Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A DFO technique to calibrate queueing models

T. Begin[a,*], B. Baynat[a], F. Sourd[a], A. Brandwajn[b]

[a]Laboratoire LIP6, Université Pierre et Marie Curie, 4, Place Jussieu, 75252 Paris Cedex 05, France
[b]University of California Santa Cruz, Jack Baskin School of Engineering, 1156 High Street, Santa Cruz, CA 95064, USA

ARTICLE INFO

Available online 21 May 2013

Keywords:
Calibration
Queueing model
Derivative-free optimization
Objective function
Optimization

ABSTRACT

A crucial step in the modeling of a system is to determine the values of the parameters to use in the model. In this paper we assume that we have a set of measurements collected from an operational system, and that an appropriate model of the system (e.g., based on queueing theory) has been developed. Not infrequently proper values for certain parameters of this model may be difficult to estimate from available data (because the corresponding parameters have unclear physical meaning or because they cannot be directly obtained from available measurements, etc.). Hence, we need a technique to determine the missing parameter values, i.e., to calibrate the model.

As an alternative to unscalable "brute force" technique, we propose to view model calibration as a nonlinear optimization problem with constraints. The resulting method is conceptually simple and easy to implement. Our contribution is twofold. First, we propose improved definitions of the "objective function" to quantify the "distance" between performance indices produced by the model and the values obtained from measurements. Second, we develop a customized derivative-free optimization (DFO) technique whose original feature is the ability to allow temporary constraint violations. This technique allows us to solve this optimization problem accurately, thereby providing the "right" parameter values. We illustrate our method using two simple real-life case studies.

1. Introduction

Predicting the behavior of information systems is a key issue in computer analysis. The predictions may be used to design a new system, as well as to forecast performance impact of changes in hardware resources and/or application workload (capacity planning). Predictions can for instance estimate how increasing the utilization of an Ethernet network affects the average delay experienced by transmitted packets. Another case in point is to forecast the performance impact of a new storage device on the performance of the I/O subsystem.

A possible way to address the effect of system or workload changes is to implement the corresponding scenario in the actual system and experimentally assess the resulting impact. Such a solution requires that the system be adequately instrumented, which can be both complex and costly. Additionally, in some instances, the instrumentation and measurement itself can drastically alter the performance of an operational system and compromise its reliability. For these reasons, engineers frequently look for alternative solutions.

Analytical models provide a non-intrusive, reproducible and controlled way to forecast the behavior of a system. Although the models may largely differ in their formalism, complexity, accuracy and solution method, the modeling process can always be decomposed into two steps: a qualitative and a quantitative step. First, a model is designed in order to capture the essential characteristics of the system. This model can rely on predefined formalisms such as queueing networks, Markov chains or Petri nets, or can simply be based on a set of ad hoc mathematical equations. For example, a Web server can be represented as a queueing system [5]. Once this qualitative step is performed, it is necessary to determine the values of the model parameters so that the model and the system match quantitatively. This adjustment of the values of model parameters is referred to as the *calibration* step. Of course, both steps are correlated and the failure of either can invalidate the entire model.

In this paper, we focus on the calibration step of a model of an existing system, i.e., we assume that a potential model derived from queueing theory, has already been developed. The values of some of the model parameters can be determined directly from the knowledge of the system, e.g., from technical specifications or empirical observations. We denote this set of *specified parameters* by $(\alpha_1, \ldots, \alpha_m)$. The values of some other parameters of the model may be unknown to the analyst, and must be somehow determined. We denote these *unspecified parameters* by $(\beta_1, \ldots, \beta_n)$, and we refer to a model that includes such unspecified parameters as an *incomplete model*.

[*] Corresponding author.
E-mail addresses: thomas.begin@lip6.fr (T. Begin), bruno.baynat@lip6.fr (B. Baynat), francis.sourd@lip6.fr (F. Sourd), alexb@cs.ucsc.edu (A. Brandwajn).

The estimation of the unspecified parameters $(\beta_1, \ldots, \beta_n)$, requires additional knowledge that is typically related to measurements. In this paper, we consider that the system under study has been measured at different times corresponding to different levels of the workload. We assume a set of N *measurement points* denoted by $(M_1, \ldots, M_N)$. To each M_i corresponds a specific, but possibly unknown, level of the workload (e.g., a given arrival rate of the incoming requests or a given number of sources of requests). M_i is a vector of *p performance parameters*. In computer engineering, typical performance parameters are the average throughput, the loss probability, the expected sojourn time and the mean queue length, denoted by $\bar{X}^{mes}, L^{mes}, \bar{R}^{mes}$, and $\bar{Q}^{mes}$, respectively. Taking again the example of an I/O controller, a measurement point can be $M_i = (\bar{X}_i^{mes}, \bar{R}_i^{mes})$, relating the expected sojourn time of a request in the system to the average number of requests served per time unit by the controller.

We present a general, easy-to-use and efficient method to derive the missing values of model parameters $(\beta_1, \ldots, \beta_n)$, such that the resulting model provides the "best" fit to the N measurement points. The problem is formulated as an optimization problem. We describe and investigate both traditional and enhanced original definitions of the objective function used to quantify the goodness of fit of the model, with a particular emphasis on practical aspects. We give a high-level description of an original derivative-free optimization (DFO) algorithm we propose to solve this optimization problem. This algorithm is based on a quadratic approximation of the objective function and provides an easy and efficient handling of the constraints imposed on the search for the optimal solution. The proposed DFO technique is reasonably simple and succeeds in calibrating queueing models with less than ten missing parameters.

The paper is organized as follows. In Section 2, we present two simple case studies derived from real-life systems. We use these examples throughout our paper to illustrate the proposed method. In Section 3, we discuss possible definitions of the objective function and we formulate the search for the unknown values of model parameters as an non-linear optimization problem. Section 4 is devoted to the derivative-free optimization algorithm used in the calibration step. Numerical results pertaining to the two case studies are presented in Section 5.

2. Case studies

Throughout this paper we consider two case studies derived from real-life situations. Their complete analysis not only provides examples of successful applications of our "high-level" method, but also serves as paradigm to understand the description of the proposed approach. In both cases, the system under study has been measured and an appropriate but incomplete model has been proposed to represent its behavior, leaving unspecified the values of some of the model parameters. These unspecified parameters values are denoted in this paper as a set $(\beta_1, \ldots, \beta_n)$. In order to be able to effectively use the model, the values of the unspecified parameters must be estimated from system measurements.

