The Relationship of Performance Models to Data

Murray Woodside

Carleton University, Ottawa, Canada
cmw@sce.carleton.ca

Abstract: Performance engineering of software could benefit from a closer integration of the use of performance models, and the use of measured data. Models can contribute to early warning of problems, exploration of solutions, and scalability evaluation, and when they are fitted to data they can summarize the data as a special powerful form of fitted function. Present industrial practice virtually ignores models, because of the effort to create them, and concern about how well they fit the system when it is implemented. The first concern is being met by automated generation from software specifications. The second concern can be met by fitting the models to data as it becomes available. This will adapt the model to the new situation and validate it, in a single step. The present paper summarizes the fitting process, using standard tools of nonlinear regression analysis, and shows it in action on examples of queueing and extended queueing models. The examples are a background for a discussion about the relationship between the models, and measurement data.

1 Motivation

Software performance modeling and measurement are insufficiently integrated. Roughly speaking we may say that measurement is used to test software and to identify performance problems, often in laboratory conditions; modeling is used for prior analysis of planned systems (when there are no measurements available), for capacity and scalability analysis (exploiting the capability to model large deployments), and for insight into deep problems. There are a number of exceptions; one is in tracking performance models for adaptation of time-varying systems (e.g. [19]).

In [20] the potential benefits of more strongly unifying these two aspects of performance analysis were identified as

- end-to-end performance process unifying prior estimates with testing/debugging and capacity modeling
- better quality models calibrated frequently and routinely from data, using a strong and maintained connection between the model and the system,
- more efficient measurement, by using models to plan the measurement trials.

The weak link here seems to be the calibration of models from data. Long experience in creating models teaches how difficult it can be to determine their parameters empirically. Some critical kinds of data are often difficult to obtain, notably the CPU demands of

particular operations. However, recent work on tracking model parameter values with a Kalman Filter estimator has indicated how data-gathering can be eased by use of a suitable estimator [20]. That work was for tracking parameters which are changing. This paper considers a different problem, modeling a system which is not changing, but which is operated under different workloads and configurations. It describes a simple framework for estimating a model by nonlinear regression, and some properties of the resulting model (including how it differs from one made up from expert knowledge alone). Standard statistical concepts provide a bridge between the practice of measurement and the practice of modeling.

The reduction of difficulty comes from indirect estimation of model parameters, using only measurements at the interfaces of the system. A prime example is the difficulty of estimating the CPU demand of a particular operation. Direct measurement requires source-code instrumentation (as in profiling, for instance) but a model-based estimator only requires accessible performance measures such as operation response times, and then estimates the CPU demand to fit the measured values.

Recent efforts to calibrate CPU demands from accessible measures include [10] and [3], in which individual operations were measured in separate benchmark experiments, and the patent application [16], which applies an optimization technique to fit a queueing model to data. The patent motivation includes the rationale:

> "There should be a simple method to estimate the parameters of the model, given high-level system measurements obtained by external monitors, rather than adding instrumentation with detailed level measurement probes to applications." [16]

The present paper combines ideas from statistics textbooks and the performance modeling literature, and shows how they might be able to unify the practice of performance engineering of software. Its methods can be applied to *any* performance modeling formalism. Relevant background for modeling is given by Smith and Williams [14], Balsamo et al [1], and in the proceedings of the WOSP conference [21]. Representative material on measurement is provided in the Paradyn papers by Miller and co-workers (e.g. [11]), and by the online tutorial [2].

2 Software Performance Analysis

2.1 A Unified Process

The vision of a unified process, which this paper intends to support, is sketched in Figure 1 reproduced from [20]. Performance predictions and data feed a common data repository with a united definition of the semantics of predicted, measured and required values, defined during requirements analysis. Performance values are derived early and evolve

with the design and the product. Performance knowledge can be leveraged, rather than being abandoned soon after it is produced. The awkward fact that performance is a moving target and one never measures or models the same system twice, is covered by versioning the data in synchrony with versions of the design, the run-time configuration and the code.

