BACHELOR

Planning preventive maintenance using Bayesian inference

van der Moore, Joris H.H.

Award date:
2020

TU/e Department of Computer Science & Mathematics

PLANNING PREVENTIVE MAINTENANCE USING BAYESIAN INFERENCE

Bachelor end project

J.H.H. van der Moore
0957504

November 17, 2020

Abstract

Monitoring components can be a very beneficial practice in many fields of industry. Many components can be preventively maintained, often less expensive than replacing the component. Analysing the data a sensor can provide can cut tremendous amounts of maintenance costs. In this paper, the different techniques of analysing and predicting the life of a component using Bayesian inference will be discussed. This information can be used to plan preventive maintenance and estimate the total expected cost of the component.

Contents

1 Introduction

Preventive maintenance is becoming a bigger part in modern companies since it can greatly reduce costs. In this paper, the optimal policy on when to perform preventive maintenance will be determined. This can then be used to find the total expected cost of a component.

We assume there is a component with a sensor attached. This sensor can continuously give information of the state of the component. The increments of the sensor will be assumed to be normally distributed. The sample path of the sensor is therefore a Brownian motion. During this research, the main focus is to establishing a structured replacement policy for single-unit components.

Similar research is done by Elwany et al. (2011) [1] and Gebraeel et al. (2005) [2]. Later on this research will be compared to the results of Elwany [1].

First off, having a certain policy will have certain expected costs associated to it. This will be analysed in section 5.

Besides the cost of a certain policy, the optimal policy can be found. This is done by calculating the so-called value function. This function can be used to find the optimal policy and use that to find the total discounted cost of a component.

In the first part of the report, full knowledge of the sensor is assumed. In practice, this is not the case and all sensors behave a bit different. This is implemented in the second part of the report where only the signal of the sensor is known every time the component is checked.

The model will then be tested and evaluated. After that, some example graphs will be shown and explained. Finally, the conclusions can be drawn.

2 Problem definition

In this report, a detailed look will be given at the timing of maintenance of a component. At every checkup, there is a choice to either preventively maintain the component, or to do nothing and hope the component is still functioning properly when the next check occurs.

In the first part of the report, the sensor is assumed to be a Brownian motion, with known parameters. Although the Brownian motion is a continuous time stochastic process, it can only be checked at set intervals. With this knowledge, the estimated cost of a certain policy is calculated. This is done by numerically modelling the component.

The Brownian motion is a reliable estimate of a signal as it is used by multiple other researchers [3, 4, 4, 5]. As such, the Brownian motion will be used to model the signal of the component. During this research, the results will be compared to the results of Elwany (2011) [1].

Note that the optimal policy will depend on the length of the interval. When the component is continuously checked, the optimal policy will be to maintain the component just before it breaks down. In this report, the Brownian motion is discretized with time steps of Δt. The optimal policy therefore depends on $\Delta t > 0$. In order to get a realistic answer Δt has to be significantly greater than 0.

After that, there will be a section testing the findings of the optimal policy for maintaining the component. This is done by modelling the findings from the previous model to find the best policy. After that the outcomes can be checked whether they are the same.

When using this model in the real world, the exact parameters of the Brownian motion are, of course, not known. This is because the usage and degradation of a component can only be estimated. Although they are not known exactly, they can be estimated using previously observed data. This can give a good estimation for the exact parameters. These will of course be tested and their error will be quantified.

After this is done, the bigger picture can be seen again: What is the optimal time to maintain a component when the parameters of the sensor are not known, and what is the expected discounted cost?

There will be a detailed approach to this problem, since it's answer is extensive and complex.

3 Assumptions

Over the course of this report, the models have to be simplified so that they can actually be evaluated. Many of the assumptions are straightforward, but there are some which might not be that obvious. Below, the assumptions will all be stated and explained.

The first assumption is that every check is done at a regular interval. That means a brand new component will be checked as frequently as one nearly failing. This interval is set at the beginning at does not change.

A second assumption is that the sensor in the component accurately describes the state of the component. When the sensor hits the critical point ξ, the component is broken and cannot be used. The sensor of the component is modelled by a Brownian motion and thus can go below zero. Every time-step the difference in the sensor is a normal distribution with known parameters. Later in the report, the assumption that the parameters are known is left out, and the parameters are estimated by the previous observations.

Another assumption is that, at such a check, the only options are totally maintaining the component, or doing nothing. There is no such thing as maintaining part of a component. This has the consequence that the signal can only be reset to 0 (when maintaining), of increased by a random, normally distributed, amount.

Besides that, the time and costs for maintaining and repairing a component are different, with the premise that repairing a component takes longer and is more expensive than preventive maintenance. This has the underlying assumption that planned preventive maintenance is cheaper than unexpected reparation. The time for maintenance and reparation are denoted by T_m and T_r. The costs associated with those actions are denoted by C_m and C_r and do not change over time.

The costs of repairing and maintaining a component are also continuously discounted using a continuous discounting factor with parameter β. This does not change over time and every timestep the discounting factor is then given by $e^{-\beta \Delta t}$. Because of inflation, costs made in the future are discounted accordingly.

4 Approach

Given the complexity of this problem, the approach has to be detailed and clear. In order to use the Brownian motion, the underlying theories must be clear and understood. The first section will be about the needed knowledge to completely follow and understand this report.

After the information needed in order to understand this report is clear, the problem can be tackled. This is done in several ways. First, the estimated cost of a given maintenance-threshold can be calculated by simulation. This is the first step in fully understanding the problem. When this is done, it can be used to verify the results in the next sections.

After that, the value function can be defined. This is fully explained in the section 6. The value function denotes the expected cost when starting with a component with a signal x, $0 < x \leq \xi$. This can be tested and verified using the estimated cost of a given maintenance-threshold explained in section 5.

When these functions are explained and verified, the optimal time to maintain a component is found, and the first part of the problem is solved. However, this method assumes that the parameters for the increase in signal are known. In reality, these can never be known. They can however, be estimated using the data of the previously observed signal. This is done in several ways, and those will be compared to each other. This is done in the next sections.

Here, the properties of the component are not the same for every component. This means the parameters will have to be estimated again, every time the component is reset. Later on, the components are all assumed to be homogeneous. In this case, the knowledge of the parameters can be used over multiple multiple components, resulting in a better estimation of the actual parameters.

The hard part of estimating the parameters is that there are two unknown parameters. This makes it substantially more difficult to estimate them, then only estimating one. That is why the first part of this section is about estimating only one (μ or σ). Finally, some research will be spend examining whether both can be accurately estimated.

5 Simulating the cost of a policy

In order to find the optimal policy, it can be easy to start a bit smaller. Instead of finding the best time to maintain the component, we can estimate the cost of a specific policy. This will be done numerically. The model used in this approach is made in R, a statistical program.

This is done by first simulating the signal, which is made by sampling multiple values from the normal distribution with mean μ and variance σ^2. These are then added and the signal is simulated. In this part, all parameters of the Brownian motion are assumed to be known. This way a certain policy together with certain parameters will result in a certain cost. Later on, this assumption is left out.

The next thing to add to the model is taking action when it reaches the point of maintenance, θ, or fails as soon as it hits the breaking point, ξ. This can be modeled as shown in the figure below:

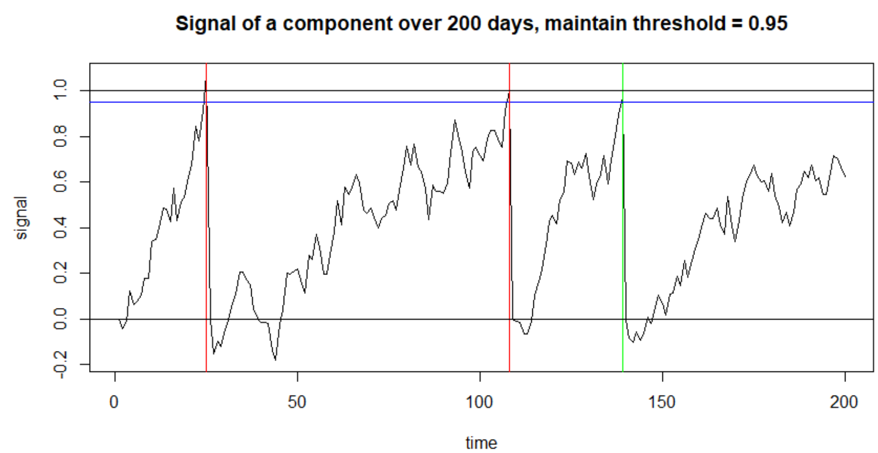


Figure 1: Sample path of the signal over 200 days. Component is repaired at $t = 25$, $t = 108$ (red) and maintained at $t = 139$ (green). (seed = 2)

When the signal is between θ and ξ, the component is maintained, and the signal is then reset and costs according to the action are added. Every timestep, the cost of repairing or maintaining is discounted. This is a correction for inflation. This way, a simulation can be run several times and eventually the expected cost for a certain policy will be presented.

The code for the model can be found in appendix A

Then, every time the simulation is run, every discrete timestep there is a check and one of three things can happen:

- $x < \theta < \xi$: The component is still functioning and does not require to be repaired or maintained.

- $\theta < x < \xi$: The component has reached the maintenance threshold, but is still functioning. Maintenance will be performed.
- $x > \xi$: The component is broken and has to be repaired.

If the component has to be maintained or repaired, the cost is discounted accordingly. Due to this fact, only simulating a finite time is very accurate. This is because after T days, the cost is discounted by $e^{-\beta T}$. Note that this is very close to 0, when T is large and β is fixed.

When the component is maintained or repaired, the simulated signal is lowered by the value of the signal at that moment. This will result in the signal resetting due to the increments being independent of each other.

Every different policy (signal at which maintenance is performed) is run $N = 10.000$ times. This is done to get an accurate mean of the cost, by the law of large numbers.

After several runs, the mean of the total discounted cost can be calculated. This is presented below.

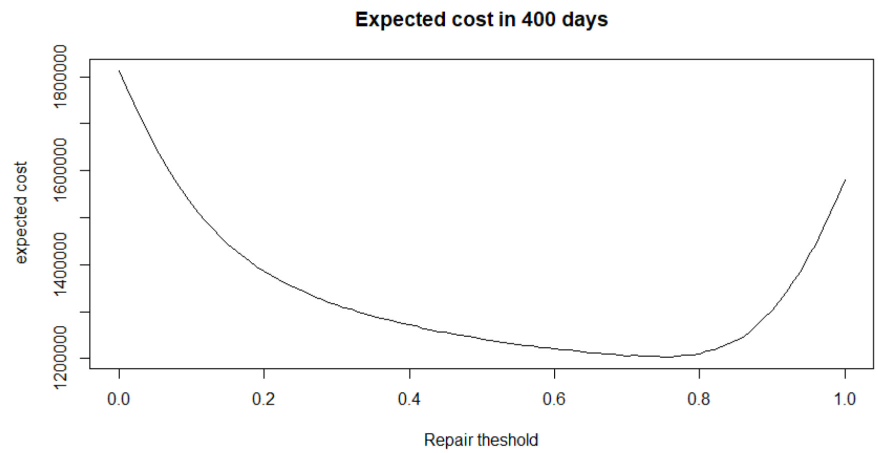


Figure 2: Total expected discounted cost for the next 400 days

Note that the parameters here are can be changed and the optimal θ will then differ. There is, however, always a steep increase in expected cost when θ is around 0 or 1. In this example, the minimum is attained at $\theta = 0.76$. This means that would be the optimal policy.

Using this method, a policy can be checked and a simulation can be run to find the optimal policy. However, this does not prove anything, but it can give useful insights and can be used to check the answers in the coming sections. In order to find a more rigorous proof, the value function will be analysed in the next section.

6 Value function

In the previous section, a model was shown which can be used to check a certain policy. This does not explain why the policy is a good one. In order to analyse this further, the value iteration function will be researched.

Now the value function, denoted by $V(x, t)$, can be analysed. This is the total expected discounted cost function.

The value function is defined to be cost of the optimal action to take, where the costs are the least, so if $x < \xi$:

$$V(x,t) = \text{min of:} \begin{cases} \text{Maintain, with cost} & C_{\text{PM}} + \mathbf{E}[e^{-T_m\beta}V(0,0)] \\ \text{Do nothing, with cost} & \mathbf{E}[V(x+\mu\Delta t + \sigma\epsilon(\Delta t))] \end{cases} \tag{1}$$

Where T_m and T_r denote the time it takes to perform maintenance or repair the component. C_{PM} denotes the cost associated to the preventive maintenance. Note that these are the only two options when the component is not yet broken, and thus does not need replacing.

