Parameterisation of Food Balance Sheet Uncertainty Distribution

Michael. C. J. Kao
Food and Agriculture Organization
of the United Nations

Abstract

In this paper, we briefly illustrate the use of the **faoswsFlag** package to parameterise the uncertainty distribution based on the confidence about a value.

Keywords: Uncertainty, Distribution.

1. Introduction

In preparing the Food Balance Sheet (FBS), one of the most indispenable yet difficult operation is the balancing mechanism. Due to the imperfection of data collection, estimation and imputation in the real world, it is the norm that the FBS is unbalanced and does not satisfy the equality constraint as first sight. Thus, a balancing mechanism is essential for satisfying the equality between the demand and supply of the FBS.

In current practice, the choice of a variable as a balancing item often reflects the availability of data (or the lack of data), rather than a clear economic rationale and empirical evidence. It is therefore not surprising that different SUA compilers/SUA approaches have chosen different variables as their balancing items. USDA's balances, for instance, use feed (and residual use) as the balancing item, while the FBS often use food to balance supply and demand. Conveniently, the XCBS approach often chooses whatever variable is not explicitly available. Clearly, none of these approaches renders a satisfactory solution to the problem and, no matter what variable is used as the balancing item, this variable is fraught with the measurement errors of all other variables. Given the fact that there is no a priori reason to assume that the measurement errors cancel out, the balancing item is bound to be the most inaccurate variable of the balance. Extending the logic to the Food Balance Sheets, using food as the balancing item would therefore be the least suitable solution.

Problems associated with the current approach motivated the research team to seek a method in where inequality occuring in the FBS to be allocated to various elements based on a logical reasoning rather than arbitrary allocation.

One method to handling the problem is to balance the FBS on a probabilistic basis. Each element is assumed to have a certain level of uncertainty, and the allocation of the imbalance will depends on the uncertainty of each element. That is, the smaller the uncertainty we have with a particular element, the less we should apportion the imbalance to that particular element. In the extreme case where we have perfect certainty about an element, than no imbalance should be added to the element.

In order to proceed with the probabilistic balancing mechanism, formulation of distribution for each of the elements in the FBS is necessary. The specification of the distributions undermines the validity of the balancing mechanism, and thus a consistent and logical construction of the distribution is crucial. The distributions should reflect the underlying uncertainty associated with each elements in order for the balancing mechanism to seek the optimal solution under the equality constraint of the FBS.

A framework guiding the specification of the distributions and the parameterisations is crucial. The absence of such framework generates inconsistency and paradox, further the use of the term probability maximisation is a disguise for a procedure which does not bear any meaning.

The paper is organised as follow. An introduction to the method and some brief theory will be provided. we will then provide illustrations of how the package provide a consistent framework for parameterise the uncertainty distribution of the FBS.

A corresponding table representing the weights or confindence of various data sources is assumed to have been constructed. For more details on the construction of the flag table, please refer to the faoswsFlag vignette.

2. The Data

```
library(faoswsFlag)
library(truncnorm)

## Create new flag table

exampleFlagTable =
    data.frame(flagObservationStatus = c("", "E", "I"),
               flagObservationWeights = c(1, 0.2, 0.6))

## Create artificial example for Supply and Utilisation Account
exampleSUA.df =
    data.frame(product = c("wheat", "wheat flour"),
               productionValue = c(100, 96),
               productionFlag = c("", "E"),
               importValue = c(10, 0),
               importFlag = c("", ""),
               exportValue = c(50, 0),
               exportFlag = c("", ""),
               foodValue = c(0, 80),
               foodFlag = c("E", "E"),
               seedValue = c(20, 0),
               seedFlag = c("", "E"),
               wasteValue = c(20, 16),
               wasteFlag = c("I", "E"),
               foodManuValue = c(100, 0),
               foodManuFlag = c("E", ""))

## Create artificial example for the unbalanced Food Balance Sheet
exampleFBS.df =
    data.frame(product = "wheat and products",
               productionValue = 220,
               importValue = 10,
               exportValue = 50,
               foodValue = 100, seedValue = 20,
               wasteValue = 40,
               foodManuValue = 100)
```

3. The Methodology

After observing a single value, how does one imposes a probability distribution that is consistent? In order to impose a distribution, we will need to impose something which reflects the uncertainty of the value we have observed. Similar to assigning a probability to whether it will rain tomorrow, we can assign a probability to the observed value reflecting our knowledge, experience and other information that is not explicitly incorporated. After assigning the probability to the observed value, we then have sufficient information to specify the distribution.