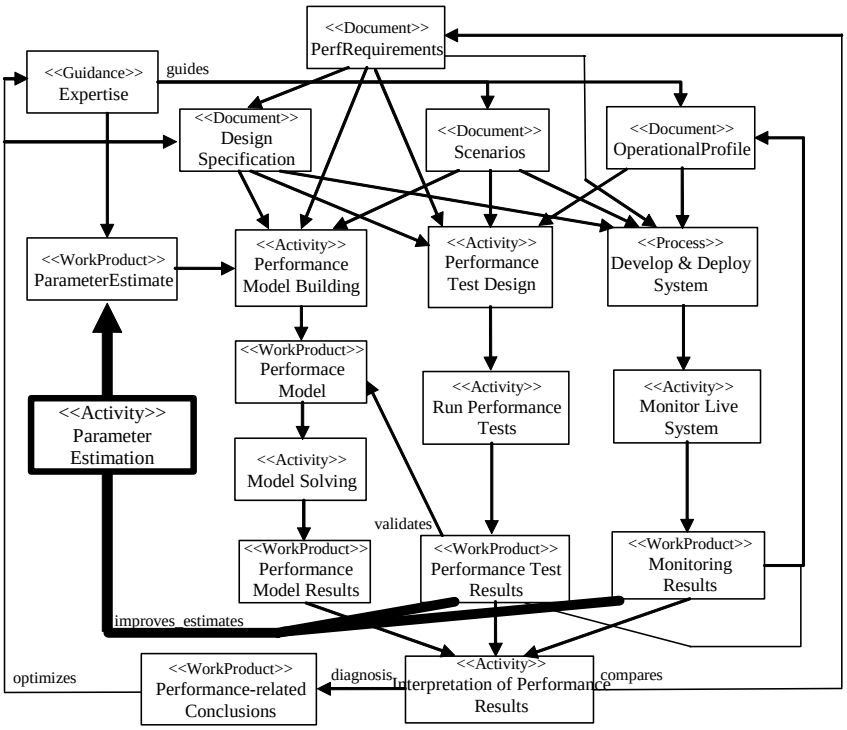


Fig. 1 A proposed landscape for a unified SPE process, from [20]

Additional activities that could be added to this figure. Model-generated performance prototyping for example would fit in above "Develop and Deploy" on the right.

The estimators described in this paper provide the links shown by the very heavy arrows, by which parameter values are calibrated from data from a test or an operational system. These links maintain the parameters of the model as the software is completed and make the model available for planning deployments and new versions of the system.

The estimators form a bridge between what we will term a data-centric view of performance, on the right-hand side of Figure 1, and a model-centric view, on the left.

2.2 Data-Centric View

An extreme data-centric view is that only measurement data is significant, because only the data can capture the full complexity and interactions of the system. Measurements are carried out in tests or trials, in which a controlled and instrumented configuration is operated under a specified workload, and as much data is recorded as possible, including performance measurements at system interfaces, and measures on internal operations.

A high-level view of system measurement is sketched in Figure 2(a), showing the inputs and outputs of a single measurement test or trial, identified as trial number t:

- $\mathbf{u}_t$, a vector of *controlled parameters* whose values can be assigned in a measurement experiment or a proposed configuration. These are system parameters that could be significant for performance. They might include the number of processor cores, the size of a thread pool, cache or buffer pool, the size of files to be transferred or of database transactions. Qualitative attributes of a configuration, such as the use of a particular middleware or component, or the presence/absence of some feature, can be included through categorical variables in u, but will not be considered here.
- $\mathbf{z}_t$, a vector of measured values of any performance quantities of interest. These can be average values, percentiles of delay distributions or any other well-defined measure. We assume the system is stationary, and for any value of the controlled parameters the measures have defined and repeatable values apart from sampling error due to a finite measurement period.

A series of measurement trials gives a tabulated relationship between the controlled variables **u** and the measured responses **z**, which for one component of each can be plotted as shown in Figure 2(b). The measurement errors are indicated as a shaded band.