When the component is broken, $(x > \xi)$, there is only one action, and that is to replace the component. The cost of this action is given by:

$$V^{(n+1)}(x,t) = C_R + \mathbf{E}[e^{-\beta * T_r}V^{(n)}(0,0)]$$

Note that the total cost will depend on the policy Π. Never maintaining the component will result in a different expected cost then maintaining it every day. The goal now is to determine an optimal strategy, minimizing the expected cost:

$$\min_{\Pi} V(0,0)$$

6.1 Bounds on the value function

The value iterated function, which will be numerically calculated in the coming sections can be bounded by two functions. The upper bound being the cost associated with never doing preventive maintenance, and the lower bound being the cost associated with always performing preventive maintenance, just before the component breaks down. A sample path of the signal would look like this, where the maintenance threshold is set at 0.95:

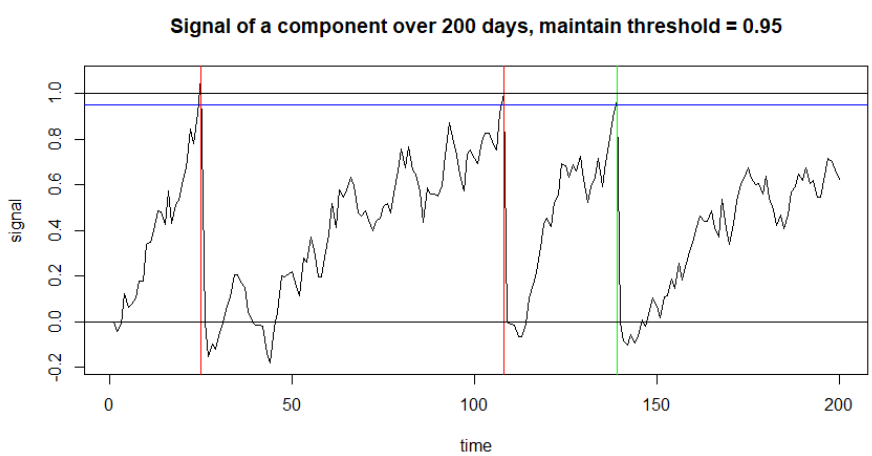


Figure 3: Sample path of the signal over 200 days. Component is repaired at $t = 25$, $t = 108$ (red) and maintained at $t = 139$ (green). (seed = 2)

From this, the expected cost of the lower bound can be calculated. The cost of the first maintenance is given by the expected value of $e^{-\beta T^{PM}}C_m$, and T^{PM} being the time until maintenance is needed (i.i.d. random variables and dependent on the set threshold), and β being the discount factor. Then, the cost of the second maintenance is again $e^{-\beta(T_1^{PM}+T_2^{PM})}C_m$. And the cost of the n'th maintenance will be $e^{-\beta(T_1^{PM}+T_2^{PM}+\ldots+T_n^{PM})}C_m$. Combining all of these costs will result in the expected cost.

This is given by:

$$
\begin{aligned}
V_T &= \mathbf{E}[C_m * e^{-\beta T_1^{PM}} + C_m * e^{-\beta T_1^{PM} + T_2^{PM}} + ... + C_m * e^{-\beta \sum_{i=1}^{\infty} T_i^{PM}}] \\
&= \mathbf{E}[\sum_{i=1}^{\infty} C_m e^{-\beta(i * T_1^{PM})}] \\
&= \mathbf{E}[C_m \sum_{i=1}^{\infty} e^{(-\beta T_1^{PM})^i}] \\
&= \mathbf{E}[C_m \frac{e^{-\beta(T_1^{PM})}}{1 - e^{-\beta(T_1^{PM})}}]
\end{aligned}
$$

And the upper bound is then given by:

$$
\mathbf{E}[C_r \frac{e^{-\beta(T^R)}}{1 - e^{-\beta(T^R)}}]
$$

This is the similar to the lower bound, but with the price of repairing the component, instead of maintaining it. Note that this is only a valid equation when $E|e^{-\beta(T)}| < 1$, but since $\beta > 0$ (inflation) and both $E[T^{PM}] > 0$ and $E[T^R] > 0$, this is always true.

6.2 Evaluating the value function

The value function $V(x, t)$ (equation 1) is hard to evaluate. This is because the expected value of the cost function at time $t + \Delta t$ further, is random. This is because the next step of $V(x, t)$ is $V(x + \mu\Delta t + \sigma\epsilon, t + \Delta t)$. Where $\epsilon \sim N(0, 1) = Z$ ($\Delta t = 1$ is assumed from now on). This means the outcome is random. The expected value can be calculated by integrating over all possible values of ϵ. This is done below:

$$
\begin{gathered}
\mathbf{E}[V(x + \mu\Delta t + \sigma\epsilon, t + \Delta t)] = \\
\int_{-\infty}^{\infty} V(x + \mu\Delta t + \sigma z, t + \Delta t) f(z) dz = \\
\int_{-\infty}^{\frac{\xi - x - \mu\Delta t}{\sigma}} V(x + \mu\Delta t + \sigma z, t + \Delta t) f(z) dz + \int_{\frac{\xi - x - \mu\Delta t}{\sigma}}^{\infty} V(x + \mu\Delta t + \sigma z, t + \Delta t) f(z) dz
\end{gathered}
$$

Note that the integral can be spit in two cases: if $z < \frac{\xi - x - \Delta t \mu}{\sigma}$, meaning the component is still functioning when Δt time has passed. And the other case where $z < \frac{\xi - x - \Delta t \mu}{\sigma}$, when the component is broken at that time. This can be used later to split the possible actions and numerically evaluate the value function.

Now that the formulas for the cost function are know, the simple question remains. What is the best strategy to do every time interval. This is just the action that costs the least: Do nothing or maintain the component.

Note that maintaining always costs the same: $C_m + \mathbf{E}[e^{-T_m \beta} V(0, 0)]$, and the cost of doing nothing is given by $\mathbf{E}[V(x + \mu\Delta t + \sigma\epsilon(\Delta t))$. Since $\mathbf{E}[V(x + \mu\Delta t + \sigma\epsilon(\Delta t))$ is increasing in x and $C_m + \mathbf{E}[e^{-T_m \beta} V(0, 0)]$ is constant, with respect to x, their minimum is also increasing in x. This means the value function is increasing with respect to x.

6.3 Numerical approach to the value function

In order to calculate the cost function, the integral explained above has to be evaluated. The problem here is that the integral domain is $D = [-\infty, \infty]$. This is impossible to numerically evaluate. That is why another approach to this problem has to be found.

This method uses an iterative function, where the first value is entered, and every iteration the values are updated. This converges to the exact answer. This relation is given by:

$$V^{(n+1)}(x,t) = min \begin{cases} C_m + \mathbf{E}[e^{-T_m \beta} * V^{(n)}(0,0)] \text{ (maintain)} \\ e^{-r\Delta t} * \mathbf{E}[V^{(n)}(x + \mu\Delta t + \sigma\epsilon(\Delta t), t + \Delta t)] \text{(do nothing)} \end{cases} \tag{2}$$

This function can be modeled in R, using a recursive function. As said earlier, the first iteration is just a guess of the correct price. After that, the values are updated and are closer to the real value. The code exists of three parts: a function that calculates the cost when no action is taken, a function which calculates the cost when preventive maintenance is performed, and a part where the minimum of those actions is chosen.

```
expectedCostDoNothing <- function(n,x,t){
 Numerically integrate the previous iteration of the cost
 function times the normal distribution associated to it
 and discount it

}

expectedCostMaintain <- function(n){
 return discounted starting cost V(n-1,0,0)
}

V <- function(n,x,t){

  if n=0,
    return initial value

  If not broken (x<xi & x>theta), do what will cost the least
    return minimum of maintaining and doing nothing

  If broken (x>xi), replace and pay repair costs
    return repair costs + discounted maintaining cost)
}
```

Now that the layout of the function is clear, the specific formulas for the expected cost can be implemented. This is done by calculating the expected cost for every possible outcome. In practice, this comes down to sampling several values of the normal distribution from -3σ, to 3σ. The region outside of this is negligible (0.99% is in between these values, $\Phi(3\sigma) > 0.99$).

This is done by sampling a value of the standard normal distribution every interval (small distance from each other) and finding the value accordingly. This can then be used to find the probabilities of the signal being that value after a time step. Finally, those values are divided by the sum of all those values, to have the total probabilities sum up to 1.

This discretization of a continuous function will lead to errors, which can be lowered by decreasing the interval width. This comes at a price since more computations are needed to calculate the value of the next iteration. In the code (appendix B) the interval can be decreased to increase accuracy.

Because of the recursive way of calculating the value iteration function, it takes quite some time to calculate the values at every point. Because the minimum is needed, both expressions have to be calculated. This is a very inefficient way of doing the calculations.

When the best option at a given value is to preventively maintain the component, all other values with a higher signal don't even have to be calculated. Up until the point where the component is broken, maintenance is the best option (because the value of doing nothing increases when x increases).

This knowledge can be used to drastically decrease the computations needed to calculate the value iteration function. Now up until the point where both actions have the same value, doing nothing is always better, and after the equality, performing the preventive maintenance is always better. This eliminates the need for the minimum of two values, which in turn requires less computations.

Besides fewer computations, this method makes use of the fast vector multiplications of R. Once maintaining is the better option at a signal level of say θ, it is the best option for all $\theta \leq x \leq \xi$. And since the possibilities of x are strictly discrete $(0, 1, ..., 999, 1000)$ for example, a vector can be made of all of those values. This means the first vector exists of $[V(0,0), V(0,1)...V(0,999)V(0,1000)]$ and the second with a 1 in the first place, denoting the amount of iterations. Updating these vectors many iterations will result in the vector $V^*(0), V^*(1), ..., V^*(999), V^*(1000)$.

Updating the vector is done in several parts. First, the elements below θ are updated. This corresponds to "do nothing". These are updated to the expected new value. This is done by the dot product of the possible value functions and their possibilities respectively.

$$V_1(0) = \begin{bmatrix} V_0(\mu\Delta t + 3\sigma) \\ V_0(\mu\Delta t + 3\sigma - 1) \\ \vdots \\ V_0(\mu\Delta t) \\ \vdots \\ V_0(\mu\Delta t - 3\sigma + 1) \\ V_0(\mu\Delta t - 3\sigma) \end{bmatrix}^T * \begin{bmatrix} \Phi(3\sigma) \\ \Phi(3\sigma - 1) \\ \vdots \\ \Phi(0) \\ \vdots \\ \Phi(-3\sigma + 1) \\ \Phi(-3\sigma) \end{bmatrix}$$

After that, the part where $x > \theta$, preventive maintenance is the better option. The updated cost of this is: $V^{n+1}(x) = V^n(0,0)e^{-\beta T_m} + C_m$ and the cost when $x > \xi$ is given by: $V^{n+1}(x) = V^n(0,0)e^{-\beta T_r} + C_r$.

Since the signal can also be negative, the first couple of terms of the vector are added and set equal to $V_n(0)$.

Finally, the estimated equilibrium has to be updated. This is done by checking whether the option of doing nothing, or preventively maintaining the component would be the best action.

This is done by checking the values of $V(\theta_n)$ and checking which is the optimal action. If performing preventive maintenance is the best action at that time, the threshold θ is lowered, and otherwise it is increased.

In pseudo-code it looks like the following:

```
for(every iteration){

    If do nothing is the optimal action (x<theta){
        update V by multiplying the possible
            outcomes times their probability
    }

    Set first elements equal to V(n,0,0)

    If maintaining is the optimal action (x<xi & x>theta){
```

```
        Set value to maintenance cost + discounted  V(n,0,0)
    }

    If broken already (x>xi){
        set value to repair cost + discounted  V(n,0,0)
    }

    If cost is lower than maintaining, increase threshold
    If cost is higher/equal to maintaining, lower threshold

}
```

This way, the threshold will end up at the point where maintaining is as expensive as doing nothing. For all signals below the threshold, doing nothing is the better option, and for all sample paths above that, preventive maintenance is preferred.

Note that, in order to calculate the expected value, a vector with the corresponding probabilities of the normal distribution has to be made. This vector is given by:

$$N = \begin{bmatrix} \Phi(3\sigma) \\ \Phi(3\sigma - 1) \\ \vdots \\ \Phi(0) \\ \vdots \\ \Phi(-3\sigma + 1) \\ \Phi(-3\sigma) \end{bmatrix}$$

Implementing this in R is done by having a vector with multiple $V(x)$ for each iteration. Finally they are all added to one big matrix which can be used to display the function. The code for this can be found in the appendix B.