3.1. Quantifying Uncertainty

To measure a piece of information, one can use the formula of self information which is a measure of the information content defined as follow,

$$I = -ln(P) \; P \in [0, 1] \tag{1}$$

Where P is the probability or the confidence about the accuracy of the value assigned to the observed value in the first place. The natural logarithm is adopted here, but logarithm of any base can be used. This is a measure of the uncertainty condition on the confidence we have about the observed value. The log ensures that the uncertainty can be added across different items. The greater the I, the larger the associated uncertainty, that is, the lower the confidence we assign to the particular value, the higher the uncertainty. When the confidence is 0, or with 0% certainty, then I is infinite or infinite uncertainty; on the other hand, when the confidence is 1 then the the value is known with certain.

The logarithm also ensures that the uncertainty is additive. Essentially, the sum of the uncertainty is the log of the products of the probabilities assigned to the values. That is, it is the log of the joint probability assuming independence.

By assuming that the observed value is the expected value (the expected value here refers to the value with the highest probability, that is, the mode) and the self information as the expected information or entropy of the distribution, the parameterisation of any chosen distribution then follows naturally. Given the level of uncertainty associated with each element, then regardless of the choice of distributions, one can always parametrize the distribution where the uncertainty is held the same. This provides a consistent framework for specifying distributions in which the uncertainty for each element is consistent and relative amongst all elements.

The following illustration provides an example of how this framework can provide consistent parameterization of various distribution while maintaining the same level of uncertainty with the value. Take the wheat production example in the introduction section again, when we have 80% confidence about the observed value of 22,000 tonnes of wheat production in Australia, then the amount of uncertainty given the provided information can then be calculated as follow

$$I = -ln(0.8) \approx 0.2231 \tag{2}$$

In order to preserve this uncertainty associated with this piece of information, we need to preserve the entropy of the distributions. That is, regardless which distribution we choose we need to parameterise the distribution such that the entropy is equivalent to the same nat of information available.

The main reason to use the entropy rather than the standard deviation is because it is unit free and does not depends on the size of the value. If we were to impose uncertainty between two values, then the uncertainty associated with both value should be set respectively to the confidence given independent of the magnitude of the value. For example, if we we have observed 2000 tonnes of wheat production and 1000 tonnes of food while the confidence in the two value are identical, then the balance should be 1500 tonnes of production and food. If we were to base the uncertainty on standard deviation or percentage of variation, then the larger value will have large standard deviation of variation based on percentage and thus the final value will be closer to 1000 even though we have equal confidence in both values.

3.2. Parameterise Distribution Given Uncertainty

To parameterise a given distribution provided the uncertainty computed above is a simple task.

One simply choose the distribution then set the entropy equivalent to the uncertainty and solve for the parameters of the distribution.

That is, we solve for the following identity.

$$I = H \tag{3}$$

Where I is the self-information or uncertainty computed, and H is the entropy of the chosen distribution.

Take the normal distribution for example which has the following entropy equation:

$$H = \frac{1}{2} ln(2\pi e \sigma^2) \tag{4}$$

We can see the only parameter of the normal distribution that governs the entropy is σ or the standard deviation. Then Substitude H with I and solve for σ, we obtain the following.

$$\sigma = \sqrt{\frac{e^{2I}}{2\pi e}} = \sqrt{\frac{e^{2(-ln(0.8))}}{2\pi e}} \approx 0.3025 \tag{5}$$

or simply,

```
parameterise(obsValue = 22000, selfInformation = -log(0.8), distribution = "nor-
mal")


## $mean
## [1] 22000
##
## $sd
## [1] 0.3024634
```

That is, with the given observed value and confidence, the associated distribution is:

$$W \sim N(22,000, 0.3025) \tag{6}$$

Following the same example, we can solve for the parameterisation of a Cauchy distribution given the same level of uncertainty.

$$\gamma = e^{I-ln(4\pi)} = e^{-ln(0.8)-ln(4\pi)} \approx 0.0995 \tag{7}$$

and,

$$W \sim Cauchy(22,000, 0.0995) \tag{8}$$

```
parameterise(obsValue = 22000, selfInformation = -log(0.8), distribution = "cauchy")


## $location
## [1] 22000
##
## $scale
## [1] 0.09947184
```

There are no analytical solution for the parameterisation of the truncated normal distribution, however, a numerical solution is provided by the package.

$$W \sim trN(22,000, 0.3025) \tag{9}$$

```
parameterise(obsValue = 22000,
             selfInformation = -log(0.8),
             distribution = "truncNorm")


## $mean
## [1] 22000
##
## $sd
## [1] 0.3024757
```

We can see that the standard deviation of the truncated normal is marginally larger than the normal distribution above. This is due to the fact to maintain the same level of uncertainty while reduced support space, one has to increase the standard deviation.