2.1. Case study A: Web server

The first case study is extracted from the paper of Cao et al. [5] and relates to the performance of an Apache Web server. The authors present several sets of measurements corresponding to different sizes of requests. In particular, they measure the expected sojourn time (or response time) $\bar{R}^{mes}$ experienced by an incoming request for increasing values of the requests throughput $\bar{X}^{mes}$. We show in Table 1 one of these sets.

Table 1
Performance measurements of a Web server.

$\bar{X}^{mes}$ (req/s)	$\bar{R}^{mes}$ (s)
80.0	1.89×10^{-02}
100.0	3.77×10^{-02}
120.0	5.66×10^{-02}
140.0	2.64×10^{-01}
140.4	$1.43 \times 10^{+00}$

Table 2
Performance measurements of a database system.

S	$\bar{R}^{mes}$ (s)
1	1.53×10^{-03}
2	1.67×10^{-03}
4	2.52×10^{-03}

Cao et al. [5] model their system as a simple M/G/1/K*PS queue [9]. In this queue, customers (job requests) arrive according to a Poisson process, and receive collectively service (time needed to process a request) according to the processor sharing policy. A new request will be rejected if the maximum number of requests allowed at the server, K, is reached. Classical results on robustness show that the average performance of the queue depends on the service distribution only through its average value τ. The mean sojourn time $\bar{R}^{th}$, the average throughput $\bar{X}^{th}$, and the blocking probability Pr^{th}, have closed-form expressions [5,9] that, beside the offered load, only depend on the size K of the queue, and the average service time τ. No specific value for these two parameters can be derived directly from the knowledge of the system. Cast in our formalism, there are two unspecified parameters $(\beta_1, \beta_2) = (\tau, K)$ whose values must be estimated from the measurements. While the authors of the paper estimated their values using a "brute force" exhaustive approach, we use this case study to show, step-by-step, how our approach can manage efficiently the calibration of this model.

2.2. Case study B: machine repairman

The second case study is extracted from a database system benchmarked with varying numbers of users. We denote by S the number of users. Each user can be viewed as a single and independent request source and the whole system is represented as a machine repairman model [9].

Table 2 gives the expected sojourn time $\bar{R}^{mes}$ spent by a request in the system, when there are currently $S = 1$, 2 or 4 sources of requests. Not surprisingly, as the number of users increases so does the mean response (sojourn) time.

Assuming exponentially distributed "machine repair" and "machine up" times, the machine repairman queueing system has four parameters: the total number of users S, the number of servers C, the mean service time per request t_s and the idle request generation rate γ. Such a machine repairman model has a simple closed-form solution [9]. In our case, the number of users S is given but we have no direct knowledge of the remaining model parameters so that $(\beta_1, \beta_2, \beta_3)$ correspond to (γ, C, t_s). As shown in the following sections, our calibration method provides an automatic way to find appropriate values for the three unspecified parameters allowing the machine repairman model to match the measurements.

Many queueing models, such as the ones just described, may have both discrete and continuous parameters. This creates a difficulty since many of the existing optimization methods require all the parameters of the objective function to be either continuous or discrete. We elect to treat all model parameters as continuous by relaxing the discrete constraints on the corresponding parameters. To this aim, we define "intermediate models" in which discrete parameters are replaced by their corresponding continuous extensions. Such intermediate models coincide with "standard" models when their "artificial" extension parameters have integer values. As an example, in case study B, C can be treated as a real number, e.g., $C = 2.87$, just by redefining the departure rates of the associated birth and death process as follows: from state 1 to state 0, the rate of service is μ, from state 2 to state 1, the rate is 2μ, and from any states $n > 2$ to $n - 1$, the rate is 2.87μ. Obviously, this intermediate machine repairman model coincides with the traditional queueing model when C is an integer. Sometimes the intermediate models may require a more involved derivation. In case study A, in order to define the M/G/1/K*PS queue with a non-integer value of K, e.g., $K = 7.35$, one must rearrange the associated Markov chain by adding a new state $K + 1$ and adjusting the arrival rate between state K and state $K + 1$ proportionally to $K - \lfloor K \rfloor$, e.g., 0.35λ. Note that, if only "classical" models (with discrete values for the appropriate parameters) are required following the calibration, we can evaluate all the models obtained from the intermediate model found, rounding the artificial continuous parameters to integers, and keep the one yielding the best results. In certain cases, it may be preferable to perform a further optimization search starting from each rounded set of integer parameters that are kept constant, and run again the optimization technique on the remaining continuous parameters. It is worthwhile noting that these further searches imply only a small additional complexity as the number of parameters of the search is smaller and the starting point is likely to be close to the optimum.

Let us finally emphasize that we chose these two case studies to illustrate our high-level modeling approach for several reasons. First, together they cover the case where the workload is included in the measurements and the case where the workload is unknown, and thus show that the method can be used in both situations. Second, the analytical models associated with the two case studies correspond to different queueing models, the first one being a simple "open" queue and the second one being a closed queuing network. Lastly, both models have discrete parameters, which highlights the ability of our method to handle them.

3. Formulation of the optimization problem

As stated before, analytical models may include both specified and unspecified parameters, denoted by $(\alpha_1, \ldots, \alpha_m)$ and $(\beta_1, \ldots, \beta_n)$, respectively. By definition, the values of the former parameters are supposed to be known. Hence, only the values of the parameters $(\beta_1, \ldots, \beta_n)$ have to be estimated. In this section, we make more precise the notion of the *right* values for the parameters $(\beta_1, \ldots, \beta_n)$ and we formulate their search as an optimization problem.

3.1. The right calibration

Any n-tuple of values for $(\beta_1, \ldots, \beta_n)$ specifies a possible calibration of the model under consideration. In the case study A, where the model is an M/G/1/K*PS queue, the two unspecified parameters, β_1 and β_2, are the mean service time τ and the size K of the buffer. The expected sojourn time $\bar{R}^{th}(\lambda)$ of a request in the queue, as well as the attained throughput $\bar{X}^{th}(\lambda)$, also depend on the load λ, i.e., the average number of requests that are submitted to the system per unit time. Let us assume that (τ, K) is equal to $(6.25 \times 10^{-3}, 250)$. Given that calibration, we depict in Fig. 1 the parametric function $(\bar{X}^{th}(\lambda), \bar{R}^{th}(\lambda))$ as well as the measurement points for the Apache Web server. As can be observed, the performance curve obtained from the model is relatively far from measurement data.