7 Results of the optimization model

Now that the basic model is finished, the results can be analyzed. One thing to observe from the model is the graph which depicts the signal x over time, together with the optimal time to perform preventive maintenance, θ. This is shown below:

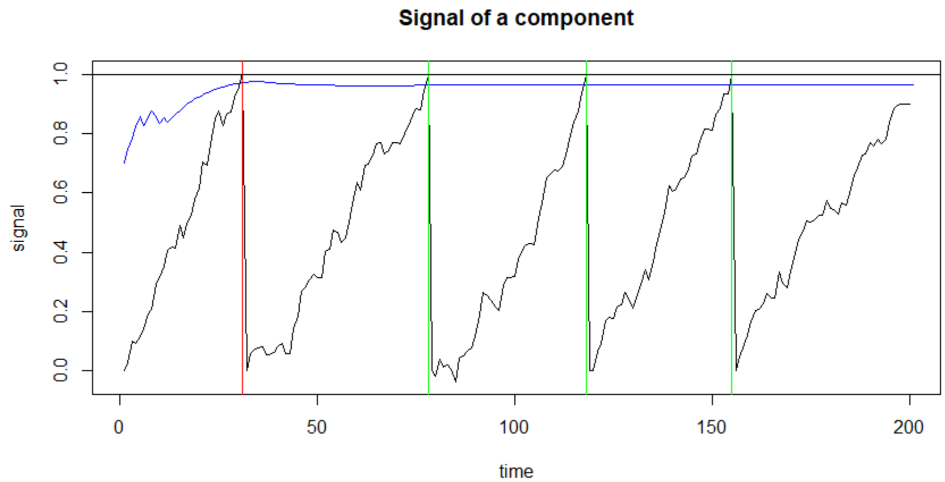


Figure 4: Signal of a component (black) with the optimal policy (blue) and the times where maintenance was performed (green) (seed = 2)

Here, it can be clearly seen that the initial guess of the optimal policy ($\theta = 0.7$) was too low. Since θ is updated every timestep, it quickly converges to the optimal point (≈ 0.95). The monotonicity in θ cannot be seen here because this is only the case when all other things stay constant, in this example, the signal is very low sometimes, resulting in a decrease in θ.

Note that this depends on a lot of factors and this is just an example. Times where maintenance was performed are indicated with the green vertical lines. When the component is completely repaired, a red line is drawn. The signal is also reset at those points.

Another interesting graph to check is the total expected discounted cost. This can be done in finite time since the total cost is discounted every time step, and finally, the discounting factor $e^{-\beta t}$ will get so close to 0 that it converges. The graph is shown below:

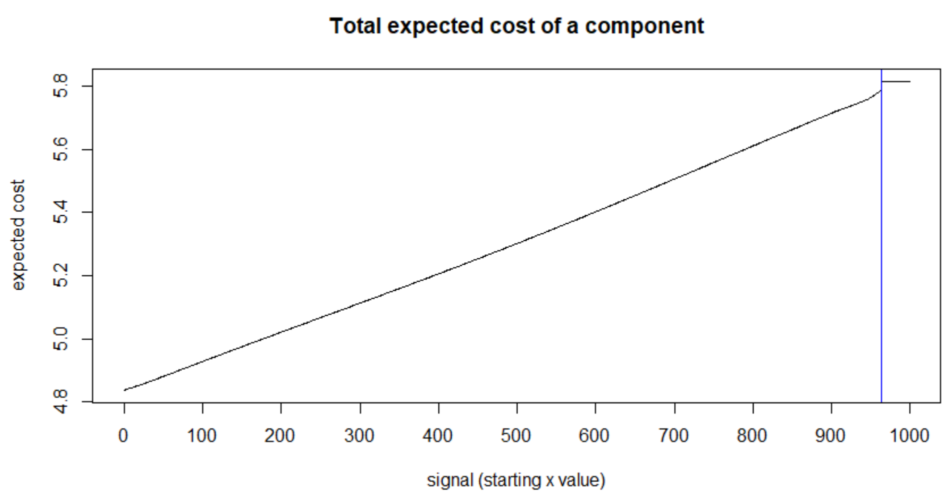


Figure 5: Total expected cost of a component (seed = 2)

This accurately depicts the total discounted cost of the component, starting with a value of x. It is cut off at a finite number of iterations, but since the discounting factor $e^{-\beta t}$ is very small, the extra costs are negligible.

8 Sensitivity analysis of the optimization model

In order to verify the correctness of the model, some test cases will be analyzed. If the model behaves as expected, it can be extended in the next sections.

In order to set a baseline for the tests, the parameters used will be shown in appendix B, together with the rest of the code.

This gives the graph shown in the previous section. As seen in the previous section, the total difference in expected cost between a brand new component and one nearly breaking down, is exactly the cost of maintaining the component once. This is clearly as it should be.

Now, the behaviur of the model can be tested. when the cost of repairing a component is equal to the cost of maintaining it, there should be no difference and the threshold should go to ξ. This should return an optimal policy where the component is never maintained.

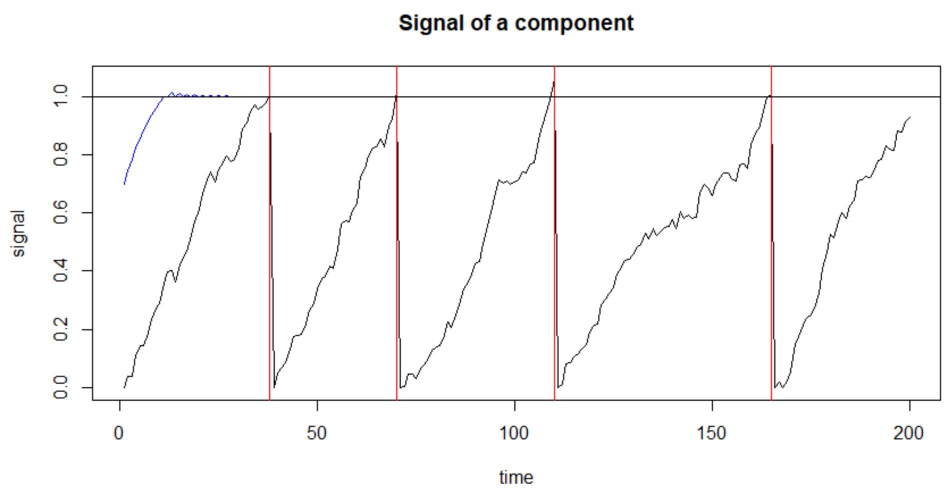


Figure 6: Signal (black) and theta (blue) when the cost of repairing is 1 (seed = 1)

As expected, the optimal policy is to never perform preventive maintenance. This is because, when both costs are equal, the optimal policy is to hold on to the component as long as possible. This can clearly be seen as θ quickly converges to ξ

9 Bayesian updating of μ

In the previous sections, the mean μ and variance σ^2 were assumed to be known. In practice, this is not the case. The component is tested in the lab, but the circumstances there are different from the circumstance of actual use. In this section, the parameters will be estimated using previously gained data. Bayesian inference will be used to estimate and update the hyper-parameters.

In Bayesian statistics, the parameters of the distribution are random. This is also the case in this problem. The mean and/or variance of the increment of the signal is assumed to be normally distributed. However, the parameters might not be known. In order to find these parameters, Bayesian statistics can play a role. This uses prior distributions to find posterior distributions of the parameter. Together with the prior distribution and some empirical data, one measurement of the signal in this case, a posterior distribution can be found. This means that $\mu \sim N(\mu_n, \tau_n)$ and that μ_n and τ_n are updated every time step.

The mean μ will first be estimated, while σ is still assumed to be known. μ will be assumed to be normally distributed. Their sum is then also normally distributed, which is very convenient during the computations.

The posterior conjugate of the normal distribution is also a normal distribution. This means that updating the parameters using obtained data and performing Bayesian inference, will not change the nature of the parameters. Only the hyper-parameters will change every update. The updated distribution is then called the posterior, while the one before is called the posterior.

9.1 Finding the updated parameters of μ

From every prior distribution, a new posterior distribution can be calculated when some observations are made. The distribution of the parameter is given by:

$$f(\mu|\underline{x}) = \frac{f(\underline{x}|\mu)f(\mu)}{p(\underline{x})}$$

In order to use the equation above, $f(\underline{x}|\mu)$ must be found. When this distribution is a prior conjugate, then the posterior distribution will be the same distribution as the prior. Luckily, the posterior conjugate of the normal distribution is a again a normal distribution. This makes the calculations easier and assures for an exact answer.

This is calculated below:

$$\begin{aligned}
f(\mu|\underline{x}) &= f(\mu)\frac{f(\underline{x}|\mu)*}{p(\underline{x})} \\
&\propto f(\mu)f(\underline{x}|\mu) \\
&= \frac{1}{\sqrt{2\pi\sigma_0^2}}e^{\frac{(-(\mu-\mu_0)^2}{2\sigma_0^2}} * \frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-(x-\mu)^2}{2\sigma^2}} \\
&\propto exp(-\frac{(-(\mu-\mu_0)^2}{2\sigma_0^2}) * exp(\frac{(-(x-\mu)^2}{2\sigma_0^2}) \\
&= exp(-\frac{(-(\mu-\mu_0)^2}{2\sigma_0^2} + \frac{(-(x-\mu)^2}{2\sigma^2}) \\
&= exp(\frac{-\sigma^2\mu^2 + 2\sigma^2\mu\mu_0 - \sigma^2\mu_0^2 - x^2\sigma_0^2 + 2x\mu\sigma_0^2 - \mu^2\sigma_0^2}{2\sigma^2\sigma_0^2}) \\
&= exp(\frac{-\mu^2(\sigma^2+\sigma_0^2) + 2\mu(\mu_0\sigma^2 + x\sigma_0^2) - (\mu_0^2\sigma^2 + x^2\sigma_0^2)}{2\sigma^2\sigma_0^2}) \\
&= exp(\frac{-\mu^2 + 2\mu\frac{\mu_0\sigma^2+x\sigma_0^2}{(\sigma^2+\sigma_0^2)} - (\frac{\mu_0\sigma^2+x\sigma_0^2)}{(\sigma^2+\sigma_0^2)})^2}{\frac{2\sigma^2\sigma_0^2}{(\sigma^2+\sigma_0^2)}}) \\
&\propto exp(\frac{-(\mu - \frac{\mu_0\sigma^2+x\sigma_0^2}{\sigma^2+\sigma_0^2})^2}{2\frac{\sigma^2\sigma_0^2}{\sigma^2+\sigma_0^2}})
\end{aligned}$$

Which is again a normal distribution with different parameters. These updated parameters are given by:

$$\frac{\mu_0\sigma^2 + x\sigma_0^2}{\sigma^2+\sigma_0^2}, \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-2}$$

Note the $\propto$, proportional to. Constant factors do not matter and can thus be left out.

9.2 Estimating μ

Now that the hyper-parameters can be updated, this can be implemented in the model. When a signal is given and a first guess is made, the data can be used to update the guessed μ. This is done by the formula just derived. Every timestep, the hyper-parameters μ and τ (variation of mean) is updated. The code for the estimation is in the first part of appendix E, together with the rest of the code.

The components are from now on not assumed to be from a homogeneous distribution. This means the estimations for each component could be different. In the next plot, the 95% confidence interval and the true value of mu.

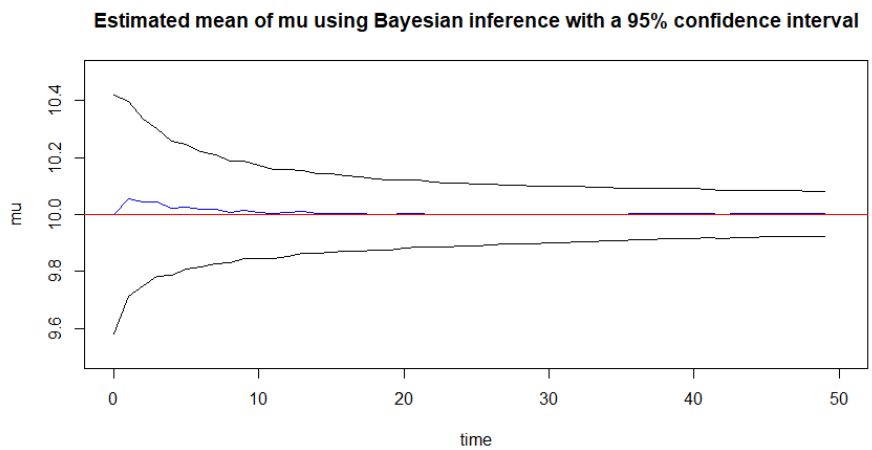


Figure 7: mean over 10.000 runs (blue) with 95% confidence interval (black) of the estimation of the actual value of μ (red)

Here it can be seen that the upper and lower bound converge towards the true value. The initial values in this case are $\mu_0 = \tau_0 = 10$.