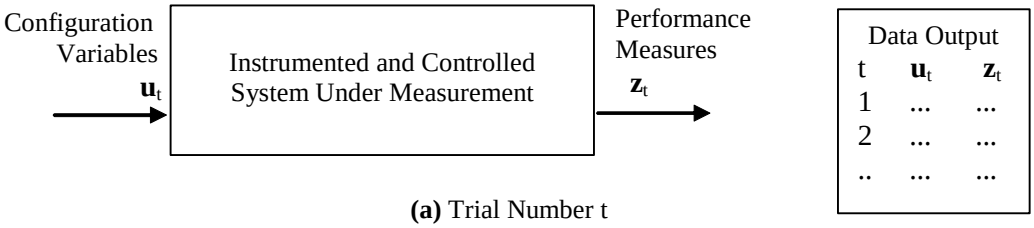
Many kinds of measurement systems have been used. We can categorize types of data into four groups, in increasing order of difficulty to obtain it:

1. data gathered by the operating system, such as processor and process utilizations, and I/O counts
2. data collected at system interfaces by timers and counters,
3. profiling data on CPU usage by operation, down to quite fine grained methods, using embedded source code instrumentation (in for instance Purify [4]) or stack sampling (as in gprof).
4. Logical resource usage, such as critical section monitoring in Paradyn [12].

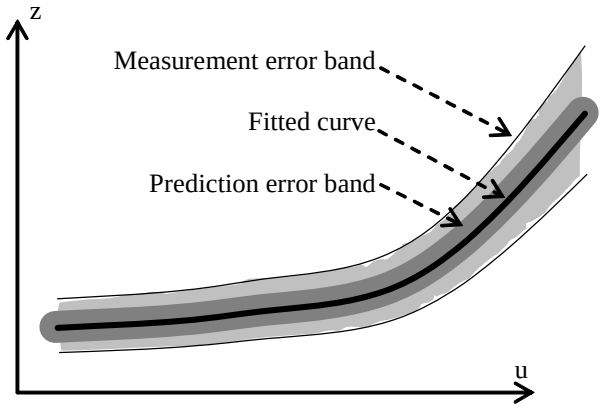
In much of the performance analysis of commercial software it is not practical to use the more sophisticated tools in groups 3 and 4, because of time and cost, and because source code is not available for third-party components. And those tools often distort the system by introducing significant fine-grained costs. Field measurements are often restricted to groups 1 and 2, and even in the lab there is an advantage in only requiring the simpler forms of instrumentation.

In measurements to support the model fitting described below, the performance measures are delays measured at component and messaging interfaces, and utilizations of processors and processes. If a model is fitted to the data, shown as a solid line, its

predictions also have a prediction error indicated by the darker shaded band. This band is narrower than the measurement error band, because the fitting process smooths out the errors of individual measurements.



(a) Trial Number t



(b) Results of Fitting a Model

Fig. 2 Inputs and outputs of a system performance measurement trial

2.3 Model-Centric View

The model-centric view seeks an abstraction that captures the essence of the system performance in its simplest form. Performance models often do not start from data. For example, to provide insight into performance issues during the system design phase, performance models can be created based entirely on the design and on expert judgment [14]. A given model has a structure, based on the elements of the design, and parameters which describe what the system will do. (The particular modeling method to be used is not our focus here, but is surveyed in [1].) We should regard the model calculation as a vector function **h(x, u),** as illustrated in Figure 3:

u = configuration parameters as before

x = parameter values in the model, which must be obtained by some process

y = vector of predicted performance measures = **h(x, u)**

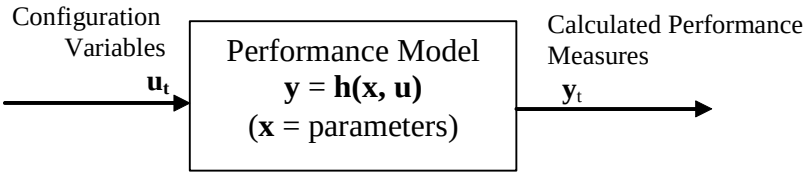


Fig. 3 Performance model as a function

Despite the difficulties in obtaining parameter values, Smith and Williams (who have developed procedures for gathering structure and parameters for early models) show that the results have many practical uses in practice [14]. Their approach includes the important notion of expressing a parameter as an interval expressing expert judgment as a range of values parameter value in the range [min, most likely value, max].
Using these interval values we can show the model predictions as a plot with a central value and an uncertainty band, for parameters within these intervals, similar to the darker band in Figure 2(b).