We designate as the "best" values for the parameters $(\beta_1, \ldots, \beta_n)$, the n-tuple that minimizes the deviation between the behavior of the system captured by the measurements, and the performance derived from the model with parameters $(\beta_1, \ldots, \beta_n)$. We thus need to define a function that quantifies the deviation between the measurements and the values produced by the model. We refer to this function as the *objective function*, and denote it by $\delta(\beta_1, \ldots, \beta_n)$ (or just δ for the sake of simplicity). The search for the right calibration can then be expressed as the search of the n-tuple $(\beta_1, \ldots, \beta_n)$ that minimizes δ. The issues related to the definition of δ are addressed in the remainder of this section, while the search procedure is developed in Section 4.

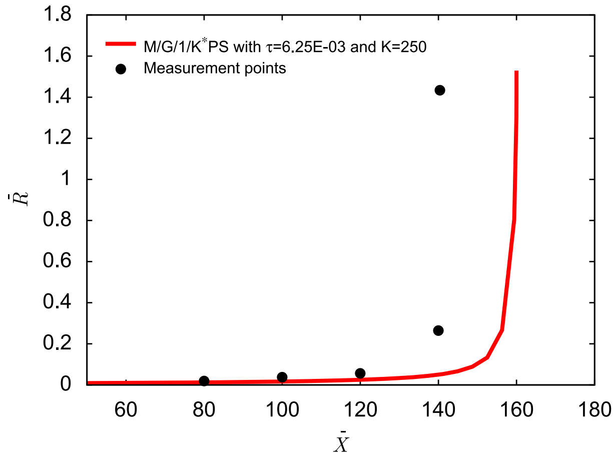

Fig. 1. A possible calibration for the M/G/1/K*PS.

3.2. Constraints

The unspecified parameters of the model, $(\beta_1, \ldots, \beta_n)$, are subjected to three types of constraints.

First, we have simple feasibility constraints: n-tuples $(\beta_1, \ldots, \beta_n)$ are obviously unfeasible as they include non-sensical parameter values. For instance, a mean service time may not be negative. Constraints of this type are usually linear and restrict the domain of definition of δ.

Constraints of the second type are related to the measurements and ensure the consistency of the queueing model with the measurement points.[1] Unlike for the first set of constraints, when these constraints are violated, the model has a meaning and the objective function δ can be evaluated. This feature is actually exploited by our optimization algorithm (see Section 4). In practice, most of these constraints come from the theoretical bounds on the performance parameters of the queueing model. For instance, in case study A, the average sojourn time of a customer in the M/G/1/K*PS queue, for any load λ, is obviously greater than τ and lower than $K\tau$. The model with a given set of parameters (τ, K), is thus inconsistent with the measurements if at least one of the sojourn times measured on the Apache Web server falls outside the range $[\tau, K\tau]$. Similar constraints can be derived for all performance parameters. As an example, for the expected throughput, the model with $(\beta_1, \ldots, \beta_n)$ will be inconsistent with the measurements if its saturation threshold is such that the model cannot reach at least one of the measured values (i.e., some measured throughput values exceed the maximum value of the throughput attainable in the model). Most of these constraints are not linear. Note that because of possible measurement uncertainties and biases, we may need to relax to some degree these constraints. Constraints of this second type will be listed in Section 5 for both case studies A and B.

Constraints of the third type may arise from a partial knowledge of the system. Indeed, one may know that a given parameter β_i although not known directly, must lie within a given range of values.

[1] If none of the possible model calibrations meet these constraints, it probably means that the queueing model is inappropriate.

Since many of our constrains are not linear, the search for the right calibration becomes a non-linear optimization problem.

3.3. *Definition of the objective function*

There are several ways to define the objective function δ, leading to potentially different model calibrations. The definition of an adequate objective function is indeed a crucial step. In this subsection, we suggest a step-by-step method to define a function δ adequate for a large class of queueing systems. We split the problem into two independent subproblems. We first define *coupled points*, by associating with each measurement point a corresponding point obtained from the model. Then we use these coupled points together with the measurement points to define suitable objective functions.

3.3.1. *The coupled points*

The goal of the objective function δ is to measure the deviation between the model calibrated by a given set of parameter values $(\beta_1, \ldots, \beta_n)$ and the corresponding measurements. Although the general idea may seem simple enough, in practice it is not always straightforward to define a measure of the *distance* between the results of the model (which could be viewed as a curve as in Fig. 1) and the discrete set of measurement points. We propose to associate with each measurement point M_i, a coupled point denoted by C_i on the model curve. There are several ways to define such coupled points, each of which corresponds to a given value of the workload.

For the sake of generality,we view the measurement points M_i and the coupled points C_i as vectors of p performance parameters: $M_i = (P^{mes}_{i,1}, \ldots, P^{mes}_{i,p})$ and $C_i = (P^{th}_{i,1}, \ldots, P^{th}_{i,p})$,where $P^{mes}_{i,k}$ and $P^{th}_{i,k}$ denote the value of the kth performance parameter coming from the measurements and the model. We denote by L^{mes}_i (resp. L^{th}_i),the workload level corresponding to M_i (resp. C_i). As mentioned before,workloads L^{mes}_i can be known (i.e.,included or related to the set of specified parameters) or unknown,and can be expressed as an arrival rate,a number of sources or a source intensity. Depending on whether the workloads are known or not,we select the coupled points as follows:

- *Known workloads.* If the workloads L^{mes}_i are known, we propose, quite naturally, to select the coupled points such that $L^{th}_i = L^{mes}_i$. In other words, coupled point C_i represents the performance indices obtained from the model for the same workload as the one used to measure M_i. Case study B provides an example of such a simple situation. Here, a measurement point comprises only the expected sojourn time of a request, $M_i = (\bar{R}^{mes}_i)$, and the workload level corresponds to the known number of sources, $L^{mes}_i = S_i$. Each coupled point $C_i = (\bar{R}^{th}_i)$ is thus obtained by evaluating the model with the same number of sources, $L^{th}_i = S_i$.
- *Unknown workloads.*
  - ○ *The closest point.* When the L^{mes}_i are unknown, the C_i can no more be defined as having the same workload as the M_i. A possible alternative is to select C_i as the "closest" point to M_i on the model curve according to some mathematical distance (e.g., Euclidean norm). In other words, the C_i are selected so that $\|M_i, C_i\|$ is as small as possible. This scheme, together with the Euclidean metric defined in the next section, corresponds to the least squares method [1,3,5,17]. While frequently applied when one deals with the calibration of a queueing model, this method has several drawbacks. First, the choice of the coupled points obviously depends on the units in which the performance parameters are expressed. Second, for a given queueing model, the search for the closest coupled point can be a difficult task, as one must find the value of the workload level L^{th}_i that minimizes the distance between the resulting C_i and M_i. Such an approach is illustrated on Fig. 2 for case study A, where $M_i = (\bar{X}^{mes}_i, \bar{R}^{mes}_i)$.