This is also supported by the variance of μ, given by τ. This is updated every time step by the relation $\tau_{n+1} = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1}$. This quickly decreases as seen in this graph. This means the accuracy of μ increases.

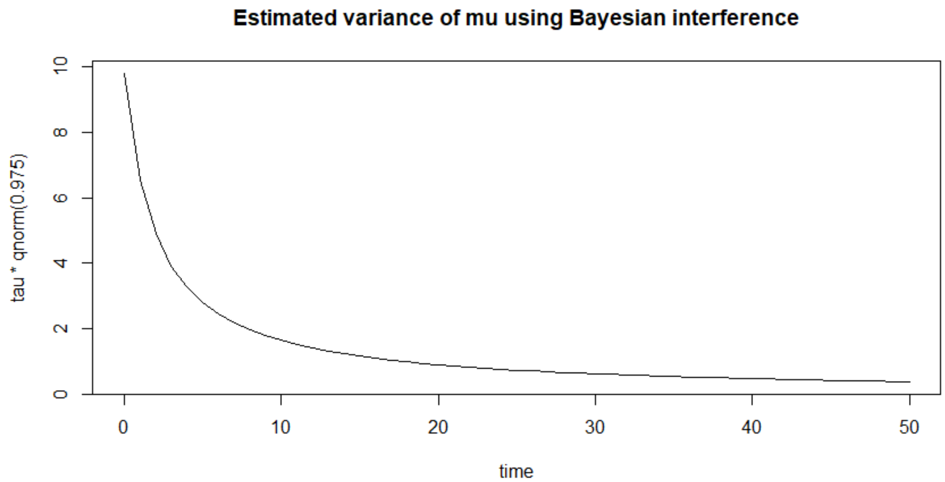


Figure 8: τ (= variance of μ) over 50 time steps

since the components are not assumed to come from a homogeneous distribution. Every component has a slightly different true mean. This means every time the component is repaired or maintained, the knowledge of that sensor is thrown away. This means the beliefs of the optimal time to perform maintenance will also change every time maintenance or reparation is performed.

9.3 Implementing the updated μ

Now that the parameters can be accurately estimated using Bayesian inference, they can be added to the model. Note that $\mu \sim N(\mu_n, \tau_n)$ and can thus be split into a deterministic part and a random part $\mu = \mu_n + \epsilon\tau_n$. They can be implemented in the model by using them in the value iteration function. The new function is now given by:

$$V^{(n+1)}(x, t, \mu_n, \tau_n) =$$
$$min \begin{cases} C_m + \mathbf{E}[e^{-T_m\beta} * V^{(n)}(0,0)] \text{ (maintain)} \\ e^{-r\Delta t} * \mathbf{E}[V^{(n)}(x + \Delta t\mu_n + (\sigma + \tau_n)\epsilon, t + \Delta t)] \text{(do nothing)} \end{cases}$$

where μ_n and τ_n are deterministic variables (but updated every time step) and ϵ is the normal distribution with mean 0 and variance 1.

Note that the μ is now a random variable which is updated every time. The only difference is present when the optimal action is to do nothing. The value of the updated value function is then given by:

$$V^{n+1}(x, \mu_n, \tau_n) = \begin{bmatrix} V(x + \mu_n\Delta + 5(\sigma + \tau_n)) \\ V(x + \mu_n\Delta + 5(\sigma + \tau_n)) \\ \vdots \\ V(x + \mu_n\Delta) \\ \vdots \\ V(x + \mu_n\Delta - 5(\sigma + \tau_n) + 1) \\ V(x + \mu_n\Delta - 5(\sigma + \tau_n)) \end{bmatrix} * \begin{bmatrix} \Phi(5(\sigma + \tau_n)) \\ \Phi(5(\sigma + \tau_n - 1)) \\ \vdots \\ \Phi(0) \\ \vdots \\ \Phi(-5(\sigma + \tau_n) + 1) \\ \Phi(-5(\sigma + \tau_n)) \end{bmatrix}$$

This represents the possible outcomes after Δt. The top row represents the value when the next signal is $x + \mu_n\Delta + 5(\sigma + \tau)$, which is then multiplied by the probability of getting there $(\Phi(5(\sigma + \tau)))$. When calculating this for all possible next signals and multiplying them by their probabilities, the expected value is represented.

When the best option is to preventively maintain the component, nothing changes since the cost is static over time and does not depend on μ_n or τ_n.

The changes to θ are not chosen to have a factor of $exp(-t)$, where t denotes the time spent estimating the parameters. The total code can be found in the appendix E

The graph of the expected cost of a component can now be calculated and plotted. With parameters arbitrarily chosen, the plot will look like this:

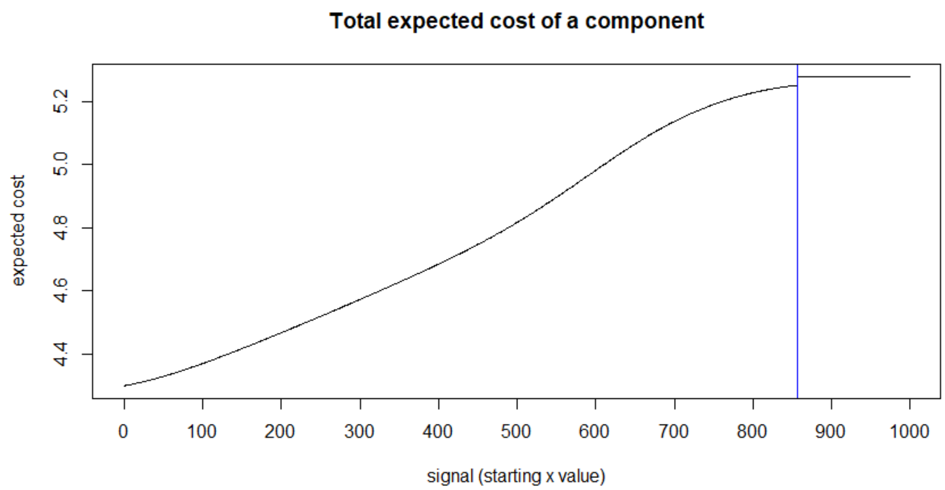


Figure 9: Total discounted expected cost when starting with a component with signal x (seed =2)

Here, the blue vertical line represents the optimal policy of maintenance. If the component is checked and the signal is lower than X, the best action is to do nothing. When the signal is between the blue line and $x = \xi$, in this case 1, then the best action is to preventively maintain the component.

Besides this, the signal and belief of optimal policy θ can be plotted over time.

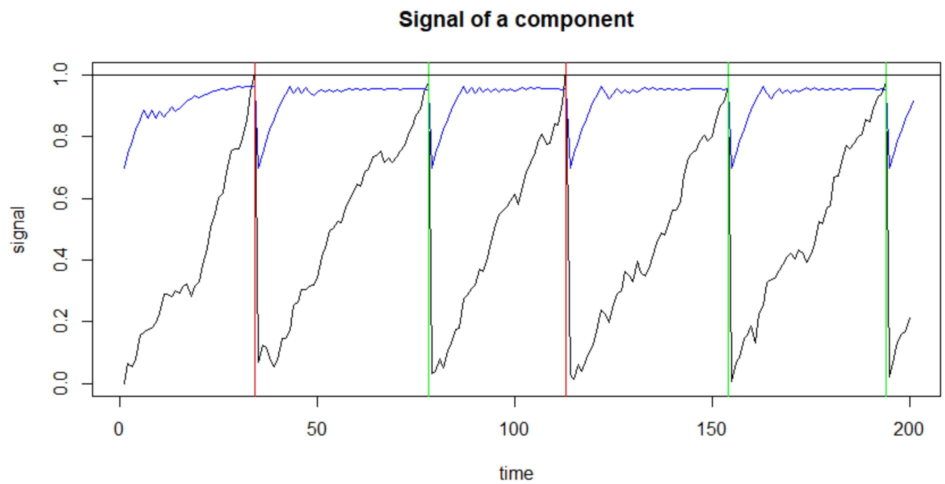


Figure 10: Signal (black) and θ (blue) over time with estimated μ. Component is maintained at $t = 78$, $t = 154$, $t = 194$ (green), and repaired at $t = 34$ and $t = 113$ (red) (seed = 5)

Note that since the components are not assumed to be homogeneous, the belief of θ resets every time, together with the estimations of μ.

10 Bayesian updating of σ^2

Now that the μ can be estimated and used in the program, σ is assumed to be random, while μ is assumed to be fixed. This requires a similar approach as used in the part where μ was assumed to be random.

10.1 Finding the updated parameters of σ^2

In order to be able to update the beliefs of σ^2, the distribution of it has to be determined. This is done by calculating the posterior which can be updated using observed data. since the distribution prior conjugate of σ, when μ is known is given by the inverse gamma distribution $f(x, \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}(1/x)^{\alpha+1}\exp(-\beta/x)$, the posterior can be calculated:

$$
\begin{aligned}
f(\sigma|\underline{x}) &\propto f(\sigma^2) * \frac{f(\underline{x}|\sigma^2)}{f(\underline{x})} \\
&\propto f(\sigma^2) * f(\underline{x}|\sigma^2) \\
&\propto \frac{\beta^\alpha}{\Gamma(\alpha)}(\sigma^2)^{-\alpha-1} exp\left(-\frac{\beta}{\sigma^2}\right)\sqrt{\frac{1}{2\pi\sigma^2}} exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \\
&\propto (\sigma^2)^{-\alpha-1} exp\left(-\frac{\beta}{\sigma^2}\right)\frac{1}{\sigma} exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \\
&\propto (\sigma^2)^{-\alpha-1/2} exp\left(\frac{-(x-\mu)^2}{2\sigma^2} - \frac{\beta}{\sigma^2}\right) \\
&\propto (\sigma^2)^{-\alpha-1/2} exp\left(\frac{-1/2(x-\mu)^2 - \beta}{\sigma^2}\right)
\end{aligned}
$$

And this is again an inversed gamma distribution, but with different parameters, namely:

$$\left(\alpha + \frac{1}{2}, \beta + \frac{(x-\mu)^2}{2}\right)$$

These are the updated parameters of the distribution of σ^2, which follows an inversed gamma distribution. $\sigma^2 \sim \Gamma'(\alpha, \beta)$, and the updated distribution is given by $\sigma^2 \sim \Gamma'(\alpha + \frac{1}{2}, \beta + \frac{(x-\mu)^2}{2})$

10.2 Estimating σ^2

After the updated distribution of σ^2 is found and proven, the random σ^2 can be implemented in the model. This is a bit different from the random μ since the sum of a normally distributed random variable and one distributed by the inverse gamma distribution is no known distribution and thus has to be calculated separately.

The overall idea is still the same as in the previous section. An initial guess for the α and β has to be made, and according to the observed data, the initial guess will be updated and eventually converge to the true value of σ^2.

This can easily be modelled into a program. Note that the mean of the inverse gamma distribution is given by $\frac{\beta}{\alpha-1}$, and that value can be used. The mean can be easily calculated by taking the average of all signals up to that point. Below, the 95% confidence interval of σ^2 is shown:

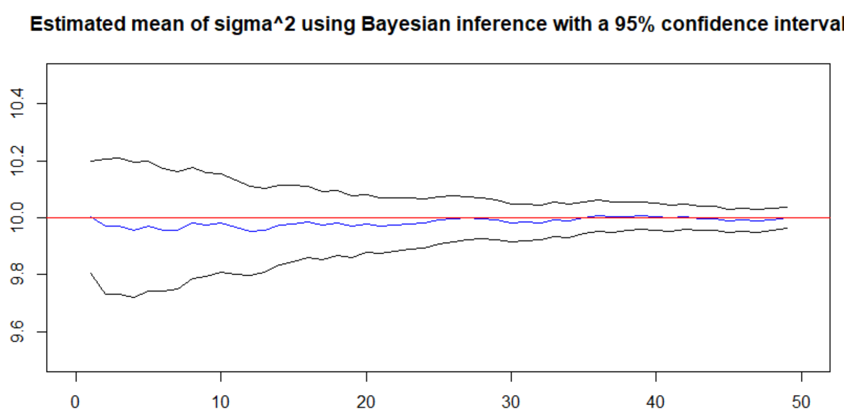


Figure 11: Mean (blue), and 95% confidence interval (black) of the estimation of the actual value of sigma (red).