It may be confidently stated that good insight into structure is often available, but the parameter values are often problematic. *The main barrier to usability of these early models lies in lack of confidence in the parameter values.*

The Bridge: Estimation

To fit a model to experimental data it is common to estimate its parameters directly, by measuring the property represented by the parameter. For CPU demand parameters (for instance) this often requires recording the CPU associated with each operation, as in profiling. Other kinds of parameters include the relative frequency of different operation invocations or messages, and the sizes of data objects.

Here we consider a more general version of estimation, which includes direct estimation as a special case. We assume a model structure is determined, within the chosen formalism, from expert knowledge or system design documents. To this structure we attach three kinds of parameters:

- *assumed parameters*, whose values are known and do not vary during the estimation process or in planned deployments. We will not consider these further here; they are lumped in with the structure.
- controlled parameter vector **u**, as above, taking value $\mathbf{u}_t$ in trial t,
- estimated parameter vector **x**, assumed to be constant.

and performance measures of interest, given by

- vector **y** for the model and

- vector $\mathbf{z}_t$ for measurement trial t, also as above.

The result of a series of trials is a pair of sequences $\mathbf{z}_t$ and $\mathbf{u}_t$ for t = 1,...Tmax.

A standard basis for estimation, which we will use, is to maximize the likelihood of the model. We assume that the measurements **z** are determined by an unknown function **h(x, u)** (to be found) plus a measurement error vector **v**:

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}, \mathbf{u}_t) + \mathbf{v}_t$$

Assuming that vector $\mathbf{u}_t$ is independent over time with a joint normal distribution with mean zero and covariance matrix **R**, we obtain the maximum-likelihood estimate as the vector **x** that minimizes E(**x**):

$$\hat{\mathbf{x}} = \arg\min E(\mathbf{x}), \quad E(\mathbf{x}) = \Sigma_t (\mathbf{y}_t - \mathbf{z}_t)^T \mathbf{R}^{-1} (\mathbf{y}_t - \mathbf{z}_t)$$

If we only know **R** to within a constant, the constant can also be estimated. If components of **v** are independent with the same variance, **R** is proportional to **I** and $\hat{\mathbf{x}}$ is the familiar least squares estimator. Estimates made with **I** in place of **R** (that is, plain least-squares estimates) are unbiased but less accurate.

This is a standard optimization problem, which can be solved in many ways. Many of these exploit the special structure of E(**x**), which is a quadratic form. A standard approach, treated in statistics texts such as [8], is Gauss-Newton iteration which gives an approximate solution through a series of linear regressions. Gauss-Newton iteration was used to analyze the example given below. Since it is not a widely-known procedure, the adaptation of Gauss-Newton iteration to performance models is given in detail in the Appendix.

If $\mathbf{x}_t$ is not assumed to be constant, then it can be modeled as a function of time (provided the trials are regularly spaced in time) and estimated with an optimal filter such as the Kalman filter; this case is not considered further here.

2.4 Inference: Knowledge and Uncertainty

A major impediment to the use of any kind of model (not just a performance model) is the feeling, in a potential user, that one should not trust the predictions if one does not understand the limitations of the model. A substantial part of these limitations is, the prediction uncertainty due to inaccurate parameter values, and these can be estimated as confidence intervals. This informs the potential user of the accuracy, which may be different for different measures coming from the model, and poor accuracy may identify the need for more information, depending on the decisions to be made. It also places the predictions into a familiar framework of statistical predictions and statistical quality control, which industrial decision makers can deal with. For example, if one can give the

probability of missing a performance target by different amounts, it enriches the consideration of risk.