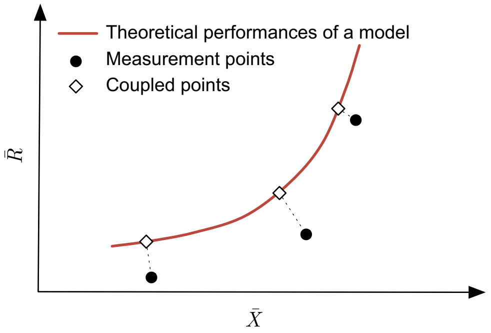

Fig. 2. The coupled point is the closest point from the measurements.

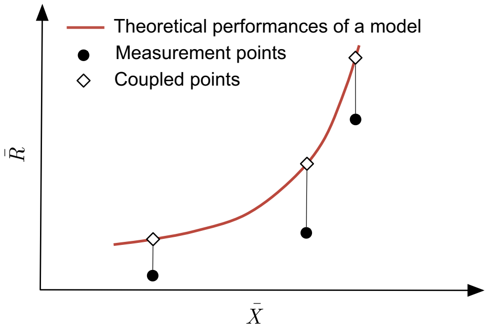

Fig. 3. The coupled point has a common feature with the measurements.

  - ○ *Common value for a performance parameter.* Another possible way to select the coupled points is to define C_i as having one of its performance parameters, referred as to the *criterion* parameter, equal to the corresponding measured performance parameter in M_i. If P^{mes}_k is chosen as the criterion parameter, the C_i are selected so that $P^{th}_{i,k} = P^{mes}_{i,k}$, requiring to find the value of the workload that generates this equality. Note that this search for such a value of the workload is usually much easier than trying to determine the closest point, as it only involves one performance parameter, and, for most queueing systems, performance parameters are monotonously increasing or decreasing functions of the load. The search can thus easily be performed using a simple bisection. To illustrate this scheme, Fig. 3 shows case study A where $\bar{X}$ is chosen as the criterion parameter. Here, the C_i are couples $(\bar{X}^{th}_i, \bar{R}^{th}_i)$ such that $\bar{X}^{th}_i = \bar{X}^{mes}_i$.

3.3.2. *A metric for the deviation*

Once the coupled points are chosen, we need a metric to quantify the deviation between the model curve (sampled by the C_i) and the measurement points (the M_i).

- *An Euclidian metric.* The easiest is to sum the distance between all couples (M_i, C_i), using any mathematical norm, e.g., the Euclidian distance

$$\delta = \sum_{i=1}^{N} \|M_i, C_i\| = \sum_{i=1}^{N} \sqrt{\sum_{k=1}^{p} (P^{mes}_{i,k} - P^{th}_{i,k})^2} \qquad (1)$$

Such a definition for δ has the main drawback to potentially sum together terms with different units, making it difficult to ensure equity between the various performance parameters. This shortcoming may cause severe unfairness when assessing the goodness of fit of a given model.

For the example of case study A, the Euclidian metric results in the following definition for the objective function:

$$\delta = \sum_{i=1}^{N} \sqrt{(\bar{X}_i^{mes} - \bar{X}_i^{th})^2 + (\bar{R}_i^{mes} - \bar{R}_i^{th})^2} \tag{2}$$

- *A multicriterion metric.* A more consistent metric is to evaluate and sum separately the deviation for each performance parameter, avoiding the drawbacks of the Euclidian metric. A weight factor γ_k is associated with each performance parameter, and allows to emphasize some of them in the metric. The definition of the objective function becomes

$$\delta = \sum_{k=1}^{p} \gamma_k \sum_{i=1}^{N} |P_{i,k}^{mes} - P_{i,k}^{th}| \tag{3}$$

For the example of case study A and an equal importance of both performance parameters $\bar{X}$ and $\bar{R}$ (e.g., all $\gamma_k = 1$), we get

$$\delta = \sum_{i=1}^{N} (|\bar{X}_i^{mes} - \bar{X}_i^{th}| + |\bar{R}_i^{mes} - \bar{R}_i^{th}|) \tag{4}$$

Note that when the coupled points C_i are selected so as to have a common performance parameter value with the M_i, then one term in (3) becomes obviously null. In case study A, if $\bar{X}$ is set as the criterion parameter, δ simply becomes

$$\delta = \sum_{i=1}^{N} |\bar{R}_i^{mes} - \bar{R}_i^{th}| \tag{5}$$

3.3.3. Refinements of the metric

So far, the definitions of δ given in (1) and (3) are based only on the absolute deviations between the performance parameters of the measurement points and the coupled points. However, quantifying only absolute deviations may not be adequate in certain cases. In the context of queueing systems, the performance indices may exhibit largely different values from one measurement point to another. In case study A, for example, the $\bar{R}_i^{mes}$ vary from 19 ms up to 1430 ms and, clearly, a deviation of e.g., 10 ms with $\bar{R}_i^{th}$ does not have the same impact on these two very different values. Summing absolute deviations may give an excessive impact to relatively moderate deviations occurring at large values of $\bar{R}$. Conversely, summing only relative deviations may lead to an excessive impact for small deviations on small values of $\bar{R}$. To address these two potential problems, we take into account both relative and absolute deviations in the definition of δ. A parameter, θ, with values between 0 and 1 allows to control the weight of the relative and absolute contribution. As discussed in Section 5.3, we recommend to use a value of 0.5. With this refinement, the multicriterion definition of δ becomes

$$\delta = \sum_{k=1}^{p} \gamma_k \sum_{i=1}^{N} \left(\theta \frac{|P_{i,k}^{mes} - P_{i,k}^{th}|}{\hat{P}_k^{mes}} + (1-\theta) \frac{|P_{i,k}^{mes} - P_{i,k}^{th}|}{P_{i,k}^{mes}} \right) \tag{6}$$

where $\hat{P}_k^{mes} = \sum_{i=1}^{N} P_{i,k}^{mes}/N$ is the arithmetic average of the kth measured performance parameters.

