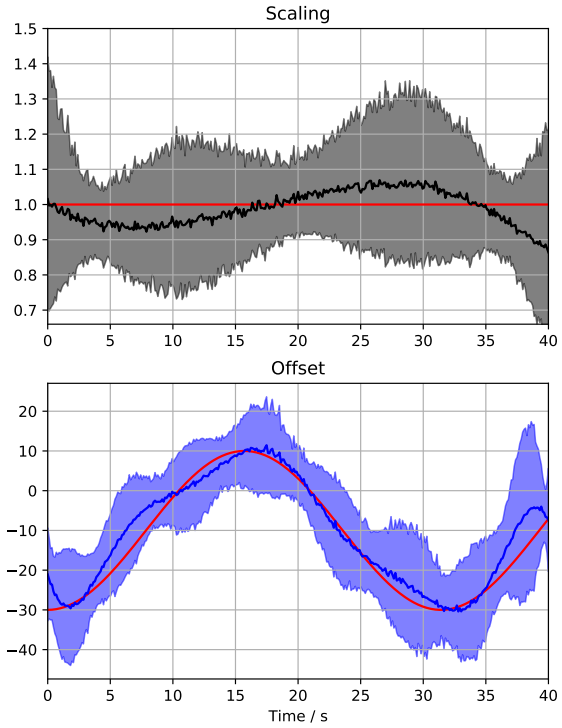

FIGURE 12 Demo predicted parameters (offset and scaling). The red lines in the two graphs show the true scaling and offset. The model can't distinguish between the two sources of discrepancy, given the limited data, which leads to large uncertainty. 95% credible intervals plotted.

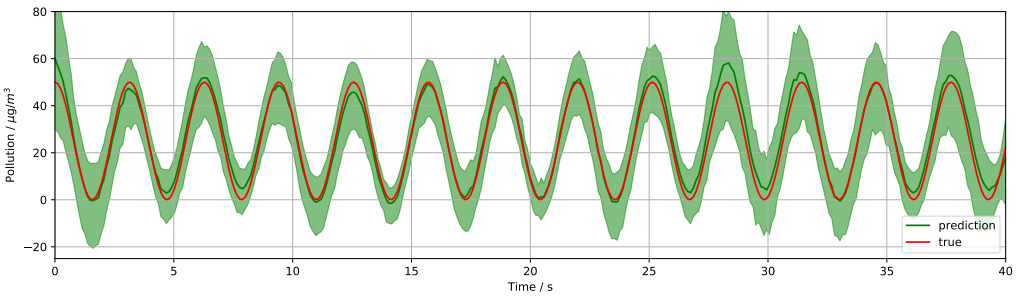

FIGURE 13 Demo predicted pollution. Red line is the true pollution. The green line shows the prediction (using the low-cost sensor). 95% credible intervals are plotted.

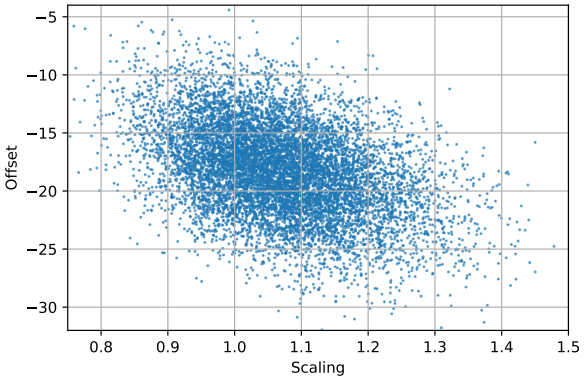

FIGURE 14 Plot of samples showing the (negative)correlation between the scaling and offset at t=25.828s. Because an exponential is used for the scaling parameter in the calibration function, the distribution is not Gaussian. The key feature is that, because we avoided using a mean-field approximation, these parameters can be correlated, allowing the model to represent a range of possible explanations for the data.

Figure 12 and 13 illustrate the estimates for the calibration parameters and the resulting prediction. Note that there is wide uncertainty in the estimates for the parameters. This is simply because there is a range of possible explanatory parameters that lead to similar predictions. The two plots hide that there is correlation between them. Figure 14 shows this (negative) correlation in the posterior, which means there is less uncertainty in the final prediction.

Algorithm 5 Python code example for toy data, using the calibration python module.

```
import numpy as np
from calibration import CalibrationSystem #our calibration module
import gpflow #uses gpflow kernels
import tensorflow as tf #uses tensorflow to compute gradients

########################################
###Generate Synthetic Data (in X and Y)

X = np.c_[np.linspace(0,40,101),np.zeros(101),np.ones(101)]
y = np.cos(X[:,0]*2)*25+25
Y = np.c_[y,y+20*np.cos(X[:,0]/5)+10]
refsensor = np.array([1,0]) #which sensors are reference sensors
Z = np.linspace(0,40,10)[:,None] #inducing points

#####################################
###Create and Optimise the Model

#Calibration Function
def transform_fn(samps,Y,sideY):
    return 10*samps[:,:,0:1] + Y*tf.exp(samps[:,:,1:2])

#Log derivative (wrt y) of calibration function
def transform_fn_loggrad(samps,Y,sideY):
    return samps[:,:,1:2]

#The kernels for the two parameters
ks = []
ks.append(gpflow.kernels.RBF(5.0,5.0)+gpflow.kernels.Bias(5.0)) #scaling
ks.append(gpflow.kernels.RBF(5.0,100.0)+gpflow.kernels.Bias(5.0)) #offset

#Which kernel to use for which parameters/sensors
kernelindices = [[0]*len(refsensor),[1]*len(refsensor)]

#Define the model
cs = CalibrationSystem(X, Y, Z, refsensor, 2, transform_fn,transform_fn_loggrad,
                       ks, kernelindices, likelihoodstd=0.001,lr=0.05,jitter=1e-3)

#Optimise the variational approximation
elbo_record = cs.run(its=200)
```

Algorithm 6 Python code example for making predictions.

```
#######################################
###To use the model to make predictions

from calibration import SparseModel
C = 2  #number of parameters
si = 1 #sensor to use for predictions

#build test observations matrix, and create the model object to use this
x = np.linspace(0,15,151)
testX = np.zeros([0,3])
for ci in range(C):
    tempX = np.c_[x,np.ones_like(x)*si,np.full_like(x,ci)]
    testX = np.r_[testX,tempX]
testsm = SparseModel(testX,cs.Z,C,cs.k)

#use the variational distribution parameters from the calibration system (cs.mu, cs.scale)
#to either give us the mean and covariance between the test points, or sample from the test
#points.
#qf_mu,qf_cov = testsm.get_qf(cs.mu,cs.scale)
samps = testsm.get_samples_one_sensor(cs.mu,cs.scale,num=1000)

#If we want to predict the true pollution, feed the low-cost observations
#into the calibration function
truey = np.cos(x*2)*25+25
testy = np.c_[truey,truey+20*np.cos(x/5)+10]
predy = transform_fn(samps,testy[:,1:2],None)
```