Quality prediction and online parameter estimation for an engine assembly line

Quality prediction and online parameter estimation for an engine assembly line.

DCT 2006.19

C.J.A. van Kats

Supervisors:
Prof. Dr. H. Nijmeijer
Prof. Dr. M. Guay

Eindhoven University of Technology
Department of Mechanical Engineering
Dynamics and Control Group

Queen's University, Kingston, Ontario, Canada
Department of Chemical Engineering
Control Group

December, 2005

Abstract

A dynamic model of Zone 2 of General Motors' engine assembly line in St. Catharines is studied. The model is based on the Stochastic Flow Model of each station in the line. Based on historical data, it makes a prediction of the First Time Quality for a future production plan. The basis of this prediction is defined by estimation of the mean time between successive rejections in the past. An improved algorithm is developed to determine this parameter. Next, based on the method of Maximum Likelihood Estimation, a recursive estimation algorithm is proposed. Based on the result of a newly developed process simulation algorithm, improved prediction are achieved.

Contents

Chapter 1

Introduction

At zone 2 of General Motors' assembly line in St. Catherines, V8 engines are assembled. The line has a throughput of approximately 2500 engines per day. Plant personnel works 24 hours a day, 7 days per week in 3 work shifts a day. At the line 23 different engine types are being assembled. The part of the production line that is studied, consists of 78 workstations in series. Each engine that enters the line, passes through all workstations. After every workstation a quality check is applied on the performed operations. This can either lead to acceptance or rejection of the engine. In case of rejection, the engine is taken out of the production line to be either repaired or disassembled. A disassembled engine reenters the line at a later time. The plant management is interested in reducing the number of rejections that occur in the process. One would like to be able to anticipate on potential future rejections. The objective of this project is to develop a tool that can predict when and where rejects will occur in the planned future.

Based on a Stochastic Flow Model (SFM) for every station in the line, a program is being developed that makes a prediction of the quality of the line. At the start of this study, an unfinished implementation the given model is provided. The first goal of this project is to further develop the program to make it able to perform quality predictions. After that, possible improvement of the predictive quality and online implementation are researched.

A description of the Stochastic Flow Model and the way it is used in calculation of the First Time Quality (FTQ) will be described in chapter 2. The implementation of the model and the working of the resulting program will be discussed and predictions for the quality of a future production plan will be made in chapter 3. Because we are interested in an online prediction of the future process quality, an online parameter estimation program is being developed. As a basis of this program, a possible recursive parameter estimation algorithm is discussed in chapter 4. To study the potential of online quality prediction, a program will be developed to simulate online behavior of the production process. In chapter

5 this program is discussed. Based on the results, new predictions for a day of future production will be made. Conclusions and recommendations are given in chapter 6.

Chapter 2

System Model Formulation

A dynamic model of the assembly was given in [1]. The model is based on a Stochastic Flow Model (SFM) of each station in the line. For each station, information about the probability of the occurrence of rejections is provided. Based on this input, the model computes an estimation of the number of rejections for any given future production plan. As a measure of the quality, the First Time Quality (FTQ) is used.

2.1 Stochastic Flow Models

Manufacturing systems generally consist of a combination of sources supplying parts to servers with buffers where parts can be stored while awaiting to be processed. A natural modeling framework for such systems is provided through queuing systems or, more generally, discrete event systems (DES) [4]. An approach to incorporate real-time behaviour of the system is given by timed-DES models [1], in which time couples the occurrence of each event. The applicability of DES is limited by the complexibility of manufacturing systems. Most times it fails to capture the underlying continuous dynamics of a manufacturing system. Addition of continuous-time dynamics to DES models, using hybrid systems modeling, provides a greater modeling flexibility. A higher level of modeling abstraction is provided through stochastic flow models (SFM). In a SFM, stochastic fluctuations in the supply and service processes are captured by treating flow rates as random processes. For study of the assembly line at St. Catharines, a dynamic model was based on a SFM of each station in the line.

2.2 Stochastic Flow Model of a workstation

The dynamic model of the manufacturing system is based on a SFM of each workstation ([4]). A graphical representation of the SFM of a workstation is shown in figure 2.1.

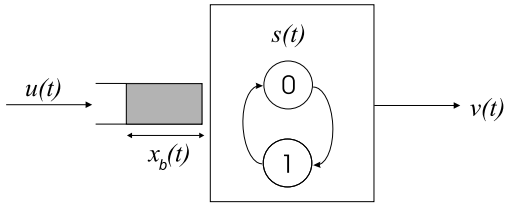

Figure 2.1: Schematic repesentation of the SFM of a workstation

The rate at which parts arrive at the workstation is given by rate $u(t)$ [parts/s]. The processing rate of the station is given by $v(t)$ [parts/s]. When the arrival rate is larger than the processing rate, parts are stored in a buffer. The buffer content at time t is denoted by $x_b(t)$ [parts]. Its dynamics can be described by the simple ordinary differential equation

$$\frac{dx_b(t)}{dt} = u(t) - v(t). \tag{2.1}$$

The station switches between two states, 0 (not working) or 1 (working). The operating state $s(t)$ of the station is thus defined as follows.

$$s(t) = \begin{cases} 1 & \text{machine working} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

The changing of the current state is caused by random stochastic events that are related to processing conditions. The value of $u(t)$ is a function of planning and scheduling. The processing rate $v(t)$ is also a function of the state $s(t)$ and the buffer content $x_b(t)$. An operational constraint of the processing rate can be formulated as

$$v(t) = \begin{cases} 0 & \text{if } s(t) = 0 \text{ or } x_b(t) = 0 \\ \phi(t) & \text{otherwise} \end{cases} \tag{2.3}$$

So the station has a processing rate $\phi(t)$ if it is able to work and the buffer contains parts to be processed.

The manufacturing system considered in this study consists of a number of stations in series. To model this, the SFM's describing successive stations can be connected such that the processing rate, $v(t)$, of a station becomes the arrival rate, $u(t)$, of the next station. This is shown graphically in figure 2.2. In this

study the processing times of all stations are assumed to be equal and constant. As a result, the arrival rates $u(t)$ and processing rates $v(t)$ are equal for all stations. With (2.1), this results elimination of the dynamics of buffer level x_b.