Another possible refinement is related to the uncertainty inherent in the measurements. A measurement point in which the analyst has a low level of confidence should have a smaller impact on δ than a measurement point associated with a high level of confidence. We accomplish this by assigning to each measurement point M_i a weight factor w_i to reflect the level of confidence associated with the measure

$$\delta = \sum_{k=1}^{p} \gamma_k \sum_{i=1}^{N} w_i \left(\theta \frac{|P_{i,k}^{mes} - P_{i,k}^{th}|}{\hat{P}_k^{mes}} + (1-\theta) \frac{|P_{i,k}^{mes} - P_{i,k}^{th}|}{P_{i,k}^{mes}} \right) \tag{7}$$

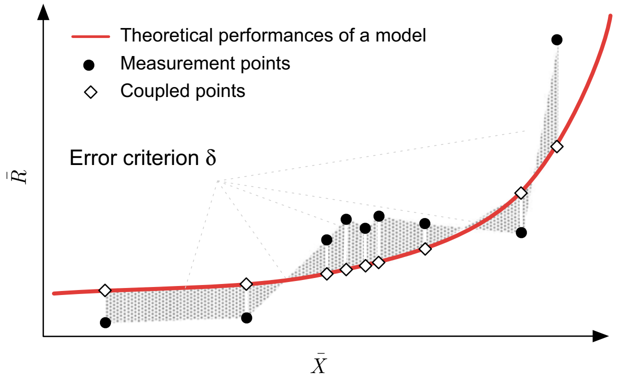

Fig. 4. An area-based function for δ.

Note that weight factors w_i can additionally be used to favor a particular domain of application of the resulting model (e.g., a specific range of workload). Clearly, if all measurements have similar uncertainty levels and if there is no need to emphasize a particular workload region, all w_i are chosen equal.

If we consider again the example of case study A where the C_i are selected so that $\bar{X}_i^{th} = \bar{X}_i^{mes}$, δ is given by

$$\delta = \sum_{i=1}^{N} w_i \left(\theta \frac{|\bar{R}_i^{mes} - \bar{R}_i^{th}|}{\hat{R}^{mes}} + (1-\theta) \frac{|\bar{R}_i^{mes} - \bar{R}_i^{th}|}{\bar{R}_i^{mes}} \right) \tag{8}$$

3.3.4. An area-based metric for the deviation

All metrics considered so far are potentially unfair when a subset of the measurement points forms a *cluster*. By cluster, we denote a group of measurement points located close together with respect to the chosen distance metric. Among the possible situations where clusters are more likely to be encountered, we can cite the case of on-line measurements, where similar measurement values can be repeatedly obtained over time.

Given the previous definitions for δ and assuming equal w_i, a cluster of measurements tends to act as a "super" measurement point, since its weight in δ is roughly proportional to the number of points of the cluster. This reduces the impact of the other measurement points in the estimation of δ, and, as a consequence, forces the model to match with the measurements much more in workload zone of the cluster than in those of the remaining measurement points. This may be an undesirable feature since, in fact, a cluster does not provide much more information than a single measurement point, other than it corresponds to a frequently encountered operating point during the observation period. A possible solution to address this problem consists in decreasing the weights w_i of the points that belong to a cluster. This however would require to define more precisely what is a cluster and how to relate weights to clusters. Even though this might be possible, we prefer an alternate simpler solution, which relies on a new definition of the objective function and which intrinsically takes into account the clustering factor. This new metric is presented below.

If the set of measurements contains clusters, we suggest to use an original definition for the objective function δ, that consists in estimating the area between the measurement points and the theoretical performance curve of the model. The first step is, as before, to associate with each measurement point M_i a coupled point C_i. For this step, all the alternatives developed in Section 3.3.1 can be used. Then, we consider the quadrilaterals made by the union of two consecutive M_i with their associated C_i, and calculate their area. Finally, the overall value of δ is simply estimated as the sum of the areas of all these quadrilaterals. Fig. 4 illustrates this area-based definition for a case similar to case study A including a cluster. Note that by doing so, the workload levels located in the vicinity of the cluster do not overwhelm other levels of workload in the computation of δ. Thus, this new formulation of the objective function addresses intrinsically the risk caused by clusters since its definition takes naturally into account the distance between the M_i. Since clusters tend to appear in on-line measurements, this area-based metric appears to be an efficient and robust choice when calibrating a model on the fly.

4. Solving the optimization problem: computing the parameters

4.1. Derivative-free optimization

As mentioned at the beginning of Section 3, our problem is now formulated as the minimization of a function $\delta(\beta_1, \ldots, \beta_n)$ subject to a set of non-linear constraints $g(\beta_1, \ldots, \beta_n) \geq 0$. The issue is to derive the optimization algorithm to solve it.

In continuous optimization, it is often required that the objective function be smooth and that one be able to compute the Hessian matrix of second derivatives. As function δ can be used with different models, we cannot be certain that δ is smooth and that second derivatives exist. Even if first two derivatives do exist, the expression of δ generally relies on a time-consuming computation of the sum of N terms, each term requiring the evaluation of the steady-state distribution of customers in the queueing model. Therefore it is hard to compute the derivative either formally or numerically. For instance, we tried to solve our problem with SolvOpt, which is based on an implementation of Shor's r-algorithm [8]. This code approximates the derivatives (and does not require the existence of second derivatives) but we encountered poor convergence of the SolvOpt algorithm: we observed that gradients were sometimes wrong, which we believe is caused by the numerical instability in the computation of δ.

Moreover, as mentioned in Section 3.2, for some values of $(\beta_1, \ldots, \beta_n)$, we were unable to evaluate the objective function δ or the left-hand side g of the constraints. When g cannot be computed, it means that the corresponding parameters are not feasible. Since we do not know by how much the constraint is violated, this makes it difficult to implement *penalty* or *barrier* methods [12, Chapter 17].

Based on these observations, the natural framework to minimize δ is the derivative-free optimization [13,15]. Indeed, methods of this family make no assumption on the properties of the objective function and are known to be effective when the computation of the objective function is time-consuming. The oldest method in this class of algorithms is the downhill simplex method (also known as Nelder–Mead method) [11]. In our preliminary tests, convergence was not satisfying. One reason is that the algorithm tried to evaluate δ out of its definition domain, i.e. for some $(\beta_1, \ldots, \beta_n)$ that violate the constraints.

Several routines implementing derivative-free optimization are available on the internet. In a recent paper, Moré–Wild [10] propose a benchmark of the algorithms, namely APPSPACK [6], NMSMAX [7] and NEWUOA [14]. However, these tests are limited to the case of unconstrained problems. In our case, we are dealing with non-linear constraints. Moreover, as mentioned in Section 3, the objective function is not defined for all values of $(\beta_1, \ldots, \beta_n)$, which may be a problem when adapting these routines. We tried APPSPACK, which is able to deal with linear constraints, but we encountered several convergence problems: it works well only when the initialization point is not too far from the optimum. Our explanation is that there are several "flat" regions that mislead the search.

Unlike these state-of-the-art algorithms, the algorithm we present is somewhat heuristic and pragmatic. It mainly works because the objective function as well as the constraints are regular (due to their physical origin).