4. Illustration

Take the example data, the first step is to construct the uncertainty of each FBS element based on the flags in the SUA.

```
## Select all the flag columns
flagColumns = grep("Flag", colnames(exampleSUA.df), value = TRUE)

## First we convert the flags to weights or confidence
(weightsSUA.df = data.frame(lapply(exampleSUA.df[, flagColumns], flag2weight)))


##   productionFlag importFlag exportFlag foodFlag seedFlag wasteFlag foodManuFlag
## 1           1.00          1          1     0.75     1.00      0.50         0.75
## 2           0.75          1          1     0.75     0.75      0.75         1.00


## Then we compute the self-information
(selfInfoSUA.df = data.frame(lapply(weightsSUA.df, selfInformation)))


##   productionFlag importFlag exportFlag  foodFlag  seedFlag wasteFlag
## 1      0.0000000          0          0 0.2876821 0.0000000 0.6931472
## 2      0.2876821          0          0 0.2876821 0.2876821 0.2876821
##   foodManuFlag
## 1    0.2876821
## 2    0.0000000


## Then we sum up the self-information for each element to obtain
## the uncertainty of each element.
totalInfo.df = data.frame(lapply(selfInfoSUA.df, sum))
```

To create the distribution, we simply prodive the function ‘distributionise‘ the observered value, the total information computed from the flag andthe desired distribution.

The function returns a list of two object. The first is the distribution function with the parameters computed, while the second object is a list with the corresponding values of the parameter.

```r
## Parameterise the production element with a Normal distribution
distributionise(obsValue = exampleFBS.df$productionValue,
                selfInformation = totalInfo.df$productionFlag,
                distribution = "normal")


## $pdf
## function (x)
## dnorm(x, mean = mean, sd = sd)
## <environment: 0x2c22b98>
##
## $parameters
## $parameters$mean
## [1] 220
##
## $parameters$sd
## [1] 0.3226276


## Parameterise the production element with a Truncated Normal distribution
distributionise(obsValue = exampleFBS.df$productionValue,
                selfInformation = totalInfo.df$productionFlag,
                distribution = "truncNorm")


## $pdf
## function (x)
## dtruncnorm(x, a = 0, b = Inf, mean = mean, sd = sd)
## <environment: 0x2b71480>
##
## $parameters
## $parameters$mean
## [1] 220
##
## $parameters$sd
## [1] 0.3226171
```

```r
productionDist =
    distributionise(obsValue = exampleFBS.df$productionValue,
                    selfInformation = totalInfo.df$productionFlag,
                    distribution = "truncNorm")


importDist =
    distributionise(obsValue = exampleFBS.df$importValue,
                    selfInformation = totalInfo.df$importFlag,
                    distribution = "truncNorm")
```

```r
exportDist =
    distributionise(obsValue = exampleFBS.df$exportValue,
                    selfInformation = totalInfo.df$exportFlag,
                    distribution = "truncNorm")


foodDist =
    distributionise(obsValue = exampleFBS.df$foodValue,
                    selfInformation = totalInfo.df$foodFlag,
                    distribution = "truncNorm")


seedDist =
    distributionise(obsValue = exampleFBS.df$seedValue,
                    selfInformation = totalInfo.df$seedFlag,
                    distribution = "truncNorm")


wasteDist =
    distributionise(obsValue = exampleFBS.df$wasteValue,
                    selfInformation = totalInfo.df$wasteFlag,
                    distribution = "truncNorm")


foodManuDist =
    distributionise(obsValue = exampleFBS.df$foodManuValue,
                    selfInformation = totalInfo.df$foodManuFlag,
                    distribution = "truncNorm")


ll = function(x){
    -log(productionDist$pdf(x[1])) -
        log(importDist$pdf(x[2])) -
            log(exportDist$pdf(x[3])) -
                log(foodDist$pdf(x[4])) -
                    log(seedDist$pdf(x[5])) -
                        log(wasteDist$pdf(x[6])) -
                            log(foodManuDist$pdf(x[7]))
}

constraint = function(x){
    x[1] + x[2] + x[3] - x[4] - x[5] - x[6] - x[7]
}

library(Rsolnp)
## Double check this solution, the resulting likelihood is infinite
balancedFBS =
    solnp(pars = as.numeric(exampleFBS.df[, -1]),
          fun = ll,
          eqfun = constraint,
          eqB = 0,
          control = list(tol =1e-10))


##
## Iter: 1 fn: 1e+24  Pars:  206.73973   9.97260  49.31507 102.73973  20.10959  40.43836 102
## Iter: 2 fn: 1e+24  Pars:  206.73973   9.97260  49.31507 102.73973  20.10959  40.43836 102
## solnp--> Completed in 2 iterations


balancedFBS$par


## [1] 206.739726   9.972603  49.315068 102.739726  20.109589  40.438356 102.739726
```

5. Conclusion

Affiliation:
Michael. C. J. Kao
Economics and Social Statistics Division (ESS)
Economic and Social Development Department (ES)
Food and Agriculture Organization of the United Nations (FAO)
Viale delle Terme di Caracalla 00153 Rome, Italy
E-mail: michael.kao@fao.org
URL: https://github.com/mkao006/sws_r_api/tree/master/faoswsFlag