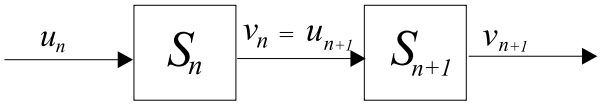

Figure 2.2: Connection of Stochastic flow models of successive stations

2.3 Quality monitoring

In this study, the purpose of application of the SFM of is to make a prediction of the quality of the production process. The quality in the plant is directly related to the number of faults that occur. In this case, a fault is equivalent to the reject of a processed parts at a certain workstation. The number of rejections x_r in the line are accounted by solving the following ordinary differential equation

$$\dot{x_r}(t) = \Delta(t)\delta(t). \tag{2.4}$$

with initial condition $x_r(0) = 0$. $\delta(t)$ is a unit pulse and $\Delta(t)$ is given by

$$\Delta(t) = \begin{cases} 0 & \text{if } v(t) = 0 \\ 1 & \text{if } v(t) \neq 0 \text{ and } t_f \leq t_p, \end{cases} \tag{2.5}$$

Here the variable t_f is a random variable that represents the time between successive rejections. The occurrence of rejections is assumed to be a Poisson process. Therefore, t_f is assumed to follow an exponential distribution that is parameterized by the mean occurrence rate λ and has a mean

$$E(t_f) = \theta = \frac{1}{\lambda}. \tag{2.6}$$

The value of λ is assumed to be fixed for a given station and is a function of plant operating conditions. These include the shift identity and the engine type. The variable t_p represents the mean processing time at a workstation. t_p is assumed to be fixed for each station. So in (2.4) a reject is counted if the time that a part type has been produced at a workstation, exceeds the expected failure time.

2.4 First Time Quality

An absolute number of counted rejects does not give a useful indication of the quality of the process. A commonly used measure for process quality is the First Time Quality (FTQ). It is the percentage of parts that go trough the workstation without rejects. The FTQ for a time period of t is thus given by

$$FTQ(t) = \frac{\int_0^t v(\tau)d\tau - x_r(t)}{\int_0^t v(\tau)d\tau} \times 100, \tag{2.7}$$

where $x_r(t)$ is the number of rejects and $v(t)$ the processing rate.

Chapter 3

Quality prediction program

Based on the implementation of the model described in chapter 2, a *Matlab* program was developed to analyse the First Time Quality (FTQ) of a portion of the assembly line. On basis of given historical reject data, it makes an estimation of the expected FTQ for each station, for any given production plan. This chapter gives a short overview of the main idea of the program. More extensive descriptions of the files and their main variables can be found in Appendix A.

3.1 Inputs

The main inputs of the model are the production plan for which the FTQ has to be estimated and the historical operational- and rejection data. All available data is provided in the form of Excel© spreadsheets. The operational data lists the engine code, the engine type and the time at which the engine is scheduled to enter the line. The rejection code data provides the engine code, the engine type and the time at which the engine was rejected, together with the station of rejection and a rejection identification code. Figure 3.1 shows parts of typical operational and reject data files.

After loading these sets into the *Matlab* workspace, the data that is relevant for the actual FTQ calculations, is substracted.

3.2 Mean time between rejects.

The main parameter in the FTQ prediction algorithm is the mean time between rejects $E[t_f]$. The program uses the historical rejection- and operational data from the plant to compute the mean time between rejections that occurred in the past. The mean times are sorted by station number, engine type and shift. Having this, the mean times are provided to the SFM to parameterize

operational

	engine code	type	year	month	day	hour	minute	second
	A	B	C	D	E	F	G	H
1	K0415222[illegible]	SHC	2004	6	1	0	0	15
2	K0415222[illegible]	SHC	2004	6	1	0	0	41
3	K0415222[illegible]	SHC	2004	6	1	0	2	1
4	K0415222[illegible]	SHC	2004	6	1	0	2	24

reject

	engine code	type	year	month	day	hour	minute	second	station	reject code
	A	B	C	D	E	F	G	H	I	J
1	K041522120	SHC	2004	6	1	0	58	39	258	25806
2	K041522121	SHC	2004	6	1	1	3	0	258	25812
3	K041522122	SHC	2004	6	1	1	3	12	211	21102
4	K041522158	SHC	2004	6	1	1	8	39	258	25812

Figure 3.1: Typical operational data and rejection data sheet

the exponential distributions of the fault time t_f, which is used as a decision making parameter in the accounting of rejects through (2.4). The line consists of 78 stations, there are 24 engine types produced and the plant personnel works in 3 work shifts. In the past however, rejects occurred at only 23 of the 78 stations. All $23 \times 24 \times 3$ possible combination of the plant conditions result in a different distribution parameter. In order to get a reasonable estimation of the mean times for all relevant combinations, the historical data set must be sufficiently large.

3.3 FTQ prediction

The resulting three dimensional array of distributional parameters is used as the main input of the SFM based FTQ prediction algorithm. For all events in the production plan, the following actions are taken by the program:

- Identify the execution time, engine type and shift.
- Select distributional parameters, corresponding with the particular engine type and shift.
- Take the random variable t_f from the corresponding exponential distribution.
- Simulate the SFM model for all stations and account rejects by solving (2.4). In this study, a constant value of 24 seconds is assumed for the proces times t_p of all stations.

After simulation, the FTQ response is given by (2.7), which in practice becomes

$$\frac{\#\text{ produced engines} - \#\text{ rejected engines}}{\#\text{ produced engines}} \times 100 \tag{3.1}$$

3.4 Simulation Results and analysis

Based on the historical data of a 6 month range of production, the stochastic decision making parameters t_f are computed. This result is used to predict the FTQ for a day's production plan. In this case, data from the period 1 June 2004 - 30 November 2004 is used to calculate the distribution of t_f. A prediction of the FTQ is made for the production plan of 3 Januari 2005. The simulation is started by running the file *run_old_ftq_calc.m*. Results are shown in figure 3.2.

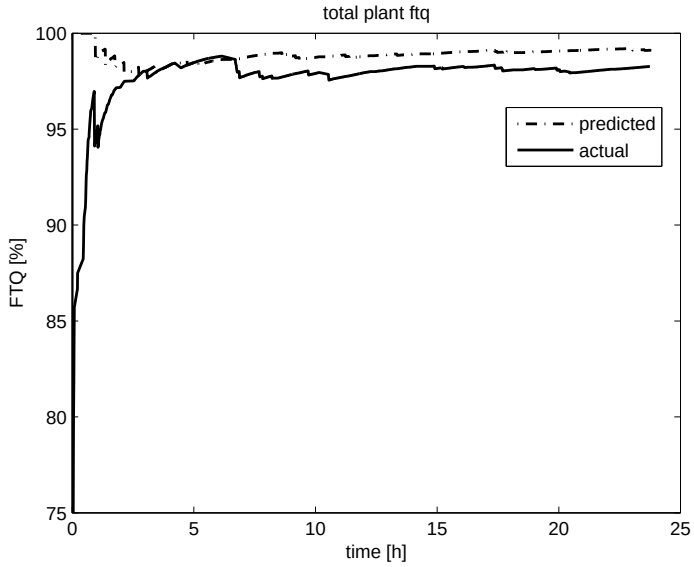

Figure 3.2: FTQ prediction for the production on 3 Januari 2005, based on historical data of half a year production in the year before.