10.3 Implementing the updated σ^2

In order to implement the updated estimation of σ, the previous model can be adapted. Note that only the random part in $V(x + \mu t + \epsilon\sigma^2)$ is different. This means the actions of repairing and maintaining do not change. The value of the next iteration when doing nothing is the best choice can also be calculated. This is done by multiplying the possible x values in the previous iteration by the probability of them occurring.

The function of the value iteration is now given by: $V^{(n+1)}(x, t, \sigma_n) =$
$$min \begin{cases} C_m + \mathbf{E}[e^{-T_m\beta} * V^{(n)}(0,0)] \text{ (maintain)} \\ e^{-r\Delta t} * \mathbf{E}[V^{(n)}(x + \Delta t\mu + \sigma_n\epsilon, t + \Delta t)] \text{(do nothing)} \end{cases}$$

Note that the product of two random variables has to be calculated. Since $\epsilon \sim N(0,1)$ and $\sigma \sim \Gamma(\alpha_n, \beta_n)$, their product is also random. Recall that $E[\sigma^2] = \frac{\beta}{\alpha-1}$

The value of the updated value function is then given by:

$$V^{n+1}(x,\alpha,\beta) = \begin{bmatrix} V(x+\mu+3*3\frac{\beta}{\alpha-1} \\ \vdots \\ V(x+\mu) \\ \vdots \\ V(x+\mu-3*3\frac{\beta}{\alpha-1} \end{bmatrix} * \begin{bmatrix} \Phi(3)*\Gamma^{-1}(3\frac{\beta}{\alpha-1}) \\ \vdots \\ \Phi(0) \\ \vdots \\ -\Phi(3)*\Gamma^{-1}(3\frac{\beta}{\alpha-1}) \end{bmatrix}$$

Note that the σ is a random variable and $\sigma_n \sim \Gamma(\alpha, \beta)$ and is updated every time step.

Here it is also assumed that, since the signal is discrete, there is almost no possibility of the signal increasing more than $3*3\frac{\beta}{\alpha-1}$ of less than $-3*3\frac{\beta}{\alpha-1}$ and it is therefore neglected.

The first vector is already calculated and with a difference of exactly 1, the step size is matched to the values of the previous iteration. The second vector is a bit more difficult. The values denote the possibilities of the component having a signal of exactly that, after one iteration. This vector can be made by going through all possible outcomes and discretizing them. This results in a vector with all probabilities of ending there after one time step. The code for this is again in appendix F.

First a vector with zeros is made. Then the different probabilities for the normal distribution and the inverse Gamma distribution are added to that vector, for all possible outcomes. That same vector is then appended to itself, without the probability of not moving (first entry of the vector), but in reversed order. This is because of symmetry in the normal distribution. Finally, the vector is scaled so that the total probabilities sum up to 1. This is needed because of the discretization, which causes rounding errors.

When the possible outcomes are known, this can be implemented in the model. This will result in a graph of the total discounted cost, when given a component with a signal of x.

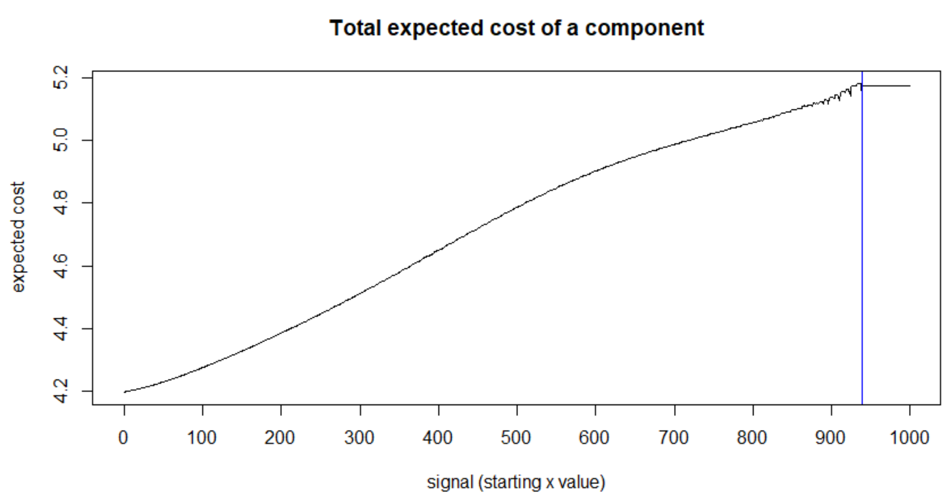


Figure 12: Total discounted cost when σ is estimated using Bayesian inference. The blue line indicates the threshold when to maintain the component.

An example of the signal over time can be seen below:

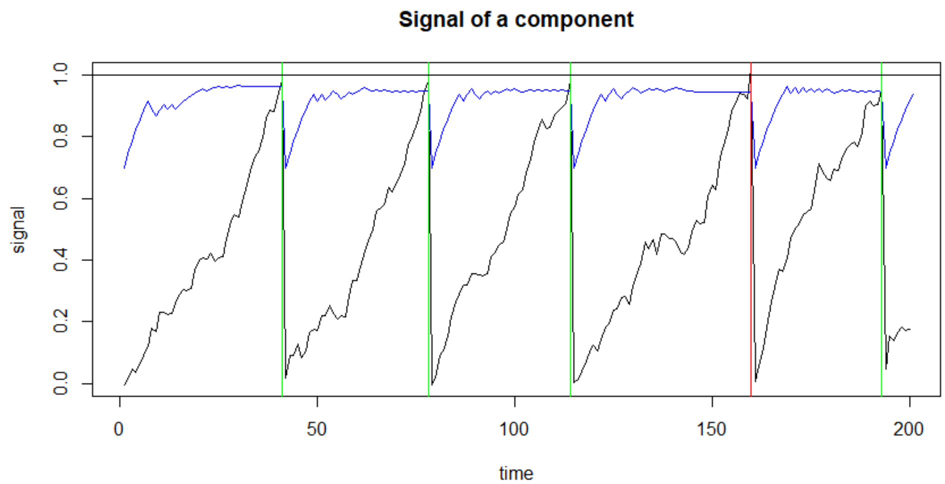


Figure 13: Signal (black) and θ (blue) over time with estimated σ. Repaired at $t = 33$ (red), maintained at $t = 59$, $t = 105$, $t = 154$, $t = 198$ (green), (seed = 15).

11 Bayesian updating of both μ and σ^2

In the previous sections, the mean and variance of the signal were estimated using Bayesian inference. Both times, only one parameter was assumed to be random, while the other one was assumed to be known. In this section, both parameters are assumed to be unknown. Estimating these parameters will be a bit harder, but Bayesian inference can still be used. First off, the updated parameters will be calculated. After that, they can be implemented in the model.

11.1 Finding the updated parameters of μ and σ^2

In order to update the beliefs of the distribution of μ and σ^2, the posterior has to be found.

$$f(\mu, \sigma^2 | \underline{x}) \propto f(\underline{x} | \mu, \sigma^2) \frac{f(\mu, \sigma^2)}{p(\underline{x})}$$

The normal-inverse-gamma distribution (or Gaussian-inverse-gamma distribution) is the conjugate prior of a normal distribution with unknown mean and variance. It is a four-parameter distribution. although σ is not directly given by the function, it can be calculated by $\sigma = \frac{\beta}{\alpha - 1}$. λ denotes the amount of observations made.

This can be used to find the updated parameters used in Bayesian inference.

The density function of the distribution is given by:

$$f(\mu, \sigma \mid x, \lambda, \alpha, \beta) = \frac{\sqrt{\lambda}}{\sigma\sqrt{2\pi}} \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left(-\frac{2\beta + \lambda(x - \mu)^2}{2\sigma^2}\right)$$

The updated parameters can now be calculated. Note that μ and σ are not directly given, but rather $\mu, \lambda, \alpha, \beta$.

$$
\begin{aligned}
f(\mu,\sigma|\underline{x}) = f(\mu,\lambda,\alpha,\beta|\underline{x}) &\propto f(\mu,\lambda,\alpha,\beta) * \frac{f(\underline{x}|\mu,\lambda,\alpha,\beta)}{f(\underline{x})} \\
&\propto f(\mu,\lambda,\alpha,\beta) * f(\underline{x}|\mu,\lambda,\alpha,\beta) \\
&= \frac{\sqrt{\lambda}}{\sigma\sqrt{2\pi}} \frac{\beta^{\alpha}}{\Gamma(\alpha)} \left(\frac{1}{\sigma^2}\right)^{\alpha+1} \exp\left(-\frac{2\beta+\lambda(x-\mu)^2}{2\sigma^2}\right) * \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \\
&\propto \left(\frac{1}{\sigma^2}\right)^{\alpha_0-\frac{1}{2}} \exp\left(-\beta_0\tau\right) \exp\left(-\frac{\lambda_0(\frac{1}{\sigma^2})(\mu-\mu_0)^2}{2}\right) \frac{1}{\sigma} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \\
&\propto \left(\frac{1}{\sigma^2}\right)^{\alpha_0} \exp\left(-\frac{1}{\sigma^2}\beta_0\right) \exp\left(-\frac{1}{2\sigma^2}\left(\lambda_0(\mu-\mu_0)^2 + (x-\mu)^2\right)\right)
\end{aligned}
$$

The part within the exponent can be simplified in the following way:

$$
\begin{aligned}
\lambda_0(\mu-\mu_0)^2 + (x-\mu)^2 &= \lambda_0\mu^2 - 2\lambda_0\mu\mu_0 + \lambda_0\mu_0^2 + \mu^2 - 2\bar{x}\mu + x^2 \\
&= (\lambda_0+1)\mu^2 - 2(\lambda_0\mu_0 + x)\mu + \lambda_0\mu_0^2 + x^2 \\
&= (\lambda_0+1)(\mu^2 - 2\frac{\lambda_0\mu_0+x}{\lambda_0+1}\mu) + \lambda_0\mu_0^2 + x^2 \\
&= (\lambda_0+1)\left(\mu - \frac{\lambda_0\mu_0+x}{\lambda_0+1}\right)^2 + \lambda_0\mu_0^2 + x^2 - \frac{(\lambda_0\mu_0+x)^2}{\lambda_0+1} \\
&= (\lambda_0+1)\left(\mu - \frac{\lambda_0\mu_0+x}{\lambda_0+1}\right)^2 + \frac{\lambda_0(x-\mu_0)^2}{\lambda_0+1}
\end{aligned}
$$

This can now be used in the previous formula:

$$
\begin{aligned}
f(\mu,\lambda,\alpha,\beta|\underline{x}) &\propto \left(\frac{1}{\sigma^2}\right)^{\alpha_0} \exp\left(-\frac{1}{\sigma^2}\beta_0\right) \exp\left(-\frac{1}{2\sigma^2}\left(\lambda_0(\mu-\mu_0)^2 + (x-\mu)^2\right)\right) \\
&\propto \left(\frac{1}{\sigma^2}\right)^{\alpha_0} \exp\left(-\frac{1}{\sigma^2}\beta_0\right) \exp\left(-\frac{1}{2\sigma^2}((\lambda_0+1)\left(\mu - \frac{\lambda_0\mu_0+x}{\lambda_0+1}\right)^2 + \frac{\lambda_0(x-\mu_0)^2}{\lambda_0+1}\right) \\
&\propto \left(\frac{1}{\sigma^2}\right)^{\alpha_0} \exp\left(-\frac{1}{\sigma^2}\beta_0 + \frac{\lambda_0(x-\mu_0)^2}{\lambda_0+1}\right) \exp\left(-\frac{1}{2\sigma^2}((\lambda_0+1)\left(\mu - \frac{\lambda_0\mu_0+x}{\lambda_0+1}\right)^2\right)
\end{aligned}
$$

And this is exactly the normal inversed gamma distribution with updated parameters:

$$
\left(\frac{\lambda_0\mu_0+x}{\lambda_0+1}, \lambda_0+1, \alpha_0+\frac{1}{2}, \beta_0+\frac{\lambda_0(x-\mu_0)^2}{\lambda_0+1}\right)
$$

Note that $\mu_{n+1}, \alpha_{n+1}, \beta_{n+1}$ are almost the same as the cases where only one variable was unknown (only λ is new).

11.2 Implementing the updated μ and σ^2

Now that both parameters can be estimated, they too can be implemented in the model. The equations used in the model can now be altered such that the μ and σ are both random.

