BnpSysRel Vignette

Jackson Curtis

2019-02-26

Introduction

The goal the the BnpSysRel is to enable the estimation of system reliability for complex systems using the beta-Stacy process. In this vignette we walk through the basics of how an analysis would proceed, but for further theoretical details, see the Masters Project PDF attached to this repository.

In this vignette we will walk through estimating reliability for a simple system and how to organize your data in order to use the functions appropriately.

Our system

The function `estimateSystemReliability()` is going to do most of our heavy lifting here. This function takes three arguments. Once you know how to set up those three arguments you'll be ready to analyze your own data. The first argument `file` is looking for a string that will be the location of a text file to define how your system's components relate to each other. For this example, we will consider a simplistic version of a bike as our system of interest. Only a couple things can go wrong on this bike: either the front or the back tire can pop, or the front and the back brake could brake. If either tire breaks the bike becomes unusable, but because we like to live on the edge we will assume we only need one brake for the bike to function. In reliability, this is known as having the tires in series and the brakes in parallel (a redundant relationship). The tires and brakes are subsystems made of several components, and they are also in series.

The File

Let's look at how this text file should be written to tell R how our system works. We will use R to create the file, but you could just use your favorite text editor.

```
require(BnpSysRel)
```

```
## Loading required package: BnpSysRel
```

```
file="MyBikeSystem.txt"
write.table("S(BackTire, FrontTire):Tires
P(BackBrake, FrontBrake):Brakes
S(Tires, Brakes):Bike", file=file, quote=F, row.names=F, col.names=F)
```

So what did we just do? The text file contains three lines. Each line will begin either with an S or a P, indicating whether it is specifying a parallel or series relationship. After, the names inside the parenthesis indicate what parts make up the component after the colon, e.g. Tires is made up of BackTire and FrontTire. The syntax is meant to be pretty simple, but the complete grammar rules can be found by reading through the documentation for `estimateSystemReliability`.

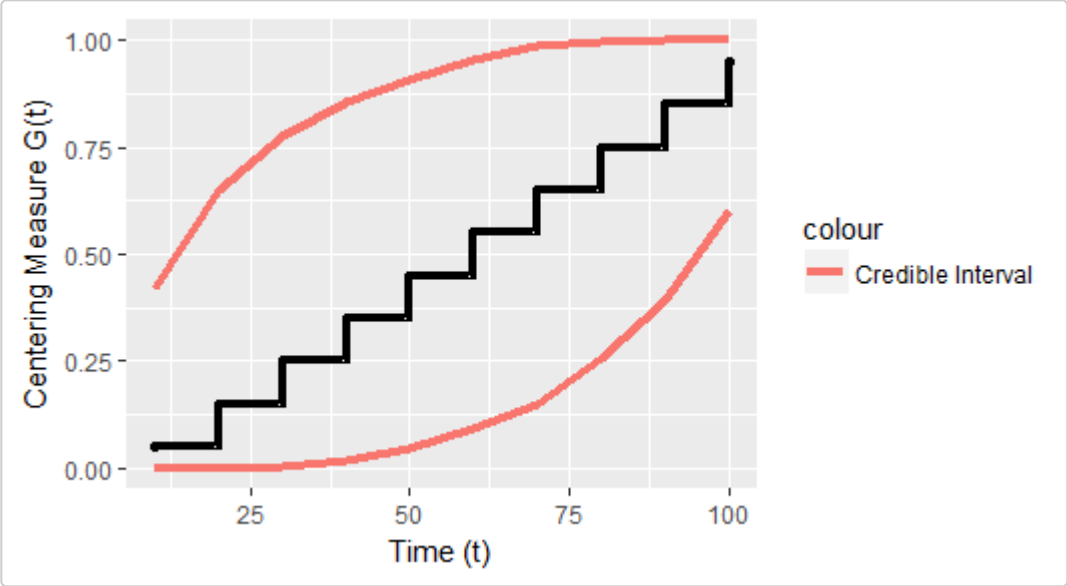
The priorList

So now that we've defined our system, we're done with the first argument to the function. Because our method is Bayesian in nature, the next argument, `priorList` specifies the priors we want to use on the components. Each prior in our model is a beta-Stacy Process, which is how we maintain conjugacy and get beta-Stacy posteriors, and the priorList is just a named list with a bunch of betaStacyProcess objects. So let's look at how to create a betaStacyProcess object:

```
tirePrior<-bsp(support = seq(10, 100, length.out = 10),
               centeringMeasure = seq(.05, .95, length.out = 10),
               precision = 3)
```