The graph shows that the predicted FTQ is mostly higher that the actual FTQ. More rejections occurred than were predicted. Possibly this is caused by the assumption of the same processing time of 24 seconds for all stations. In fact the only difference between the models of the different stations, is the distribution paramter λ. The model also does not take into account that some engines do not go through all stations. A rejection or a start at the end of the measuring period however, causes an engine not to reach the end of the line. Neglecting these phenomena results in a higher values of t_f. Looking at (2.5), this partly explains the accounting of less rejections. In future models, this should be taken into account.

Chapter 4

Recursive parameter estimation

The key parameter that is required for predicting the FTQ is the stochastic variable t_f, representing the mean time between following rejections. To make FTQ predictions that are adaptive to the current situation of the production process, we are interested in an adaptive online estimation of the parameter t_f. A method to make a point estimate of a parameter is given by the theory of maximum likelihood estimation.

4.1 Maximum Likelihood Estimation

The value of t_f is assumed to follow an exponential distribution parameterized by the mean occurrence rate λ. The mean time between rejections is a function of the station, the engine type and the shift identity. At each time we are interested in an estimation of t_f, sorted for these three plant conditions.

We define θ as the mean time between rejections. An estimate of the parameter θ is given by $\hat{\theta}$. At any given time t, the set of all rejections and corresponding planning data provides a set of N independent observations of the time between rejects, $t_{f1},\ t_{f2}, ..., t_{fN}$. An estimate of the mean time between rejects $\hat{\theta}$ can be obtained by the value of θ that maximizes the likelihood function. The likelihood function is the joint probability density function after substitution of the available observations:

$$\begin{aligned} L(\lambda|t_{f1}, t_{f2}, ..., t_{fN}) &= \prod_{i=1}^{N} \lambda e^{-\lambda t_{fi}} \\ &= \lambda^N e^{-\lambda \sum_{i=1}^{N} t_{fi}}. \end{aligned} \tag{4.1}$$

Where $\lambda = \frac{1}{\theta}$. The logarithmic likelihood function is given by:

$$\Lambda(\lambda) = \ln(L(\lambda)) = N \ln(\lambda) - \lambda \sum_{i=1}^{N} t_{fi} \tag{4.2}$$

The maximum likelihood estimator $\hat{\lambda}$ can be obtained by maximizing L or Λ. Here Λ is used, because it is easier to work with. The derivative of (4.2) with respect to λ is given by

$$\begin{aligned} \frac{\partial \Lambda}{\partial \lambda} &= \frac{N}{\lambda} - \sum_{i=1}^{N} t_{fi} \\ &= \frac{N}{\lambda} - N\bar{t}_f. \end{aligned} \tag{4.3}$$

Upon equating this last result to zero, we obtain

$$\hat{\lambda} = \frac{N}{\sum_{i=1}^{N} t_{ri}} = \frac{1}{\bar{t}_f} \tag{4.4}$$

where $\bar{t}_f = \frac{1}{N} \sum_{i=1}^{N} t_{fi}$ is the sample mean.

An estimation routine for the occurrence rate of rejects λ is obtained by updating in the steepest descent direction of the log likelihood function (4.2):

$$\dot{\hat{\lambda}} = g \frac{\partial \Lambda}{\partial \lambda}, \tag{4.5}$$

where g is a positive constant gain. Using update law (4.5), an adaptive estimation routine for the distribution parameter λ is obtained. Substitution of this parameter into the stochastic flow model, adaptive predictions of the process quality can be made.

Chapter 5

Process simulation with online parameter estimation

To study the realtime potential of the recursive estimation method described in chapter 4, a *Matlab* program is developed. Using the rejection- and operational data of a certain time period, it performs a simulation of the production process and makes a pseudo-realtime estimation of the mean fault time $\bar{t}_f$. More information about the program files and corresponding script, can be found in appendix B.

5.1 Working principle

After loading the rejection- and operational data sheets into the workspace, the program *nshift_times.m* extracts the relevant data and brings it in the format required for the simulation. It also calculates the mean times between rejects directly from the given data. Next, the actual simulation is started by running *lambdarun.m*. The program simulates online behavior in the plant by assuming the simulation time units as real time second. At every time step, the operational starting times are observed and compared to the simulation time. Next, basically the following actions are taken at each time step:

- Find engines that have started, thus starting times that are exceeded by the simulation time.
- Estimate the station number that the started engines are being produced at. Here, the assumption is made that no intermediate buffer storage occurs. So all engines in the line move to the next station as soon as a new engine enters the line.

- For engines that are not rejected before the estimated station number, the process times at the corresponding positions (station,engine type,shift) are updated with the simulation time step.
- For engines that are rejected at the station number that they are assumed to be at:
  - update the production times at the corresponding positions with the simulation time step
  - count a rejection at the corresponding position
  - determine rejection time interval: production time between current reject and last rejection at the same station,engine type and shift.
- Update value of mean times between rejects $\bar{t}_f$ and the occurrence rate λ.

5.2 Update mechanism

An adaptive estimation of the occurrence rate of rejections is given by the estimation routine (4.5). With (4.3), this becomes

$$\dot{\hat{\lambda}} = g\left(\frac{N}{\lambda} - \sum_{i=1}^{N} t_{fi}\right) \tag{5.1}$$

$$= g\left(\frac{N}{\lambda} - N\bar{t}_{f.}\right) \tag{5.2}$$

In time, the operational conditions in the production line are liable to changes. Measures taken by the management like for example training of operator personnel or changes the machinery, may effect the probability of the occurrence of rejections and thus the value of λ. Because of this fact, it is desired to make an estimation, based on a recent set of time intervals t_{fi}, which is large enough to maintain accuracy. This can be achieved by only using a constant number of N last time measurements in (5.1). That way a moving measurement window can be created.