Within these confidence bounds, the model becomes a representation of the data.

Using the approximate maximum likelihood and non-linear regression framework considered here, estimation errors and prediction errors are effectively assumed to be normally distributed. Normality is a reasonable assumption for measurement errors which are averages or sums, due to the central limit theorem. However the nonlinearity of the model reduces the validity of the assumption for estimation and prediction errors.

The most basic and familiar representation of uncertainty takes the form of confidence intervals, which are found as part of the standard inference results for regression:

- confidence intervals for the parameters $\mathbf{x} = \hat{\mathbf{x}} \pm \mathbf{CIx}$
- confidence intervals for the predictions $\mathbf{y} = \mathbf{h}(\hat{\mathbf{x}}, u) \pm \mathbf{CIy}$

where the vector **CIx** is the confidence interval half-widths for **x**. As we will see, parameters that make little difference to the prediction tend to be estimated with large confidence limits.

3 Illustration: Queueing Model

The small queueing network model shown in Figure 4 will be considered as the first example. It represents a small Web server with its disk (node 2) and a separate node for CGI application service (node 3). A response includes all the work done between visits to the "Users" node in the Figure, which represents the operation in which a user responds to one system output and generates the next request to the system. Users have a characteristic "think" time for this operation, which will be set to zero here. Service times are assumed to be exponential. We consider one controllable parameter, three demand parameters to be estimated, and four measures.

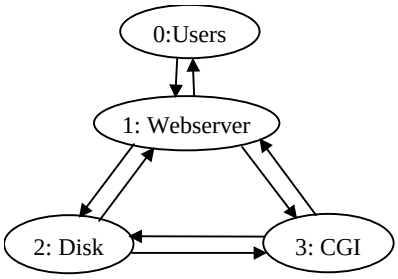


Fig. 4 A small queueing model

Then the queueing model has four parameters:

u = N = the number of active jobs, assumed to be constant (so this is a "closed" model), with default value 4,

x = [x(1), x(2), x(3)] = the total average demands for service by nodes 1, 2 and 3, with actual values [2, 3, 4] sec/response.

y = [y(1), y(2), y(3), y(4)] = [T(1), T(2), T(3), f], where T(i) is the mean response time of node i, totalled over a user response, and f is the throughput of user requests.

The model is assumed to satisfy the separability conditions for product form queueing networks, which means that it can be solved by Mean Value Analysis (MVA) [5].

The data for illustrating the use of nonlinear regression were obtained by simulating the same queueing network for different durations, and with different numbers of users, as shown in Figure 5. Clearly this oversimplifies the fitting problem, since the performance model ought to fit to some degree. However it serves to demonstrate that the method can find the right model, and it illustrates the important issue of accuracy of the fitted parameters.

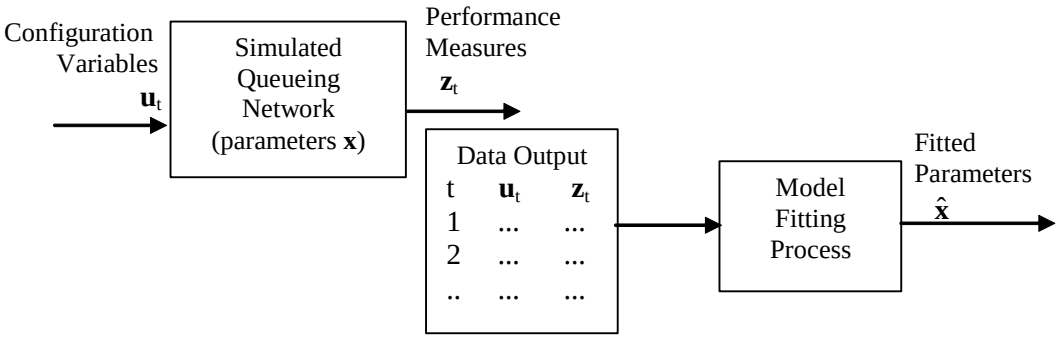


Fig. 5 Configuration of the Computations for the Illustration

To simulate measured data, the system was simulated for 10 trials, each of duration S time units, with from 1 to 10 users; node 3 approaches saturation at a population of about 8. S was varied from 1000 time units to 100000, and the total simulation time was varied from 10^4 to 10^7 time units. The throughput ranged from about 0.1/time unit with one user, to 0.25 with 10 users, so a trial of length 1000 includes between 100 and 250 responses. In the estimation, analytic derivatives were computed for the H matrix by extending the MVA algorithm as described in [19].