4.2. Our algorithm

The general scheme of the algorithm is to start with a feasible vector of parameters $\beta^0 = (\beta_1^0, \ldots, \beta_n^0)$ and to iteratively improve it until we find a sufficiently good solution. There is no general heuristic to select the starting vector β^0. The latter is randomly generated within the possible range of each parameter. If the vector violates at least one constraint, the random generation process is repeated until a feasible vector is found. In practice, this Las Vegas algorithm tends to quickly find such a feasible vector.

We now describe a step of the iterative algorithm. We denote by β^k the current feasible vector. The core idea is to locally approximate δ by a quadric function. The minimum of this function can be easily computed and is a good heuristic candidate to improve β^k.

In the following algorithm we will say that a vector, or equivalently a *point* β, is:

- green if it satisfies all the constraints (i.e., it is a feasible vector);
- *orange* if the objective function δ can be computed but at least one constraint is violated;
- red if it is not in the domain of definition of δ (either because δ cannot be computed or because the queueing model involved by the vector of parameters is non-sensical).

At each iteration, there are three main steps:

(1) Randomly select $p = (n^2 + 3n + 2)/2$ non-red points[2] in the "neighborhood" of β^k, the size of this neighborhood is defined by some vector of radii $\rho = (\rho_1, \ldots, \rho_n)$.
(2) Iterate at most $t_{\max} = 10$ times:
    (a) If one of these points is green and improves the objective function δ, it becomes the new current point β^{k+1} and we go to step 1.
    (b) These p points generate a single quadric function f. Let $\beta^\star$ be the critical point where the derivatives of f vanish. If $\beta^\star$ is green and improves the objective function δ, it becomes the new current point β^{k+1} and we go to step 1.
    (c) Randomly replace one of the p points by another one in the neighborhood.
(3) If this step is reached, it means that β^k has not been improved in the $t_{\max}$ iterations of the previous loop. The neighborhood is then reduced by decreasing one of the radii ρ_i. Then we go back to step 1 unless a stopping condition is met.

An original feature of our algorithm is to consider the so-called orange points that violate some constraints. They are only used to build the quadric approximation function, which is especially useful when β_{opt}, the sought optimum, is close to the border of the set of feasible solutions.

In our implementation, the p points chosen in step 1, are selected on the surface of the ellipsoid centered at β^k whose semi-axes are of length $\rho_1, \ldots, \rho_n$. When the size of the ellipsoid is reduced at step 3, we first select the index i that maximizes the quantity $\max\{\delta(\beta_1^k, \ldots, \beta_i^k + \rho_i, \ldots, \beta_n^k), \delta(\beta_1^k, \ldots, \beta_i^k - \rho_i, \ldots, \beta_n^k)\}$, and then decrease ρ_i in order to make the ellipsoid more isotropic. The initialization of the vector ρ depends on their physical meaning (we give more details in Section 5) and we observe that any radius ρ_i is non-increasing during the execution of the algorithm.

Step 2(b) of our algorithm has two parts. First, it requires to determine the parameters of the quadric function such that the

[2] These p points are intended to generate a quadric function.

(hyper-)surface goes through the p selected points.[3] This corresponds to a system of p equations with p unknowns that is solved using a simple Gaussian elimination technique. Second, we need to find the coordinates of the minimum on that quadric. To do so, we calculate its derivative by solving a system of n equations with n unknowns, and use again a Gaussian elimination technique to find its vanishing point.

We use two stopping criteria in our implementation. The first one is met when the largest radius becomes smaller than a given threshold (typically about 10^{-4}) and the second one occurs when $\delta(\beta^k)$ is not improved after a given number of iterations (typically 10^5).

5. Numerical results

5.1. The objective function

The objective function δ represents a key factor when calibrating a model, and, as mentioned in Section 3, its definition has to be chosen in accordance with the type of the measurement data. In case study A, since the workloads that generated the measurements are unknown, the coupled points C_i are chosen as having the same throughputs as the corresponding measurements M_i. Then, δ is evaluated using the multicriterion metric defined by relation (8) with a θ of 0.5. All the w_i are set to an equal value since none of the measurement points is assumed to carry a greater or smaller level of confidence than others.

The (non-obvious) constraints to which the model parameters are subject include the following:

(1) $\max_i(\bar{X}_i^{mes}) \leq 1/\tau$
(2) $\forall i, \tau \leq \bar{R}_i^{mes} \leq K\tau$
(3) $K \geq \max_i(\bar{R}_i^{mes}\bar{X}_i^{mes})$

The first constraint comes from the fact that the average throughput of the model cannot exceed the service rate of the queue. If a given vector violates this constraint, at least one M_i is beyond the model saturation threshold. As a consequence, no coupled point can be associated with M_i (with regards to the chosen definition for δ) and the objective function cannot be computed. The vector is then red (see Section 4.2). The second constraint has already been discussed in Section 3.2, and the third one comes from Little's law [9]. Contrary to the first case, if one of these constraints is violated, the objective function can be computed and the resulting vector is thus orange. The degree of relaxation, due to uncertainties in the measurements (mentioned in Section 3.2), applies only to orange constraints. Here we arbitrarily chose to increase (resp. decrease) by 10% the upper (resp. lower) value of the bounds appearing in constraints (2) and (3). As a general rule, the degree of relaxation can be set to a value comparable to the level of uncertainty associated with the measurements.

In case study B, the measurement points include the workload. As recommended in Section 3.3.1, the coupled points C_i are simply defined as the model response when subjected to the same number of sources as the associated measurement M_i. δ is then computed using the multicriterion metric defined by relation (6) with a θ of 0.5 and w_i of equal values (note that, in this particular case, the chosen metric is equivalent to the Euclidean distance). The single non-obvious constraint to which the model parameters are subject is the following:

(1) $\forall i, t_s \leq \bar{R}_i^{mes} \leq (S_i - 1)t_s/C + t_s$

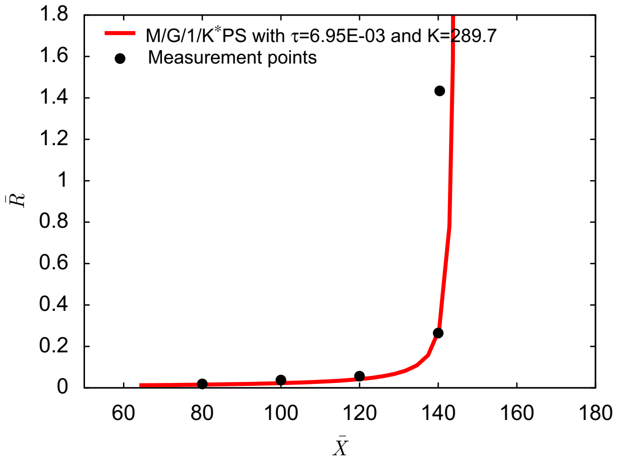

Fig. 5. The calibration found for the M/G/1/K*PS queue.