For each possible combination of plant conditions (station,engine type, shift), a different occurrence rate has to be estimated. Therefore, the system has a state vector of length $(78 \cdot 23 \cdot 3)$. Due to the limitations of computing power and memory in combination with the size of the variables, (5.1) has not been applied in this study. Instead, at every time step k, the value of $\bar{t}_f$ is updated with the sequence

$$\bar{t}_{f,k+1} = \frac{1}{N_k}(\bar{t}_{f,k}k \cdot N_{k-1} + t_{f,k}). \tag{5.3}$$

Next, (5.2) is used to perform the adaption update. A plot of (5.2) is shown in figure 5.1. It shows that a discontinuity occurs at $\lambda = 0$.

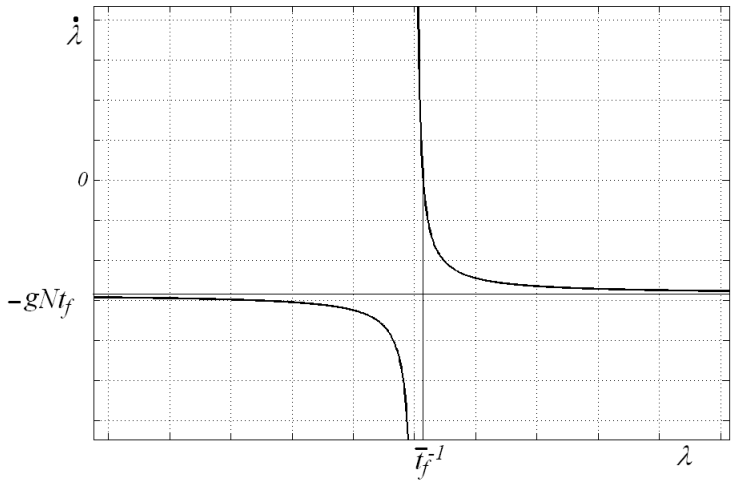

Figure 5.1: Graphical representation of update law (5.2)

If the initial condition is in the right half plane and the simulation time step is sufficiently small, the value of λ will always converge towards $\bar{t}_f^{-1}$. In this study however, we are forced by limitations of computing capacity, to use a First Order Euler fixed step method for solving the differential equations. The time range of the measurement period is of order months. To achieve a reasonable simulation speed, simulation time steps of order seconds are necessary. As initial condition, the right values of $\bar{t}_f$ from earlier simulation are taken. Under these circumstances, no gain g can be found that achieves convergence for all states. Because of the fixed time step, in some cases the value of λ crosses the discontinuity, which causes divergence of the estimator. As a solution to this problem, the following update law was proposed:

$$\dot{\hat{\lambda}} = \begin{cases} g(\frac{N}{\lambda} - N\bar{t}_f) & \text{if } \lambda > \epsilon \\ \\ g(\frac{N}{\epsilon} - N\bar{t}_f) & \text{otherwise} \end{cases} \tag{5.4}$$

Where $0 < \epsilon < \bar{t}_f$. A plot of (5.4) is shown in figure 5.2. By choosing the right value for ϵ, the value of $\hat{\lambda}$ will always converge towards $\bar{t}_f$. In this study, one update law was used to update all $(78 \cdot 23 \cdot 3)$ states. Because of the large difference between mean reject times for the various positions, it turns out to be hard to select one ϵ that achieves fast convergence of all values. For some states, the difference between the absolute values of the update law in the left-

and the right halfplan becomes very large. This large difference in combination with the large simulation step size, causes the parameters to jump far over the discontinuity and converge back very slowly.

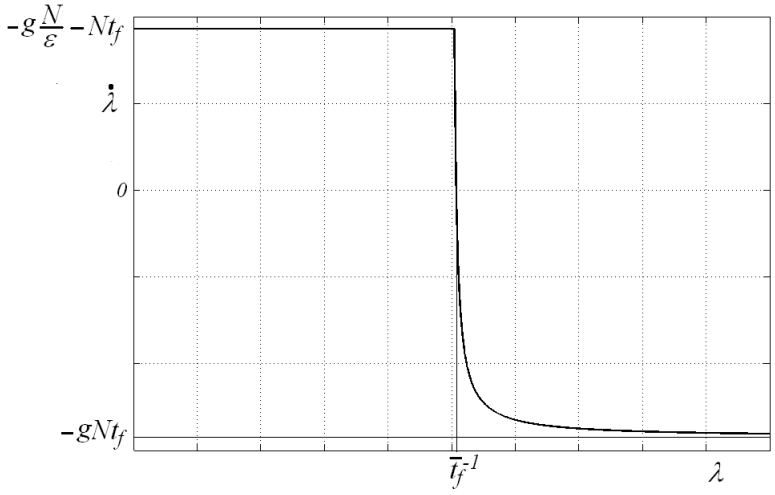

Figure 5.2: Graphical representation of update law (5.4)

5.2.1 Remarks

More study has to be done to achieve right convergence of the estimators. Within the time span of this short study, no adaptive update method has lead to satisfying estimation of all parameters. The limitation of computing power was experienced as an important restraint. The used software turns out to be not very suitable to deal with the large variables that were used. After optimizing the code and vectorizing all search loops, there is still a lot of cpu power necessary to perform the search actions and parameter updates at every time step. Being forced to use a First Order Euler method with large time steps for solving the differential equation, limits the possibilities in considerable extent. The machine at which simulations were run, has a fairly high capacity. Therefore, most improvement can be gained by writing the program in a different programming language, for example *Fortran*© or *C*©.

5.3 Simulation Results and Analysis

To show the potential of the program, it is used to simulate the production of the period 1 June - 30 November 2004. First *nshift_times* is run to sort the

data. It also produces estimates of the mean times between rejections, which are computed directly from the data. Next, the production simulation was started by running *lambdarun.m*. In the simulation, the mean time between rejections is computed at every time step. The main advantage is that real time data from the plant is used, instead of assuming a constant process time. Both estimations are used in the FTQ program to make a prediction of the plant quality on 3 Januari 2005. Results are shown in figures 5.3 and 5.4.