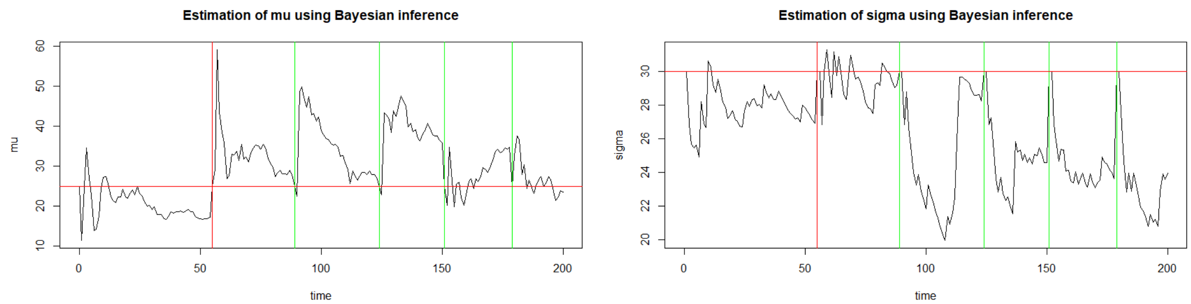
$$
V^{(n+1)}(x,t,\mu_n,\lambda_n,\alpha_n,\beta_n) = \\
min \begin{cases} C_m + \mathbf{E}[e^{-T_m r} * V^{(n)}(0,0)] \text{ (maintain)} \\ e^{-r\Delta t} * \mathbf{E}[V^{(n)}(x+\Delta t\mu + \sigma\epsilon, t+\Delta t)] \text{(do nothing)} \end{cases}
$$

Where $\mu \sim N(\mu_n, \frac{\beta}{(\alpha-1)\lambda})$ and $\sigma^2 \sim \Gamma^{-1}(\alpha, \beta)$.

The expected value of $V^n(x + \Delta t\mu + \sigma\epsilon, t + \Delta t)$ can be calculated in the same way as done before, the in-product of two vectors. One with all possible outcomes after Δt time, and one with the corresponding probabilities.

The values in between the first vector all differ by one because of discretization. The probabilities are represented in the second vector respectively.

The code can be found in appendix G. Below, the estimations of μ and σ can be seen below. An example of the signal and θ over time can be seen thereafter.



(a) Estimation of μ using Bayesian inference. True value is $\mu = 25$

(b) Estimation of σ using Bayesian inference. True value is $\sigma = 30$

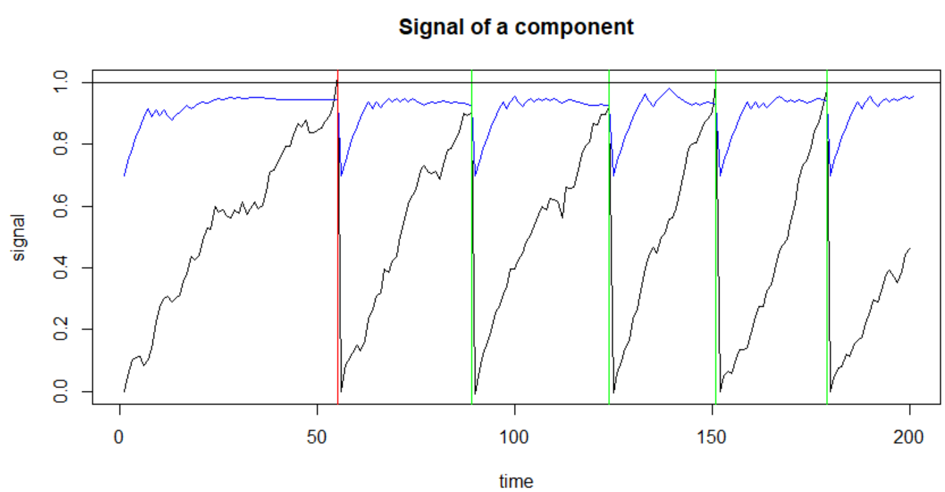


Figure 14: Signal (black) and θ (blue) over time with estimated μ and σ (seed = 58)

Here, it can be seen that the first time, maintenance was too late and the component had to be repaired. The next four times, maintenance was performed on time.

Also note that the optimal threshold θ can increase and decrease. Earlier in the report, the monotonicity of θ was stated. This does not mean earlier statement was wrong, since there everything else was assumed to stay constant. Since the signal changes every step, this is not the case and a decrease in θ can happen.

Finally, the three different approaches can be compared. Below, there is a signal, together with the beliefs of optimal policies using the three different ways.

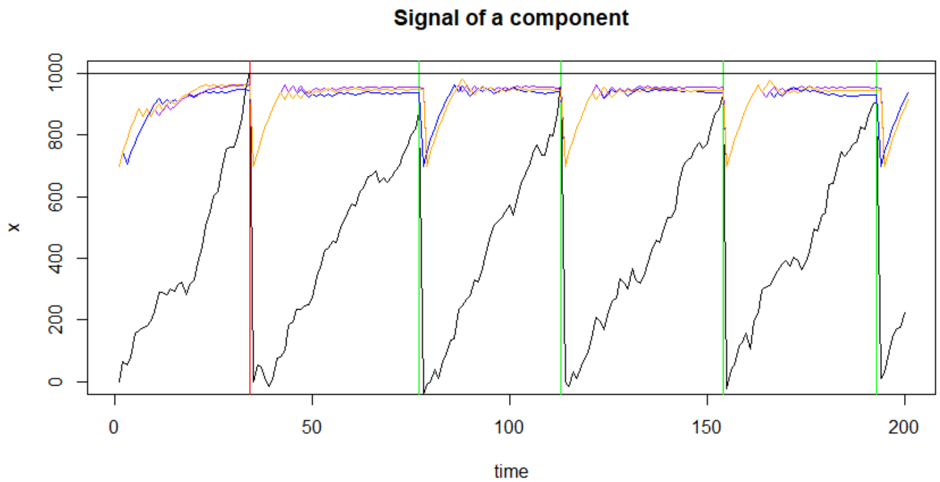


Figure 15: Signal (black) and θ for all three approaches over time with estimating μ (purple), σ (blue) and both μ and σ (orange) (seed = 97,$C_r = 2$)

Here it is clear that all approaches are similar, but they still differ a bit from each other. Depending on the available information, the method can be chosen.

12 Discussion

Throughout this report, several things are assumed to be simpler than they in fact are. This model does not take every factor into account. Several things that could be improved in the model are as follows:

If the sensor is just below θ, and at the next time step just below ξ, the model suggests the component to be maintained. There is however, a possibility that the component breaks down in between two checkups and that its signal is lower than ξ at both times. This is something to look into as it is totally neglected in this report.

Another aspect of the model that could be improved is that the change in θ depends on the knowledge of the value iteration function. In the first few time steps however, the value iteration function is not very accurate since it greatly depends on the initial guess of θ. Later in in time, this gets more accurate.

Finally, this research can be compared to the paper that was the main motivation for this research (Elwany, 2011 [1]). In their paper they have also chosen for a discrete time problem. The sample path of the signal is also assumed to be a Brownian motion. The difference in methodology lies in the increments in the signal. Their signal is split into a homogeneous part and a components-specific part. In this paper, all components are assumed to be from a non-homogeneous set. another difference is that they also include observation costs, which are neglected in this paper.

During their research they also used the value iteration function (equation 2) in order to calculate the total expected discounted costs. Another difference is that their signal is transformed to be an exponential function, whereas it is a linear function in this paper.

Their research also provides empirical data as to how their findings behave in the real world. This is a great way of verifying the results, but due to time limitations could not be done in this paper.

Their model is not included, so no comparison could be made. Fortunately, their pseudo-code was included and looked similar to the one used in this paper (section 6.3).

13 Conclusion

When a component is observed for a period of time, a maintenance-policy can be made in order to minimize the cost of it. This can be done by evaluating the value iteration function. At every step in time, the underlying parameters of the increase in the signal (normally distributed) can be estimated using Bayesian inference. This increases the accuracy of the estimated parameters.

From here, the optimal policy, together with the signal can be plotted.

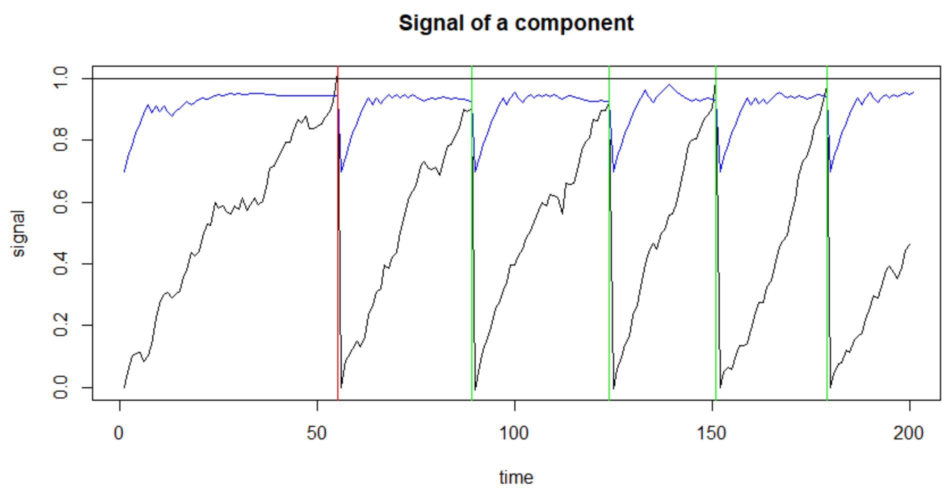


Figure 16: Signal (black) and θ (blue) over time with estimated μ and σ (seed = 58)

Here, the belief of θ, the optimal time to perform maintenance can be seen over time, together with the signal x. This will help reduce the cost of repairing components, as well as the ability to give a clear estimation of the total discounted cost of a component in the future.

References

[1] Alaa H. Elwany, Nagi Z. Gebraeel, and Lisa M. Maillart. Structured Replacement Policies for Components with Complex Degradation Processes and Dedicated Sensors. *Operations Research*, 59(3):684–695, 2011.

[2] Nagi Z. Gebraeel, Mark A. Lawley, Rong Li, and Jennifer K. Ryan. Residual-life distributions from component degradation signals: A Bayesian approach. *IIE Transactions (Institute of Industrial Engineers)*, 37(6):543–557, 2005.

[3] Wenbin Wang, Matthew Carr, Wenjia Xu, and Khairy Kobbacy. A model for residual life prediction based on brownian motion with an adaptive drift. *Microelectronics Reliability*, 51(2):285 – 293, 2011. 2010 Reliability of Compound Semiconductors (ROCS) Workshop Prognostics and Health Management.

[4] Hanwen Zhang, Donghua Zhou, Maoyin Chen, and Xiaopeng Xi. Predicting remaining useful life based on a generalized degradation with fractional brownian motion. *Mechanical Systems and Signal Processing*, 115:736 – 752, 2019.

[5] Zeyi Huang, Zhengguo Xu, Xiaojie Ke, Wenhai Wang, and Youxian Sun. Remaining useful life prediction for an adaptive skew-wiener process model. *Mechanical Systems and Signal Processing*, 87:294 – 306, 2017.