This constraint comes from the fact that the mean sojourn time of a customer in the queue is obviously greater than the average time to complete a request, and, the maximum expected sojourn time is reached if a customer arrives while all the remaining $S_i - 1$ requests are in process. This constraint is orange as δ can be computed even if a vector violates it.

5.2. Initialization of the DFO technique

First, the DFO technique, as any iterative method, requires a starting vector of parameters, β_0, that is feasible and consistent, i.e., green. Multiple heuristics exist to generate such a vector. For case study A, we ensure that $\beta^0 = (\tau, K)$ is green by selecting an initial value of K greater than $\max_i(\bar{R}_i^{mes}\bar{X}_i^{mes})$ and by randomly choosing τ between $\max_i(\bar{R}_i^{mes})/K$ and $\min_i(1/\bar{X}_i^{mes}, \bar{R}_i^{mes})$. Considering case study B, the initial values of γ, C and t_s are randomly chosen in the intervals $[10^{-5}, 10]$, $[1, 100]$ and $[\max_i(\bar{R}_i^{mes}C/(S_i + C - 1)), \min_i(\bar{R}_i^{mes})]$, respectively. The two former intervals do not arise from model constraints and have been arbitrarily chosen. Note however that this choice is not crucial as it does not restrict the final calibration of the parameters.

Second, we need to set the initial size of the neighborhood of β_0, i.e., to dimension the initial semi-axes ρ_i of the ellipsoid. To the extent possible, we try to set the initial value of each ρ_i an order of magnitude smaller than the expected value of the corresponding parameter. In case study A, if we assume that at least one measurement point corresponds to a high load, we can use the limit conditions of constraints (1) and (3), and initialize ρ_1 to $1/(10\max_i(\bar{X}_i^{mes}))$ and ρ_2 to $\max_i(\bar{X}_i^{mes}\bar{R}_i^{mes})/10$. For case study B, we initialize our DFO technique with $\rho_1 = 0.1$, $\rho_2 = 10$, and $\rho_3 = \min_i(\bar{R}_i^{mes})/10$. Again, the first two values are arbitrary. The last one comes from the lower bound of the constraint.

5.3. The calibrations resulting from the DFO technique

We now compare the model calibrated using the DFO technique with the measurements. For case study A, the calibration process results in values for (τ, K) given by $(6.95 \times 10^{-3}, 289.7)$. In case study B, the calibration process yields the values $(1.7, 152.9, 6.7 \times 10^{-3})$ for (γ, C, t_s). Figs. 5 and 6 show the performance of the calibrated queueing models, together with the measurements points. As shown by these figures, the performance of the models closely matches the measurements. The average deviation between the measurement points and the corresponding coupled points is of 7% for case study A and is less than 1% for case study B.

[3] The standard form for a quadric in two dimensions is: $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$.

Recall that these calibrations have been obtained with a value of $\theta = 0.5$. With $\theta = 0$, i.e., considering only absolute deviations in the computation of the objective function δ (see Section 3.3.3), the DFO returns a calibrated model that better matches the last point. Indeed this point has a much greater value of $\bar{R}$ than the others, thus having a stronger impact in δ. However, the resulting model deviates from the other points, and therefore less accurately reproduces the behavior of the system for a large range of workload levels. On the other hand, if the DFO is run with θ=1, corresponding only to relative deviations, a small measurement error on the first points (with small values of $\bar{R}$) will result in a model that gives them an excessive credit at the expense of others.

5.4. Performance evaluation of the DFO approach

We compared the performance of our DFO algorithm, with three standard optimization techniques, namely the simplex method [11], APPSPACK [6] and SolvOpt [8]. Here we only present the results obtained with SolvOpt as it outperforms the two other techniques. We also compared the results with a “brute force” approach based on explicit enumeration that simply evaluates the model for all combinations of all possible values of the different unknown parameters. This approach requires to discretize the continuous parameters and to introduce reasonable lower and upper bounds for all of them.

The CPU time, the number of evaluations of δ (denoted by #δ), as well as the rate of successful convergence, referred as to the *robustness*, are used as measures of algorithmic performance. Each algorithm was implemented in C and was run on a 2 GHz Intel Core 2 Duo processor. We ran thousands of experiments corresponding to different starting vectors β_0. Table 3 shows the average performance of the three algorithms for both case studies A and B.

Our results indicate that our DFO algorithm gets a significantly better rate of convergence than SolvOpt while being somewhat slower. Furthermore, results not shown in this paper indicate that its complexity (expressed through #δ) grows reasonably with the number of unspecified parameters, unlike the “brute force” approach which is combinatorial. In fact, the complexity of our DFO algorithm is close to polynomial since the number of points required to build the quadric function, which determines the number of evaluations of the objective function, increases in $O(n^2)$ with the number of parameters, n. However, this rudimentary estimation does not account for the fact that the approximation of a complex surface by a quadric function is likely to fail more often as the number of parameters to calibrate increases. This forces the algorithm to generate additional quadrics and increases the overall computational complexity of our technique.

Despite its limited size, this study supports our general experience with the DFO algorithm presented. Unlike the existing techniques we have tried, our algorithm appears easy-to-use, reasonably fast and highly effective on all tested examples [2]. More specifically, for queueing models with less than about 10 unspecified parameters, it appears as a good trade-off between gradient-based methods that are not robust enough (e.g., SolvOpt) and intractable systematic approaches.

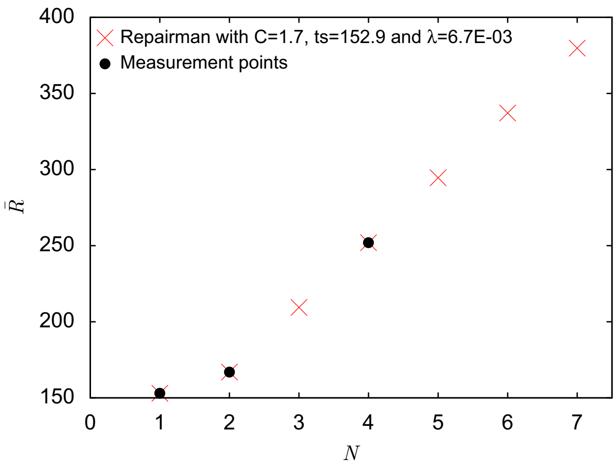

Fig. 6. The calibration found for the machine repairman.