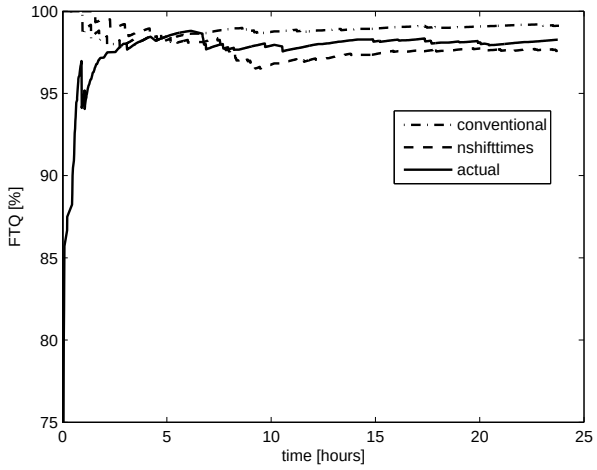

Figure 5.3: FTQ prediction for the production on 3 Januari 2005, based on mean times from *nshift_times*

From figure 5.3 it can be concluded that the distributional parameters calculated by the program *nshift_times* give slightly more accurate results than the conventional model. Apparently, the more accurate approximation of the mean time between rejections worked positively. However, especially in the beginning of the simulation, a large difference remains beween the predicted- and the actual FTQ. Obviously, from figure 5.4 can be concluded that the prediction based on the online parameter estimation program performs much better. Apparently the assumption of a process time of 24 seconds for all stations, can not be verified. Especially for the first stations in the line, this assumption has an obstructing influence on the quality of the prediction. Still, the FTQ SFM model makes this same assumption. Probably, removing this assumption there will lead to more improvement. In further study, this should be considered as an option.

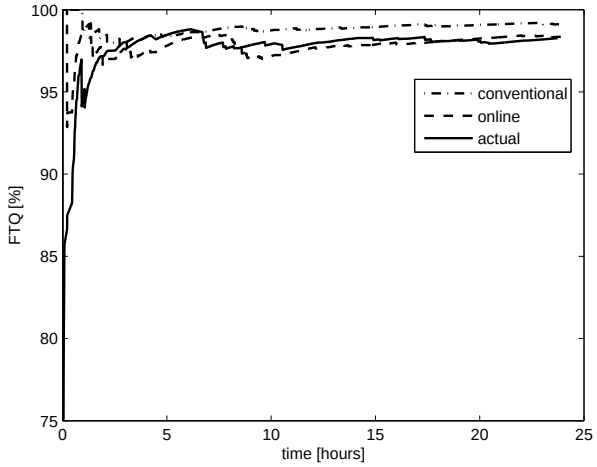

Figure 5.4: FTQ prediction for the production on 3 Januari 2005, based on mean times from the *lambdarun* simulation.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

In the three months of this project, an FTQ prediction program was analyzed and improved. The program uses historical data from the GM St. Catherines assembly line to make a predictions for future quality. It is based on a Stochastic Flow Model of each station in the line. The key parameter in the predictions is the mean time between the occurrences of successive rejections. An improved method to determine this parameter was implemented, which resulted in a more accurate prediction. To study the potential of online parameter estimation, a new simulation program was developed. The program simulates online behavior of the line and makes an online estimation of the mean time between rejections. FTQ predictions based on the newly estimated parameters, turned out to experience great improvement. An adaptive parameter estimation routine was proposed and implemented. The algorithm is based on the theory of Maximum Likelihood Estimation. The method works, but mainly due to limitations of computing power and the used software, in combination with the large variable sizes, no satisfying parameter convergence could be achieved in the within the time span of this study.

In this study a lot of improvement was gained in the computing method of the distribution parameters. This lead to better FTQ predictions for a day's production. The results and findings from this study prove that there is a great potential for an accurate online prediction of the occurrence of rejections.

6.2 Recommendations

In the current Stochastic Flow Model, the mean processing times of all engines in the line are assumed to have an equal constant value. The results showed that this assumption is wrong and influences the predictive quality a lot. In future study it is therefore highly recommendable to improve this.

More research is necessary on adaptive parameter estimation algorithms. The limitations of the used software had a great influence on the possibilities in this study. An implementation of the programs in a more suitable program language like *Fortran*© or *C*© can improve the efficiency a lot. That way variable step size ODE solvers may be used.

For verification of the written programs, only predictions of the total plant FTQ were used in this study. To get to the conclusions of this study, more detailed information was not necessary. To analyse the quality of the estimation of each distribution parameter individually in future research, FTQ predictions should be made, for different stations, engine types and shifts.

Bibliography

[1] Martin Guay, *Using the Dynamic FTQ Model*, unpublished report ,2005, Department of Chemical Engineering, Queen's University, Kingston

[2] Martin Guay, *Recursive Parameter Extimation for FTQ calculations*, unpublished report ,2005 Department of Chemical Engineering, Queen's University, Kingston

[3] P. Egbonunu, M. Guay, *Funcional Flow Chart of the Matlab Programs for the FTQ Model.*, unpublished report, 2005, department of Chemical Engineering, Queen's University, Kingston

[4] H. Yu & C.G. Casandras, *Perturbation analysis for production control of optimizatin of manufacturing systems.*,Automatica, June 2004 Volume 40 Number 6, 945-956.

[5] D.C. Montgomery, G.C. Runger, *Applied Statistics and Probability for Engineers*, 1999, Jonhn Wiley & Sons

Appendix A

Description of the FTQ program files and list of main variables.

The FTQ model generates a prediction of the FTQ for a specified future production schedule. Past planning data and reject history are provided in the form of Excel© data. The model is built of four main files. Two files, *nshift_run.m* and *nshift_data.m*, transform the plant excel sheets to a format that is usable in the simulation and calculate the mean time between rejects. During this process, various function files are called. Most of them are used to transform the available time information into a general usable time indication in seconds. The actual simulation is performed by the file *nsfmallrun.m*, which calls the ODE function *sfmall.m*.

A.1 *nshift_run*

The file *nshift_run* processes the past rejection- and operational data Excel© spreadsheets. First the reject data is loaded into the workspace. *Matlab* divides the data into a numerical data set "*data*" and a textual set "*textdata*". These are renamed to "*reject_data*" and "*reject_textdat*" respectively. The operational data is loaded and remains in the workspace as "*data*" and "*textdata*". The main output of the file is a three dimensional matrix "*exp_shift*". It stores the number of rejections that occurred per workstation per engine type per shift. Table A.1 lists the variables that are used in this file.