A Finding the cost for a specific policy

```
for(i in signal){
        cost <- 0
        for(n in N){
            x <- rnorm(n = length(time) -1 ,
                        sd = sqrt(variance),mean = mu)
            x <- c(0, cumsum(x))
            for(t in time){
                if(x[t]>threshold) {
                    x[(t+1):endtime] <- x[(t+1):endtime] - x[t+1]
                    cost <- cost + cr * exp(-(beta * t))
                }
                if(x[t] > threshold*i & x[t]<threshold){
                    x[(t+1):endtime] <- x[(t+1):endtime] - x[t+1]
                    cost <- cost + cm * exp(-(beta * t))
                }
            }
        }
  costVec <- c(costVec,mean(cost))
}
```

B Code for the basic model

```
set.seed(2)

###################
Do not increase sigma too much (>100), since the signal going below 0 causes the model to
    exit
###################


# Time variables
beta <- 0.01
Tm <- 0.01
Tr <- 0.05
deltaT <- 0.001

# Discount for time
discountRepair <- exp(-beta *Tr)
discountDoNothing <- exp(-beta *deltaT)
discountMaintain <- exp(-beta * Tm)

# Amount of iterations
maxIteration <- 200

endT <- maxIteration
time <- seq(1,endT,1)

mu <- 25
sigma <- 30

# Generate x and the signal vectors
x <- rnorm(rep(0,(endT)),mu,sigma)
signal <- x
x <- cumsum(x)
x[1] <- 0

# costs
repairCost <- 1.2
maintainCost <- 1

```

```
# Signal starts at 100, broken at 1100 steps of 0.1 from 1 to 100
newLevel <- 1000
brokenLevel <- 2000
endX <- 2200

signalAxis <- seq(1,endX,1)

# Initial value for V0
initialValue <- 0

# OldV is first guess
oldV <- rep(initialValue,endX)
newV <- rep(initialValue,endX)

# Add first guess to total matrix of costs
totalV <- oldV

# Initial guess for x* and increase it by starting position of X
initialX <- 700
X <- initialX+newLevel

timeSinceReset <- 0

for(iteration in 1:maxIteration){
  # Calculate new cost (n+1)
  timeSinceReset <- timeSinceReset + 1
  # Resize normalprob to the appropriatesize
  NormalProb <- dnorm(seq(-5*sigma,5*sigma, length.out = (10*sigma + 1) ) ,0,sigma)
  NormalProb <- NormalProb/sum(NormalProb)

  # If do nothing is the best choice
  for(i in newLevel:X[iteration]){
    newV[i] <- sum( oldV[ (i+mu-(5*sigma)) :
                              (i+mu +(5*sigma))] * NormalProb)
  }
  # If signal is negative, assign value of V(newlevel)
  for(i in 1:(newLevel-1)){
    newV[i] <- newV[newLevel]
  }
  # If maintaining is the best choice
  for(i in (X[iteration]+1):brokenLevel){
    newV[i] <- maintainCost + discountMaintain* oldV[newLevel]
  }
  # If broken already
  for(i in (brokenLevel+1):endX){
    newV[i] <- repairCost + discountRepair * oldV[newLevel]
  }

  # Add new cost to total cost matrix
  totalV <- cbind(totalV,newV)

  # Make new cost vector the old
  oldV <- newV


  # Update X
  if(X[iteration]<=brokenLevel && X[iteration]>newLevel && !is.null(oldV[X[iteration]])){
    # If cost of doing nothing is lower than maintaining, lower threshold
    if(newV[X[iteration]] >= maintainCost + discountMaintain* oldV[newLevel]){
      X <- c(X,X[iteration] - exp(-timeSinceReset/10) * (50))
    } # If cost is higher/equal to maintaining, increase threshold
    else{
      X <- c(X,X[iteration] + exp(-timeSinceReset/10) * (50))
    }
  }else{
    X <- c(X, brokenLevel-1)
  }
}
```

```

# Plot with right starting X Values
plot(signalAxis[newLevel:(brokenLevel-1)],newV[newLevel:(brokenLevel-1)],type = 'l',
      xlab = "signal (starting x value)", ylab = "expected cost",axes = FALSE,
      main = 'Total expected cost of a component')
abline(v=X[maxIteration],col = "blue")
axis(side=1, at=seq(newLevel,brokenLevel,100),labels = seq(0,newLevel,100))
axis(2)
box()

##############################################

repairTiming <- -100
maintainTiming <- -100

for(t in time){
  if(!is.null(x[t])){
    # repair
    if(x[t]>brokenLevel-newLevel) {
      x[(t+1):endT] <- x[(t+1):endT] - x[t+1]
      repairTiming <- c(repairTiming,t)
    }
    if(!is.na(x[t])){
      # maintain
      if(x[t] >=  X[t]-newLevel && x[t] < brokenLevel-newLevel){
        x[(t+1):endT] <- x[(t+1):endT] - x[t+1]
        maintainTiming <- c(maintainTiming,t)
      }
    }
  }
}

plot(time,x,type = 'l',main = 'Signal of a component')
lines(c(time,(endT+1)),X-newLevel,type = 'l',col = "blue")
abline(v = repairTiming,col = 'red')
abline(v = maintainTiming, col = 'green')
```

C Code for estimating μ

```
mu <- guessedMu
tau <- guessedTau

for(t in time){
  mu <- c(mu,  (tau[t]*signal[t] /(sigma^2+tau[t]^2) + sigma*mu[t] /(sigma^2+tau[t]^2)))
  tau <- c(tau,1/(tau[t]^2+1/sigma^2))
}
```

D Code for estimating σ

```
beta <- guessedBeta
alpha <- guessedAlpha
sigma <- beta/(alpha-1)

for(t in time){
  alpha <- c(alpha, alpha[t] + 1/2)
  beta <- c(beta, beta[t] + (signal[t]-actualMu)^2/2)
  sigma <- c(beta[t]/(alpha[t]-1))
```

```
}
```

E Value iteration with μ unknown

```
set.seed(2)

# Time variables
beta <- 0.001
Tm <- 0.01
Tr <- 0.05
deltaT <- 0.001

repairTiming <- -10
maintainTiming <- -10

# Discount for time
discountRepair <- exp(-beta *Tr)
discountDoNothing <- exp(-beta *deltaT)
discountMaintain <- exp(-beta * Tm)

# Amount of iterations
maxIteration <- 200

# Initial variables
actualMu <- 25
actualSigmasq <- 900
sigmasq <- actualSigmasq

endT <- maxIteration
time <- seq(1,endT,1)

# Generate x and the signal vectors
x <- rnorm(rep(0,(endT)),actualMu,sqrt(actualSigmasq))
signal <- x
x <- cumsum(x)
x[1] <- 0

# costs
repairCost <- 1.2
maintainCost <- 1

newLevel <- 1000
brokenLevel <- 2000
endX <- 2200

signalAxis <- seq(1,endX,1)

# Initial value for V0
initialValue <- 0

# OldV is first guess
oldV <- rep(initialValue,endX)
newV <- rep(initialValue,endX)

# Add first guess to total matrix of costs
totalV <- oldV

# Initial guess for x* and increase it by starting position of X
initialX <- 700
X <- initialX+newLevel

# Initial guess for mu and tau
muInitialGuess <- 20
```

```
tauInitialGuess <- 10
sigma <- sqrt(sigmasq)

timeSinceReset <- 0

mu <- muInitialGuess
tau <- tauInitialGuess

###############################################

for(t in time){
  mu <- c(mu, (tau[t]*signal[t] /(actualSigmasq+tau[t]) + actualSigmasq*mu[t] /(
      actualSigmasq+tau[t])))
  tau <- c(tau,1/(1/tau[(t)]+1/actualSigmasq))
}

for(iteration in 1:maxIteration){
  # Calculate new cost (n+1)


  # Resize normalprob to the appropriatesize
  NormalProb <- dnorm(seq(-5*sigma,5*sigma, length.out = (10*(sigma+ceiling(tau[iteration]))
      + 1) ) ,0,(sigma+tau[iteration]))
  NormalProb <- NormalProb/sum(NormalProb)

  # If do nothing is the best choice
  for(i in newLevel:X[iteration]){
    newV[i] <- sum( (oldV[ (i+mu[iteration]-(5*(sigma+ceiling(tau[iteration])))) :
                              (i+mu[iteration]+(5*(sigma+ceiling(tau[iteration]))))] *
                                  NormalProb) )
  }
  # If signal is negative, assign value of V(newlevel)
  for(i in 1:(newLevel-1)){
    newV[i] <- newV[newLevel]
  }
  # If maintaining is the best choice
  for(i in (X[iteration]+1):brokenLevel){
    newV[i] <- maintainCost + discountMaintain* oldV[newLevel]
  }
  # If broken already
  for(i in (brokenLevel+1):endX){
    newV[i] <- repairCost + discountRepair * oldV[newLevel]
  }

  # Add new cost to total cost matrix
  totalV <- cbind(totalV,newV)

  # Make new cost vector the old
  oldV <- newV

  timeSinceReset <- timeSinceReset + 1

  # Update X
  if(X[iteration]<=brokenLevel && X[iteration]>newLevel && !is.null(oldV[X[iteration]])){
    # If cost of doing nothing is lower than maintaining, lower threshold
    if(newV[X[iteration]] >= maintainCost + discountMaintain* oldV[newLevel]){
      X <- c(X,X[iteration] - exp(-timeSinceReset/10) * (50))
    } # If cost is higher/equal to maintaining, increase threshold
    else{
      X <- c(X,X[iteration] + exp(-timeSinceReset/10) * (50))
    }
  }else{
    X <- c(X,brokenLevel-1)
  }

  # Check if signal is reset, if so, forget mu
  if(x[iteration]>= X[iteration]-newLevel){

```

```
      timeSinceReset <- 0

      # reset signal
      x[(iteration+1):endT] <- x[(iteration+1):endT] - x[iteration]

      # Forget knowledge of mu after reset
      mu <- mu[1:(iteration)]
      tau <- tau[1:(iteration)]

      mu<- c(mu, muInitialGuess)
      tau <- c(tau, tauInitialGuess)

      # Estimate mu and tau again
      for(t in (iteration+1):(endT)){
        mu <- c(mu, (tau[t]*signal[t] /(actualSigmasq+tau[t]) + actualSigmasq*mu[t] /(actualSigmasq+tau[t])))
        tau <- c(tau,1/(1/tau[(t)]+1/actualSigmasq))
      }
      # reset belief of X
      X <- c(X[-length(X)], initialX+newLevel)
    }

    # repair
    if(x[iteration]>=brokenLevel-newLevel) {
      repairTiming <- c(repairTiming, iteration)
    }

    # maintain
    if(x[iteration] >= X[iteration]-newLevel && x[iteration] < brokenLevel-newLevel){
      maintainTiming <- c(maintainTiming, iteration)
    }
}

# Plot estimated parameters
plot(c(0,time),mu,type = 'l', main = 'Estimation of parameter using Bayesian inference')
abline(v = repairTiming, col = 'red')
abline(v = maintainTiming, col = 'green')
abline(h = actualMu, col = 'red')



# Plot with right starting X Values
plot(signalAxis[newLevel:(brokenLevel-1)],newV[newLevel:(brokenLevel-1)],type = 'l',
     xlab = "signal (starting x value)", ylab = "expected cost",axes = FALSE, main = 'Total expected cost of a component')
abline(v=X[maxIteration], col = "blue")
axis(side=1, at=seq(newLevel,brokenLevel,100),labels = seq(0,newLevel,100))
axis(2)
box()
###############################################

plot(time,x,type = 'l',main = 'Signal of a component' ,ylim = c(0,1000))
lines(c(time,(endT+1)),X-newLevel,type = 'l',col = "blue")
abline(v = maintainTiming, col = 'green')
abline(v = repairTiming, col = 'red')

abline(h = 1000)

muX <- X-newLevel
```