The `bsp()` function creates betaStacyProcess objects. Again, you'll have to read the paper for the more technical details, but we are basically defining our best guess as to what the CDF of the component should look like, in a discrete way. `support` is the list of jumps we want to see in our prior, `centeringMeasure` is how large those jumps should be, and `precision` is a measure of how certain we are in that CDF. We can look at our prior belief in what the CDF looks like:

```
plot(tirePrior, withConfInt = T)
```



We expect the CDF to be like the black line, but because we set a low precision (equivalent to measuring three failure times), our confidence is a very wide interval. Essentially, we're saying that at time t=50 (months maybe) we think somewhere between 5% and 90% of tires will have failed.

Now we do the same thing for the other components. We don't need priors for brakes, tires, or bike because we use the posterior of the subcomponents as the prior for the subsystem.

```
brakePrior<-bsp(support=c(22,44,66), centeringMeasure=c(.25,.5,.75), precision = 1)
priorList=list(FrontTire = tirePrior, BackTire=tirePrior,
               FrontBrake = brakePrior, BackBrake = brakePrior)
```

We're all done! Of course we don't have to assign the back tire and front tire the same prior, but for simplicity we will here.

The dataList

The dataList is the exact same format as the priorList, but instead of betaStacyPrior objects as elements of the list, we provide it data matrices. The data we collect is a bunch of failure times indicating how long it took the system to fail, which can be right-censored (meaning we ended the test before failure). Let's say we tested five back tires, the last two of which were censored:

```
backTireData<-matrix(c(14, 1,
                       29, 1,
                       67, 1,
                       75, 0,
                       75, 0), byrow = T, nrow=5)
```

The first column indicates the time recorded for each experiment, the second column is a 0/1 indicator for whether the observation was censored or not (1 if it was fully observed, 0 if it was censored). We can do the same for any component we have measurements on! If we don't have measurements, we will pass the computed prior through as the posterior. Let's finish measurements (with no more censoring).

```
set.seed(20)
frontTireData<-cbind(rnorm(50, 50, 10), 1)
backBrake<-cbind(rnorm(30, 20, 5), 1)
frontBrake<-cbind(rchisq(40, 20), 1)
bike<-cbind(c(40,47,42,46), 1)
dataList<-list(BackTire=backTireData,
               FrontTire=frontTireData,
                BackBrake=backBrake,
                FrontBrake=frontBrake,
                Bike=bike)
```

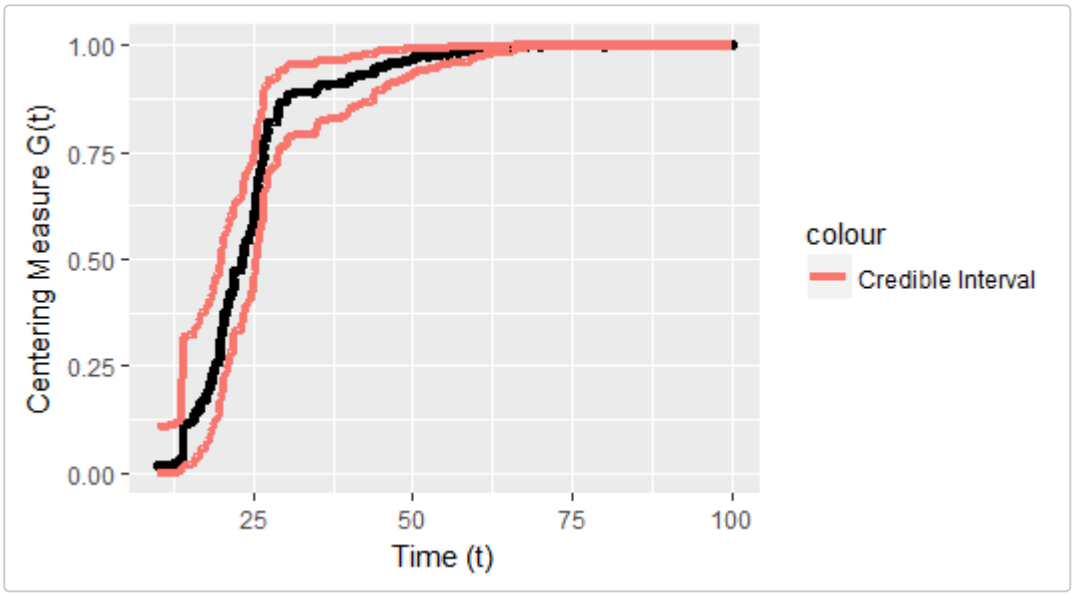
Now we have a list of our data matrices, and we are ready to use the function.

```
posteriors = estimateSystemReliability(file=file, priorList=priorList, dataList=dataList)
```