Table A.1: List of variables in $nshift\_run.m$

Variable name	Dimensions	Description
mm	1×1	number of rows in the matrix *reject_data* i.e. number of rejections
nn	1×1	number of collumns in the matrix *reject_data*
shift	mm×1	shift identity (A,B or C) in which each rejection occurred
engine	mm×1	with engine codes or each reject
mm2	1×1	number of rows in the array *data*
nn2	1×1	number of collumns in the array *textdata*
enginetype	nengine×1	different engine types present in past *textdata*
nengine	1×1	number of different engine types in past data
nengines_all	1×nengine	number of engines present in past data of each type
npos	1×1	number of stations that experienced rejections
position	1×npos	stations at which rejections occurred
ia/ib/ic	1×1	number of rejects that occurred in shift A,B and C resp
nengine_shift_a	1×npos	total number of each engine type rejected on the current shift
nposition_zone_2	npos×nengine	number of rejections that occurred per workstation 'per engine in current shift
exp_shift	npos× nengine×3	number of rejections that occurred per workstation per engine type per shift

A.2 *nshift_data*

The main outputs of the file *shift_data* are the vectors *timeng* and *shiftn*. The first one lists all execution times in the future data, i.e the times at which the engines will enter the line. The second stores the mean occurrence rate of rejections, $\frac{1}{t_f}$ for each engine at each station. For each engine type, for each station, the occurrence rate is computed by

$$\lambda = \frac{\#rejects}{\#engines\ in\ future\ data \times t_p}, \tag{A.1}$$

where t_p is the process time that is assumed to have a constant value of 24 seconds.Table A.2 lists the variables that are used in this file.

Table A.2: List of variables in $nshift\_data.m$

Variable name	Dimensions	Description
mm2	1×1	number of rows in the matrix *data*
nn2	1×1	number of collumns in the matrix *data*
mm	1×1	number of rows in the matrix $reject\_data$
nn	1×1	number of collumns in the matrix $reject\_data$
rtimeng	1×mm2	times of all rejections in seconds
timeng	1×mm	times of scheduled executions in seconds
engine	mm×1	future engine codes
shift	mm×1	future shift identity
na/nb/nc	1×1	number of of scheduled executions in shift a,b and c resp.
xa/xb/xc	nn1×nengine	part of exp_shift that belongs to shift a,b and c resp. i.e. the number of rejects that occurred per station per engine type
iengine	1×1	position inside *enginetype* of the current
nengines_all	nengine×1	engine typenumber of engines of each engine
shiftn	mm1×nn1	type in past planning datapredicted mean fault times for each engine at each station

A.3 *nsfmall_run*

This program calculates the First Time Quality (FTQ) from the data provided by the files *shift_run* and *shift_data*. A call is made to function *sfm_all.m*. By doing this the SFM is run for all stations for all engines in the production planning. By means of (2.4), the number of rejections is counted. After simulating for the entire production plan, the FTQ is calculated by 3.1. Table A.3 lists the main variables.

Table A.3: List of variables in $nsfmall\_run.m$

Variable name	**Dimensions**	**Description**
tt	mm×1	$timeng^T$ i.e execution times
mm1	1 ×1	number of rows in $shiftn$ i.e. number of planned executions
nn1	1 ×1	number of collumns in $shitn$ i.e. number of stations
mu	mm1×nn1	matrix with random numbers chosen from the exponential distribution with as location parameters the values of $shiftn$
Sx	mm×1	estimated start times of each engine at each station. In the calculation, a fixed production time is used to update tt.
imm	mm1×1	for each engine: stores a 1 if rejected, a 0 if not
delta	1×1	time step that is used in the simulation
engineid	1×nn1	stores which engine is processed at each station
t0	1×1	start time of simulation
Tp	1×nn1	processing time, assumed to be equal for all stations
tf	1×1	ending time of simulation
y	1×3	simulation results (states)
x0	1×3	initial value for ODE; value is updated in every step with the final values of y
tftq	mm1×1	time vector corresponding to the final time values of each simulation it is used for plotting the predicted FTQ.
rejectid	1×mm1	the number of rejections that occurred at each station
imm	mm1×1	stores for each engine a 1 if a rejection is counted, a zero otherwise
ftq_predtot	mm1×1	predicted FTQ for the total plant
ftq_pred	mm1×1	predicted FTQ per reject station
X		state response from simulation
T		time output vector from simulation

Appendix B

Adaptive parameter estimation program

An adaptive update law was implemented to estimate the mean time in a simulation based on past rejection- and operational data. The right hand side of the update law is implemented in the function *lambdadot.m*. The function is called by the script *lambdarun.m*. The input data is provided by running the file $nshift\_times.m$.

The simulation can be started by the following procedure:

- load reject excel© data into workspace
- rename variables *data* and *textdata* to *reject_data* and *reject_textdat* respectively
- load operational excel© data into workspace
- run file *nshift_times.m*, which processes the data to the format required for the simulation
- run file *lamdarun.m*

B.1 *nshift_times*

The program *nshift_times.m* extracts data from the plant excel sheets, that is necessary for the adaptive parameter estimation program and FTQ calculations. The program is built on the conventional program *nshift_run.m*, which counted the the total number of rejects in the past data and sorted this information per station, per engine type and per shift. This extended program performs the same actions as the original. In addition, it generates an estimation of the mean

times between rejections, which can be used as input of the FTQ model. It also generates all necessary inputs for the adaptive parameter estimation program *lambdarun.m*.

Inputs

Past rejection- and planning data, loaded into the workspace as *reject_data*, *reject_textdat* and *data*, *textdata* respectively.

Mean time between rejections

The calculations of the mean time between rejections in this program are made to be more accurate than the ones performed by the conventional FTQ program files. In opposite to the old method, the new file takes into account that some stations that were started, do not go through all stations. When an engine is rejected at a certain station, its processing time is not counted for the stations after the station of rejection. Also, for engines that were started at the end of the measurement period and not able to reach the last station within this period, only the processing times at the reached stations are counted. Taking these things into account resulted in more accurate values

Outputs

Table B.1 lists the relevant output variables of the program.

Function calls

The following function calls are made:

- *weekofyear.m*: returns "weeks" and "years" in which a rejection occurred. i.e. a reject that occurred on January 29, at 0h 56min 52s, will return "weeks = 4.7199" and "years = 2005".
- *date2unixsecs.m*: returns the number of seconds from 1 Januari 1970 at which the input event occurred.
- *shift_id.m*: returns the shift id (A,B,C) at which the input event occured.

Table B.1: List of variables in *nshift_times.m*

Variable name	Description
meanrejtime	3d array of mean time between rejections in the past data. Sorted per station, per engine type and per shift.
exp_time	3d array with number of rejections per station, per engine type, per shift
exp_reject	number of rejections at each station per engine type per shift.
rtimeng	time in seconds from 1 Januari 1970 at which the rejections occurred
ptimeng	starting times in past operational data in seconds from 1 Januari 1970
shift	shift id (A,B,C) for engines in the operational data
engine	engine types for rejected engines
shiftno	shift numbers (1,2,3) for rejected engines
pshiftno	shift id (1,2,3) for past operational data
pshift	shift id (A,B,C) for past operational data
pengine	engine codes in past operational data
restation	station numbers for each rejection
enginenum	engine numbers in past operational data
renginenum	engine numbers in past planning data
rstationpos	position inside variable "position" for each rejection station
rejected	0 if there was no reject for the started engine, position inside reject data if the engine was rejected
enginetype	different engine types that are present in the data
nengine	total number of different engine types in data
penginepos	positions inside variable "enginetype" for each engine in operational data
renginepos	positions inside variable "enginetype" for each rejected engine
nengines_all	number of engines of each engine type present in the past planning data
position	station numbers of stations that experienced at least one reject
npos	number of stations that experienced rejections
nengine	number of engine types in past operational data

B.2 *lambdarun*

Program *lambdarun.m* simulates the production plant and performs an online estimation of the mean time between rejections, $\bar{t}_f$. A call is made to function file *lambdadot.m*.