F Value iteration with σ unknown

```
library("invgamma")
set.seed(2)

# Time variables
interest <- 0.001
Tm <- 0.01
Tr <- 0.05
deltaT <- 0.001

repairTiming <- -10
maintainTiming <- -10

# Discount for time
discountRepair <- exp(-interest *Tr)
discountDoNothing <- exp(-interest *deltaT)
discountMaintain <- exp(-interest * Tm)

# Amount of iterations
maxIteration <- 200

# Initial variables
actualMu <- 20
mu <- actualMu
actualSigmasq <- 900


endT <- maxIteration
time <- seq(1,endT,1)

# Generate x and the signal vectors
x <- rnorm(rep(0,(endT)),actualMu,sqrt(actualSigmasq))
signal <- x
x <- cumsum(x)

# costs
repairCost <- 1.2
maintainCost <- 1

newLevel <- 1000
brokenLevel <- 2000
endX <- 2200

signalAxis <- seq(1,endX,1)

# Initial value for V0
initialValue <- 0

# OldV is first guess
oldV <- rep(initialValue,endX)
newV <- rep(initialValue,endX)

# Add first guess to total matrix of costs
totalV <- oldV

# Initial guess for x* and increase it by starting position of X
initialX <- 700
X <- initialX+newLevel

# Initial guess for mu and tau
alphaInitialGuess <- 11
betaInitialGuess <- 9000
sigmasqInitialGuess <- betaInitialGuess/(alphaInitialGuess-1)

timeSinceReset <- 0


alpha <- alphaInitialGuess
beta <- betaInitialGuess
```

```
sigmasq <- beta/(alpha-1)


##############################################

for(t in time){
  alpha <- c(alpha, alpha[t] + 1/2)
  beta <- c(beta, beta[t] + (signal[t] - mu)^2/2)
  sigmasq <- c(sigmasq, beta[t]/(alpha[t]-1))
}

for(iteration in 1:maxIteration){
  # Calculate new cost (n+1)

  alpha[iteration] <- ceiling(alpha[iteration])
  beta[iteration] <- ceiling(beta[iteration])

  # Calculate probabilities of ending up at certain values with 3 random variables

  probability <- rep(0, (3*ceiling(sqrt(sigmasq[iteration])))+1)

  N <- seq(0,2,0.25)
  G <- seq(sqrt(sigmasq[iteration]),2*sqrt(sigmasq[iteration]), length.out = 20)

  for(n in N){
      for(g in G){
        if(ceiling(n*g) <length(probability)){
          probability[ceiling(n*g)] <-  probability[ceiling(n*g)] +
            dinvgamma(g,alpha[iteration],beta[iteration]) * dnorm(n,0,1)
        }
      }
  }

  probability <- c(rev(probability[-1]),probability)
  probability <- probability/sum(probability)


  # If do nothing is the best choice
  for(i in newLevel:X[iteration]){
    newV[i] <- sum( oldV[ (i+mu-3*ceiling(sqrt(sigmasq[iteration]))):
                          (i+mu+3*ceiling(sqrt(sigmasq[iteration])))] * probability)
  }
  # If signal is negative, assign value of V(newlevel)
  for(i in 1:(newLevel-1)){
    newV[i] <- newV[newLevel]
  }
  # If maintaining is the best choice
  for(i in (X[iteration]+1):brokenLevel){
    newV[i] <- maintainCost + discountMaintain* oldV[newLevel]
  }
  # If broken already
  for(i in (brokenLevel+1):endX){
    newV[i] <- repairCost + discountRepair * oldV[newLevel]
  }

  # Add new cost to total cost matrix
  totalV <- cbind(totalV,newV)

  # Make new cost vector the old
  oldV <- newV

  timeSinceReset <- timeSinceReset + 1

  # Update X
  if(X[iteration]<=brokenLevel && X[iteration]>newLevel && !is.null(oldV[X[iteration]])){
    # If cost of doing nothing is lower than maintaining, lower threshold
    if(newV[X[iteration]] >= maintainCost + discountMaintain* oldV[newLevel]){
      X <- c(X,X[iteration] - exp(-timeSinceReset/10) * (50))
```

```
    } # If cost is higher/equal to maintaining, increase threshold
    else{
      X <- c(X,X[iteration] + exp(-timeSinceReset/10) * (50))
    }
  }else{
    X <- c(X,brokenLevel-1)
  }

  # Check if signal is reset, if so, forget sigma
  if(x[iteration]>=  X[iteration]-newLevel){

    # reset signal
    x[(iteration+1):endT] <- x[(iteration+1):endT] - x[iteration]
    timeSinceReset <- 0

    # Forget knowledge of mu and sigma after reset
    sigmasq <- sigmasq[1:(iteration)]
    alpha <- alpha[1:iteration]
    beta <- beta[1:iteration]

    sigmasq <- c(sigmasq,sigmasqInitialGuess)
    alpha <- c(alpha,alphaInitialGuess)
    beta <- c(beta,betaInitialGuess)

    # Estimate mu and sigma again
    for(t in (iteration+1):(endT)){
      alpha <- c(alpha,alpha[t] + 1/2)
      beta <- c(beta,beta[t] + (signal[t] - mu)^2/2)
      sigmasq <- c(sigmasq,beta[t]/(alpha[t]-1))
    }

    # reset belief of X
    X <- c(X[-length(X)],initialX+newLevel)
  }

  # repair
  if(x[iteration]>=brokenLevel-newLevel) {
    repairTiming <- c(repairTiming,iteration)
  }

  # maintain
  if(x[iteration] >=  X[iteration]-newLevel && x[iteration] < brokenLevel-newLevel){
    maintainTiming <- c(maintainTiming,iteration)
  }
}

# Plot with right starting X Values
plot(signalAxis[newLevel:(brokenLevel-1)],newV[newLevel:(brokenLevel-1)],type = 'l',
     xlab = "signal (starting x value)", ylab = "expected cost",axes = FALSE,main = 'Total
        expected cost of a component')
abline(v=X[maxIteration],col = "blue")
axis(side=1, at=seq(newLevel,brokenLevel,100),labels = seq(0,newLevel,100))
axis(2)
box()
##############################################

# Plot estimated parameters
plot(c(0,time),sigmasq,type = 'l', main = 'Estimation of parameter using Bayesian inference'
    )
abline(v = repairTiming,col = 'red')
abline(v = maintainTiming, col = 'green')
abline(h = actualSigmasq, col = 'red')

# Plot signal
plot(time,x,type = 'l',main = 'Signal of a component', ylim = c(0,1000))
lines(c(time,(endT+1)),X-newLevel,type = 'l',col = "blue")
abline(v = repairTiming,col = 'red')
abline(v = maintainTiming, col = 'green')
```

```
abline(h = 1000)
```

G Value iteration with both μ and σ unknown

```
library("invgamma")

set.seed(3)

# Time variables
interest <- 0.001
Tm <- 0.01
Tr <- 0.05
deltaT <- 0.001

repairTiming <- -10
maintainTiming <- -10

# Discount for time
discountRepair <- exp(-interest *Tr)
discountDoNothing <- exp(-interest *deltaT)
discountMaintain <- exp(-interest * Tm)

# Amount of iterations
maxIteration <- 200

# Initial variables
actualMu <- 20
actualSigmasq <- 900

endT <- maxIteration
time <- seq(1,endT,1)

# Generate x and the signal vectors
x <- rnorm(rep(0,(endT)),actualMu,sqrt(actualSigmasq))
signal <- x
x <- cumsum(x)
x[1] <- 0

# costs
repairCost <- 1.2
maintainCost <- 1

newLevel <- 1000
brokenLevel <- 2000
endX <- 2200

signalAxis <- seq(1,endX,1)

# Initial value for V0
initialValue <- 0

# OldV is first guess
oldV <- rep(initialValue,endX)
newV <- rep(initialValue,endX)

# Add first guess to total matrix of costs
totalV <- oldV

# Initial guess for x* and increase it by starting position of X
initialX <- 700
X <- initialX+newLevel

# Initial guess for mu and tau
```

```
muInitialGuess <- 25
lambdaInitialGuess <- 0
alphaInitialGuess <- 3
betaInitialGuess <- 100
sigmaInitialGuess <- betaInitialGuess/alphaInitialGuess

timeSinceReset <- 0

mu <- muInitialGuess
lambda <- 0
alpha <- alphaInitialGuess
beta <- betaInitialGuess
sigma <- beta/(alpha-1)

############################################

for(t in time){
  mu <- c(mu,(lambda[t]*mu[t] + signal[t] )/ (lambda[t] + 1))
  lambda <- c(lambda,lambda[t] + 1)
  alpha <- c(alpha,alpha[t] + 1/2)
  beta <- c(beta,beta[t] + 1/2 * (lambda[t] *(signal[t] - mu[t])^2)/(lambda[t] + 1))
  sigma <- c(sigma,beta[t]/(alpha[t]-1))
}

for(iteration in 1:maxIteration){
  # Calculate new cost (n+1)

  alpha[iteration] <- ceiling(alpha[iteration])
  beta[iteration] <- ceiling(beta[iteration])

  # Calculate probabilities of ending up at certain values with 3 random variables

  probability <- rep(0, (3*ceiling(sqrt(sigma[iteration])))+1)

  N <- seq(0,2,0.25)
  M <- seq(0,2,0.25)
  G <- seq(beta[iteration]/(alpha[iteration]-1)/2,beta[iteration]/(alpha[iteration]-1)*2 ,
          length.out = 10)

  for(n in N){
    for(m in M){
      for(g in G){
        if(ceiling(n*m*g) <length(probability)){
          probability[ceiling(n*m*g)] <-  probability[ceiling(n*m*g)] +
            dinvgamma(g,alpha[iteration],beta[iteration]) * dnorm(n,0,1) * dnorm(m,0,1)
        }
      }
    }
  }

  probability <- c(rev(probability[-1]),probability)
  probability <- probability/sum(probability)


  # If do nothing is the best choice
  for(i in newLevel:X[iteration]){
    newV[i] <- sum( oldV[ (i+mu[iteration]-3*ceiling(sqrt(sigma[iteration]))):
                                (i+mu[iteration]+3*ceiling(sqrt(sigma[iteration])))] *
                                    probability)
  }
  # If signal is negative, assign value of V(newlevel)
  for(i in 1:(newLevel-1)){
    newV[i] <- newV[newLevel]
  }
  # If maintaining is the best choice
  for(i in (X[iteration]+1):brokenLevel){
    newV[i] <- maintainCost + discountMaintain* oldV[newLevel]
  }
```

```r
  # If broken already
  for(i in (brokenLevel+1):endX){
    newV[i] <- repairCost + discountRepair * oldV[newLevel]
  }

  # Add new cost to total cost matrix
  totalV <- cbind(totalV,newV)

  # Make new cost vector the old
  oldV <- newV

  timeSinceReset <- timeSinceReset + 1

  # Update X
  if(X[iteration]<=brokenLevel && X[iteration]>newLevel && !is.null(oldV[X[iteration]])){
    # If cost of doing nothing is lower than maintaining, lower threshold
    if(newV[X[iteration]] >= maintainCost + discountMaintain* oldV[newLevel]){
      X <- c(X,X[iteration] - exp(-timeSinceReset/10) * (50))
    } # If cost is higher/equal to maintaining, increase threshold
    else{
      X <- c(X,X[iteration] + exp(-timeSinceReset/10) * (50))
    }
  }else{
    X <- c(X, brokenLevel-1)
  }

  # Check if signal is reset, if so, forget mu
  if(x[iteration]>=  X[iteration]-newLevel){

    # reset signal
    x[(iteration+1):endT] <- x[(iteration+1):endT] - x[iteration]
    timeSinceReset <- 0

    # Forget knowledge of mu and sigma after reset
    mu <- mu[1:(iteration)]
    sigma <- sigma[1:(iteration)]
    lambda <- lambda[1:iteration]
    alpha <- alpha[1:iteration]
    beta <- beta[1:iteration]

    mu <- c(mu, muInitialGuess)
    sigma <- c(sigma, sigmaInitialGuess)
    lambda <- c(lambda,0)
    alpha <- c(alpha, alphaInitialGuess)
    beta <- c(beta, betaInitialGuess)

    # Estimate mu and sigma again
    for(t in (iteration+1):(endT)){
        mu <- c(mu,(lambda[t]*mu[t] + signal[t] )/ (lambda[t] + 1))
        lambda <- c(lambda,lambda[t] + 1)
        alpha <- c(alpha, alpha[t] + 1/2)
        beta <- c(beta, beta[t] + 1/2 * (lambda[t] *(signal[t] - mu[t])^2)/(lambda[t] + 1))
        sigma <- c(sigma, beta[t]/(alpha[t]-1))
    }

    # reset belief of X
    X <- c(X[-length(X)], initialX+newLevel)
  }

  # repair
  if(x[iteration]>=brokenLevel-newLevel) {
    repairTiming <- c(repairTiming, iteration)
  }

  # maintain
  if(x[iteration] >=  X[iteration]-newLevel && x[iteration] < brokenLevel-newLevel){
    maintainTiming <- c(maintainTiming, iteration)
  }
```

```r
}

# Plot with right starting X Values
plot(signalAxis[newLevel:(brokenLevel-1)],newV[newLevel:(brokenLevel-1)],type = 'l',
      xlab = "signal (starting x value)", ylab = "expected cost",axes = FALSE,main = 'Total
        expected cost of a component')
abline(v=X[maxIteration],col = "blue")
axis(side=1, at=seq(newLevel,brokenLevel,100),labels = seq(0,newLevel,100))
axis(2)
box()
###############################################

for(t in time){
  if(!is.null(x[t])){
    # repain
    if(x[t]>=brokenLevel-newLevel) {
      x[(t+1):endT] <- x[(t+1):endT] - x[t+1]
      repairTiming <- c(repairTiming,t)
    }
    if(!is.na(x[t])){
      # maintain
      if(x[t] >=  X[t]-newLevel && x[t] < brokenLevel-newLevel){
        x[(t+1):endT] <- x[(t+1):endT] - x[t+1]
        maintainTiming <- c(maintainTiming,t)
      }
    }
  }
}

# Plot estimated parameters
plot(c(0,time),sigma,type = 'l', main = 'Estimation of parameter using Bayesian inference')
abline(v = repairTiming,col = 'red')
abline(v = maintainTiming, col = 'green')
abline(h = actualSigmasq, col = 'red')

# Plot signal
plot(time,x,type = 'l',main = 'Signal of a component', ylim = c(0,1000))
lines(c(time,(endT+1)),X-newLevel,type = 'l',col = "blue")
abline(v = repairTiming,col = 'red')
abline(v = maintainTiming, col = 'green')
abline(h = 1000)
```