Analyzing the results

The function returns a posterior for every component in the system. We could explore and make inference on any of these components, but we will focus on the bike. Let's plot it:

```
plot(posteriors$Bike, withConfInt=TRUE)
```



Now we can get a point estimate and credible interval on the percentage of bicycles that will have failed by t=20 and t=50 months using the `evaluate_centering_measures()` and `bspConfint()` functions:

```
evaluate_centering_measures(posteriors$Bike, times=c(20,50))
```

```
## [1] 0.3283582 0.9701220
```

```
bspConfint(posteriors$Bike, times=c(20,50), conf.level = .05)
```

```
##              20        50
## 2.5%  0.1809604 0.9332901
## 97.5% 0.5167226 0.9931072
```

At t=20 our point estimate for the proportion of bikes that will have failed is 32.84% with a 95% credible interval of (18.1%, 51.7%).

Similarly we can calculate when we would expect the first 10% and 90% of bikes to have failed by using the quantile function:

```
quantile(posteriors$Bike, probs=c(.1,.9))
```

```
##      10%      90%
## 13.59176 34.78649
```

So 14 months in we expect 10% of the bikes to have failed and 35 months in we expect 90% of the bikes to have failed.

One last function that might be useful is that if you don't have an entire system, but just a single component, you can use the `bspPosterior()` function to calculate one posterior at a time. Its arguments are the same–one prior and one data matrix!

```
BackTirePosterior=bspPosterior(priorList$BackTire, dataList$BackTire)
```

We can see that they’re the same whether regardless of which function we use:

```
print(posteriors$BackTire)
```

```
##    support centeringMeasure precision
## 1        0        0.0000000  8.000000
## 2       10        0.0187500  8.000000
## 3       14        0.1437500  8.000000
## 4       20        0.1812500  8.000000
## 5       29        0.3062500  8.000000
## 6       30        0.3437500  8.000000
## 7       40        0.3812500  8.000000
## 8       50        0.4187500  8.000000
## 9       60        0.4562500  8.000000
## 10      67        0.5812500  8.000000
## 11      70        0.6187500  8.000000
## 12      75        0.6187500  2.754098
## 13      80        0.7276786  2.754098
## 14      90        0.8366071  2.754098
## 15     100        0.9455357  2.754098
```

```
print(BackTirePosterior)
```

```
##    support centeringMeasure precision
## 1        0        0.0000000  8.000000
## 2       10        0.0187500  8.000000
## 3       14        0.1437500  8.000000
## 4       20        0.1812500  8.000000
## 5       29        0.3062500  8.000000
## 6       30        0.3437500  8.000000
## 7       40        0.3812500  8.000000
## 8       50        0.4187500  8.000000
## 9       60        0.4562500  8.000000
## 10      67        0.5812500  8.000000
## 11      70        0.6187500  8.000000
## 12      75        0.6187500  2.754098
## 13      80        0.7276786  2.754098
## 14      90        0.8366071  2.754098
## 15     100        0.9455357  2.754098
```