Output

The output variables of the program are:

- *trmean3tr*: 3d matrix of estimated mean times between rejections
- *trmean3X* : 3d matrix of estimated mean times from adaptive estimation

State vector to matrix

The states of the system are defined as the mean times between rejections for each station, each engine type and each shift. Until now, information sorted by these criteria was stored in three dimensional arrays. Since working with $Matlab$ ODE solvers only allows state vectors, the positions inside a three dimensional matrix have to be transformed to a position inside a vector. Considered matrix A with size $n \times m \times l$, element $A(i, j, k)$ corresponds to a position inside vector $\bar{p}$ following:

$$A(i, j, k) = \bar{p}\left((i-1) \cdot m \cdot l + (j-1) \cdot l + k\right) \tag{B.1}$$

Script

Undergoing, the commented script file is given.

```
% LAMBDARUN
% Author: Kees van Kats
% Lambdarun provides an adaptive estimation of the mean time between
% rejections. The input to the file is provided by running the file
% "nshift_times.m" for the available reject and past planning data. The right
% hand side of the differential equation is provided by function
% "lambdadot.m" It updates the rate parameters of the exponential time
% distribution in the direction of maximum likelihood.
%
% Variables:
% "N":          number of rejects for each case
% "trmean":     mean reject times for each case
% "starttimes": collumn with start times
% "rejtimes":   collumn with reject times
% "renginenum2: reject engine id codes in double format
% "enginenum2:  planning engine id codes in double format
% "lowb":       first engine in reject data that was started in the
%               planning period
% "prodtimes":  3d array with production times for each station for each
%               engine in each shift
% "rejects":    3d array with number of rejects for each case
% "reject:      initially zeros for all start time. Becomes 1 if engine is
%               rejected.

global starttimes Tp rejected prodtimes reject rejects rej_prodtime_old ...
       renginepos shiftno nengine tri g penginepos pshiftno rejtimes N time...
       lb ub statno startd tvec t0 restation check rstationpos position sumtri...
       rej_timeint rejstation rej_prodtime_old time rest trmean lowb xs xr xt...
       trmean_direct

N             = zeros(78*nengine*3,1);       % initial number of rejects
trmean        = 1e10*ones(78*nengine*3,1);   % initial mean times
trmean_direct = 1e10*ones(78*nengine*3,1);   % initial mean times
```

```
% time is set to 0 at the start of the first engine
starttimes = ptimeng - ptimeng(1);
rejtimes   = rtimeng - ptimeng(1);

% in order to use the function "find", engine id codes have to be in double
% format

renginenum2 = 1*renginenum(:,10)+10*renginenum(:,9)+1e2*renginenum(:,8)+1e3*renginenum(:,7)+...
              1e4*renginenum(:,6)+1e5*renginenum(:,5)+1e6*renginenum(:,4)+1e7*renginenum(:,3)+...
              1e8*renginenum(:,2);
enginenum2  = 1*enginenum(:,10)+10*enginenum(:,9)+1e2*enginenum(:,8)+1e3*enginenum(:,7)+...
              1e4*enginenum(:,6)+1e5*enginenum(:,5)+1e6*enginenum(:,4)+1e7*enginenum(:,3)+...
              1e8*enginenum(:,2);

% use only reject data from engines that were started in the simulation period
lowb = 1;
for i = 1:length(renginenum2)
    if lowb == 1
    if length(find(enginenum2(1:50)==renginenum2(i))) == 1
        lowb=i;
    end
    end
end

rejtimes = rejtimes(lowb:length(rejtimes),:);
reject = zeros(length(starttimes),1);

prodtimes = zeros((258-181+1),nengine,3);        % updates process times
rejects   = zeros((258-181+1),nengine,3);        % updates number of rejects
rej_prodtime_old = zeros((258-181+1),nengine,3); % stores  production times at last rejects


load trmean_direct.mat
x0 = (trmean_direct.^-1)';        % initial condition
%x0 = 1e-3*ones(1,78*nengine*3);

x0 = x0';
t0 = 0;                           % start time
lb = 1;                           % start value of search window
g  = 1e-8;                        % gain for adaptive law
Tp = 24;                          % assumed processing time at each station

%Euler ode1: To prevent memory overrun problems, the simulation time
%            had to be cut into pieces.
i = 1;
while i<1581
    if i == 1
        T = linspace((i-1)*1e4,i*1e4,1e4);
        X = ode1(@lambdadot8_2,T,x0);
        %save (['X_' num2str(i-1) '.mat'],'X','T')
        i = i+1;
    else
        T = linspace((i-1)*1e4,i*1e4,1e4);
        [p,q] = size(X);
        x0 = X(p,:)';
        clear X
        X = ode1(@lambdadot8_2,T,x0);
        %save (['X_' num2str(i-1) '.mat'],'X','T')
        i = i+1;
    end
end

save Xend.mat

% In order to be able to use the resulting mean times in the ftq model, the
% state vector of the last time step is transformed to the 3d matrix form
% "orpos" :
```

```
orpos{78*nengine*3} = [];

[n,m,l] = size(prodtimes);
for  o= 1:n
    for j = 1:m
        for k = 1:l
            orpos{(o-1)*m*l + (j-1)*l+k} = [o; j; k];
        end
    end
end

sizeX = size(X);
for hj = 1:length(orpos)
    index = orpos{hj};
    ratesX(index(1),index(2),index(3)) = X(sizeX(1),hj);
    ratestr(index(1),index(2),index(3)) = trmean(hj)^-1;
    ratestr_dir(index(1),index(2),index(3)) = trmean_direct(hj)^-1;
end

% 3D matrix with all possible cases.
trmean2X = ratesX.^-1;
trmean2tr = ratestr.^-1;
trmean2tr_dir = ratestr_dir.^-1;

% format usable in FTQ calculations. i.e only information of the stations
% where rejects occurred
trmean3X = trmean2X(position-180,:,:);
trmean3tr = trmean2tr(position-180,:,:);
trmean3tr_dir = trmean2tr_dir(position-180,:,:);
```