5.5. Making use of the calibrations

So far, we have shown the ability of our DFO technique to deliver accurate calibrations of a model. Having a properly calibrated model allows a better understanding of the system and a deeper insight into its behavior, and provides a valuable tool for making predictions. Here we illustrate some of the possible uses of our automatic calibration tool. (Other examples can be found in [2].)

A model can be of interest to forecast performance at workload levels for which measurements may not have been obtained. To test the prediction capability of our approach, we use case study A and deliberately remove one data point from the measurement set. We run our DFO technique on the remaining measurements and test the ability of the resulting calibrated model to predict the system performance at the removed data point. Our results show that in all cases, including the case for the highest load level, which is likely to be the most difficult to reproduce accurately, the calibration yields accurate predictions for $\bar{X}$ (the deviation is less than 3%). Predictions for $\bar{R}$ are given with an error lower than 2% except for the last point. Note however that in this latter case, the saturation threshold of the model closely matches the measurements.

Thus, selected capacity planning issues can be addressed with our method. For instance, in case study A, assuming a growth of 10% in the workload in the vicinity of the second-to-last measurement point, the model forecasts only a 3% degradation in the average sojourn time that a request spends inside the Apache Web server. In case study B, the queueing model can be used to estimate the performance of the database system if the number of users increases to 8. Still for case study B, the model can be used to determine the maximum number of sources such that the mean sojourn time of a request stays below a given threshold, e.g., for 350 ms the corresponding maximum number of sources is 6 (see Fig. 6).

Finally, in many cases, the proposed method can provide insight into the system under consideration by estimating parameters that cannot be measured. As an example, for case study A, the first

Table 3
Performance of the three optimization algorithms.

Case study	DFO			SolvOpt			Brute force	
	CPU	#δ	Robust (%)	CPU	#δ	Robust (%)	CPU	#δ
A	359 ms	1418.9	100	247 ms	1127.3	77	75 s	1.49×10^6
B	11.1 s	31103.0	89	6.2 s	17851.6	23	> 10 h	1.12×10^9

unspecified parameter gives an estimation of the average time the Apache Web server needs to treat a request. The value of this parameter would have been very difficult to obtain otherwise as it represents the aggregate of several interactions between various components of the server.

6. Conclusion

In this paper, we have presented a general, easy-to-use and efficient method for the calibration of queueing models. Given a set of measurements collected from an operational system, the method aims at finding the values of the model parameters that provide the best fit between the values produced by the model and the measurements data. We have formulated the search for the parameters as a continuous optimization problem, with a particular focus on defining a consistent objective function to quantify the "distance" between measurements and model results. We have described in detail a simple DFO algorithm, based on a quadratic approximation of the objective function, to solve the optimization problem. Nonlinear constraints can be easily incorporated into our DFO algorithm, without hampering the efficiency of the procedure. This represents a clear advantage in our context, since most queueing models are subject to constraints. Finally, as our calibration method only requires the evaluation of the objective function, the approach can generally be applied to any queueing model, including those solved using a numerical method or via simulation. However, as the overall speed of our method depends directly on the time required to solve the model and to compute the objective function, fastest execution will be obtained with a speedy underlying model solution.

The proposed method has been successfully applied in several real-life problems [2]. Its domain of application covers a wide variety of systems, including computer systems and communication networks, e.g., I/O disk controllers, processors, wired and wireless networks. Note that our DFO technique is suitable for models with less than ten unknown parameters, which is enough for most situations. However, if the number of parameters is higher, a more sophisticated DFO technique with a higher computational overhead [15], can be used within our calibration framework.

Additionally, we have presented two simple case studies, both derived from real-life systems, to illustrate the application of the method, its usefulness and its efficiency. Based on these two case studies, we have shown that our DFO technique is significantly more robust than other existing methods at the expense of only moderate increase in its search time. This has been our experience in a number of real case studies.

We have explored some of the possible applications of a properly calibrated model, e.g., a better understanding of the system, a deeper insight into its behavior, and a tool for making predictions. Furthermore, because of its simplicity and speed, the calibration method can be used on operational systems with on-line measurements, by continuously updating the calibration of the model and then providing prediction on the fly. This promising usage of our calibration method is currently under investigation.

Acknowledgment

The authors would like to thank Safia Kedad from Université Pierre et Marie Curie for her valuable help in designing the specific derivative-free optimization algorithm.

References

[1] Abatzoglou T, Mendel J. Constrained total least squares. In: Proceedings of the IEEE international conference on acoustics, speech, and signal processing, ICASSP; 1987. p. 1485–8.
[2] Begin T, Brandwajn A, Baynat B, Wolfinger B, Fdida S. High-level approach to modeling observed system behavior. SIGMETRICS Performance Evaluation Review 2007;35:34–6.
[3] Björck A. Numerical methods for least squares problems. Cambridge; 1996.
[5] Cao J, Andersson M, Nyberg C, Kihl M. Web server performance modeling using an M/G/1/K*PS queue. In: ICT'2003: 10th international conference on telecommunications, vol. 2, 2003. p. 1501–6.
[6] Gray GA, Kolda TG. Algorithm 856: APPSPACK 4.0 asynchronous parallel pattern search for derivative-free optimization. ACM Transactions on Mathematical Software 2006;32:485–507.
[7] Higham NJ. The matrix computation toolbox ⟨www.ma.man.ac.uk/~higham/mctoolbox⟩.
[8] Kappel F, Kuntsevich AV. An implementation of Shor's r-algorithm. Computational Optimization and Applications 2000;15:193–205.
[9] Kleinrock L. Queueing systems, vol. 1: Theory. New York: Wiley; 1975.
[10] Moré JJ, Wild SM. Benchmarking derivative-free optimization algorithms. SIAM Journal on Optimization 2009;20:172–91.
[11] Nelder JA, Mead R. A simplex method for function minimization. The Computer Journal 1964;7:308–13.
[12] Nocedal J, Wright SJ. Numerical optimization. Berlin: Springer; 2006.
[13] Powell MJD. Unconstrained minimization algorithms without computation of derivatives. Bollettino della Unione Matematica Italiana 1974;9:60–9.
[14] Powell MJD. The NEWUOA software for unconstrained optimization without derivatives. In: Di Pillo G, Roma M, editors. Large scale nonlinear optimization. Netherlands: Springer; 2006. p. 255–97.
[15] Powell MJD. A view of algorithms for optimization without derivatives. Cambridge NA Reports. Optimization Online Digest; June 2007.
[17] Van Huffel S, Vandewalle J. The use of total least squares techniques for identification and parameter estimation. In: Proceedings of the seventh IFAC/IFORS syrup on identification system parameter estimation; 1985. p. 1167–72.