B.3 *lambdadot*

lambdadot

```
% Author: Kees van Kats

% The function "lambdadot" is the right hand side of the adaptive update law
% for estimation of the mean reject times. The rate parameter of the
% assumed exponential distribution is updated in the direction of Maximum
% Likelihood.
% The program simulates online behavior. The simulation time is assumed as
% real time and corresponding are engines entered in the line and rejected.

% For the first station, the production times are updated with the simulation
% time step, if the simulation time exceeds the start time.
% Once a new engine is started, the time for the next
% station is being updated. The production times are stored in the array
% "prodtimes". For engines that were rejected, only stations before the
% reject station can be updated.
%
% variables:
% "startd"       :vector with:  start times for started engines, 0 for other engines
% "statno"       :vector with estimate of the station number at which each engine
%                 is being processed
% "wind_start"   :defines a window in which the data is searched. The window is used
%                 to increase efficiency.
% "tvec"         :simulation time
% "prodtimes"    :production times per station per engine type per shift
% "rejstation"   :stations at which rejects occurred
% "rej_timeint"  :process time interval between last and current rejects
% "rej_prodtime_old: process times at current reject
% "triposition"    : position inside vector "tri", that corresponds to the
%                    position (i,j,k) in the time matrices
% "wind_start"     : defines a window for the start times. The window is moved as
```

```
%                    soon as the first 80 engines in the line are started.
% "reject"         : initially zeros for all start times. Becomes one if an
%                    engine is rejected
% "rest"  : zero if not rejected, station at which reject occurred if rejected
% "N"     : row vector with number of rejects for each case
% "trmean": mean process time between rejects for each case


function lambdadot = lambdadot(t,x)

global starttimes Tp rejected prodtimes reject rejects rej_prodtime_old ...
       renginepos shiftno nengine tri g penginepos pshiftno rejtimes N time...
       lb ub statno startd tvec t0 restation check rstationpos position sumtri...
       rej_timeint rejstation rej_prodtime_old time rest trmean lowb xs xr xt...
       trmean_direct

% a window of observed data is defined to prevent searching unnecessary data
% and thus to improve efficiency
ub = min(200,(length(starttimes)-lb));
wind_start = starttimes(lb:lb+ub);
check = find(wind_start<=t);

% the window is moved if 100 engines were started inside the window
if check(length(check)) >= 100 & wind_start(length(wind_start))~=starttimes(length(starttimes));
    lb = lb+1;
end
wind_start = starttimes(lb:lb+ub);

startd = zeros(size(wind_start));
startd(find(wind_start<=t)) = wind_start(find(wind_start<=t));

% station number at which each engine is being processed
statno = zeros(size(wind_start));
statno(find(wind_start<=t)) = ...
      length(find(wind_start<=t)).*ones(size(find(wind_start<=t))) - find(wind_start<=t) +1;

% for engines that were started and not rejected, times at all stations are allowed to
% be updated with the simulation time step.
xs = find(statno~=0 & rejected(lb:lb+ub) == 0 & statno<=(258-181+1));
if length(xs)>0
    for i = 1:length(xs)
        prodtimes(statno(xs(i)),penginepos(xs(i)+lb-1),pshiftno(xs(i)+lb-1)) = ...
             prodtimes(statno(xs(i)),penginepos(xs(i)+lb-1),pshiftno(xs(i)+lb-1)) + (t-t0);
    end
end

% for engines that were started and rejected, only the times at the stations
% before the reject are allowed to be updated
wind_reject = rejected(lb:lb+ub);
u = find(wind_reject);
v = wind_reject(find(wind_reject));
rest = zeros(size(wind_reject));
rest(u) = restation(v);

xt = find(statno~=0 & rejected(lb:lb+ub) > 0 & rest-180 >= statno);
if length(xt)>0
    for i = 1:length(xt)
        prodtimes(statno(xt(i)),penginepos(xt(i)+lb-1),pshiftno(xt(i)+lb-1)) = ...
             prodtimes(statno(xt(i)),penginepos(xt(i)+lb-1),pshiftno(xt(i)+lb-1)) + (t-t0);
    end
end


% for engines that arrive at the station after their reject station, a reject is counted
% at the corresponding position
% "rejects"             : counts number of rejects
% "rej_timeint"         : time interval since last reject for this case
% "rej_protime_old"     : process times at current reject
```

```
xr = find(statno == rest+1-180 & reject(lb:lb+ub) == 0);
if length(xr>0)
    xr+lb-1;
    reject((xr+lb-1),1) = 1;
    % position inside reject data of current reject
    rejpos = rejected(xr+lb-1)

    % stations of rejects
    rejstation = position(rstationpos(rejpos)) - 180;

    for i = 1:length(rejstation)
    rejects(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1)) = ...
              rejects(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1))+1;

    %reject time interval
    rej_timeint = prodtimes(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1)) - ...
              rej_prodtime_old(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1))

    % process time current reject
    rej_prodtime_old(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1)) = ...
              prodtimes(rejstation(i),renginepos(rejpos(i)),pshiftno(xr(i)+lb-1));

    o = rejstation(i);
    j = renginepos(rejpos(i));
    k = shiftno(rejpos(i));
    [n,m,l] = size(prodtimes);

    triposition = (o-1).*m.*l + (j-1).*l + k
    tripos(xr) = triposition;

    %counts number of rejects per case
    N(triposition) = N(triposition)+1;

    if N(triposition) == 1
        trmean(triposition) = rej_timeint;
    else
        trmean(triposition) = ...
                (trmean(triposition)*(N(triposition)-1) + rej_timeint)/N(triposition);
    end

    if prodtimes(o,j,k) ~= 0 & rejects(o,j,k) ~= 0
    trmean_direct(triposition) = prodtimes(o,j,k)/rejects(o,j,k);
    end

    end
end

% The right hand side of the adaptive law is formed by the derivative of the
% Likelihood function of the exponential distribution with rate parameter lambda.
lambda = x;

if sum(N) == 0
    lambdadot = zeros(length(x),1);
else
    dLdl = N./lambda -  N.*trmean;
    lambdadot = dLdl;

    filt = find(lambda<=1e-5);
    lambdadot(filt) = N(filt)/1e-5 - N(filt).*trmean(filt);
end

lambdadot = g*lambdadot/sqrt(1+norm(lambdadot));

t0 = t;                                          %time of last evaluation
```