Two sets of experiments were performed with zero "think time" at node 0, and with a mean think time of 10 units. Table 1 shows the demand parameter estimates and their confidence intervals. Since the random think time introduces additional variation in the data, it is not surprising that most confidence intervals are a little wider for the second set.

Table 1 Queue Model Parameter Estimates and their Confidence Intervals

Expt/Trials/ Length of Trial	Think Z	$\hat{\mathbf{x}}$ (1) ± CIx(1)		$\hat{\mathbf{x}}$ (2) ± CIx(2)		$\hat{\mathbf{x}}$ (3) ± CIx(3)	
A1/10/1000000	0	2.003	0.0109	3.005	0.0108	4.005	0.0087
A2/10/100000	0	1.992	0.0223	2.989	0.0222	4.011	0.0178
A3/100/10000	0	2.011	0.039	3.077	0.038	4.044	0.031
A4/10/10000	0	2.018	0.018	2.994	0.018	4.004	0.014
A5/10/1000	0	2.182	0.204	2.967	0.217	3.848	0.181
B1/10/1000000	10	2.005	0.0089	2.994	0.0097	4.009	0.0081
B2/10/100000	10	2.003	0.0215	3.010	0.0234	4.036	0.0194
B3/100/10000	10	2.001	0.025	2.999	0.027	3.941	0.023
B4/10/10000	10	1.939	0.072	2.944	0.078	3.925	0.065
B5/10/1000	10	1.854	0.221	2.798	0.247	3.980	0.188

From the results in Table 1, we can see that:

- the confidence intervals are tighter for the largest demand values (server 3), which corresponds to the most saturated resources. This is very natural, since the regression is controlled by the sensitivity of the performance measures to the parameters. This is shown by the sensitivity matrix H, which at convergence (in experiment B5, but the others are similar) is

  $$\mathbf{H} = \begin{matrix} \mathbf{2.9219} & -0.3164 & -1.8522 \\ -0.1388 & \mathbf{5.6991} & -3.5126 \\ -0.4019 & -1.7367 & \mathbf{10.3132} \\ -0.0044 & -0.0119 & -0.0474 \end{matrix}$$

  The columns represent the sensitivity of the four measures (three node response times and one throughput) to the three node CPU demands. The bold values show that each server response time is most sensitive to its own service time, but the other values show larger magnitudes in column 3 than the other columns. In particular row 4 showing sensitivity of overall system performance, has its largest element in column 3.
- Although server 2 is more heavily utilized than server 1, (and its demand shows greater sensitivity in the H matrix) the confidence intervals for their demands are similar, since neither is determining for performance.
- longer measurement trials give more accurate estimates, which is not surprising since the measurement error is less. However Table 1 does not provide very consistent advice on how much better the accuracy will be. Estimates by averaging improve their accuracy in the ratio of the square root of the estimation time, but these estimates improve more slowly (a factor of 10 more simulation time gives only a factor of 2 or less reduction in the confidence interval width).

- It is interesting that dividing the total trial time into 100 shorter trials gave worse confidence intervals, than 10 trials each ten times as long. Possibly end effects in the simulations (start-up of the queues) account for this.

The response-time predictions of the model for populations from 1 to 20 are plotted in Figure 6, for Expt. 5 with the shortest simulations, including 95% confidence intervals. We see an increasing error band because the parameter errors are amplified at larger populations, Essentially they reflect the percentage error in the demand parameters, particularly the dominant one.

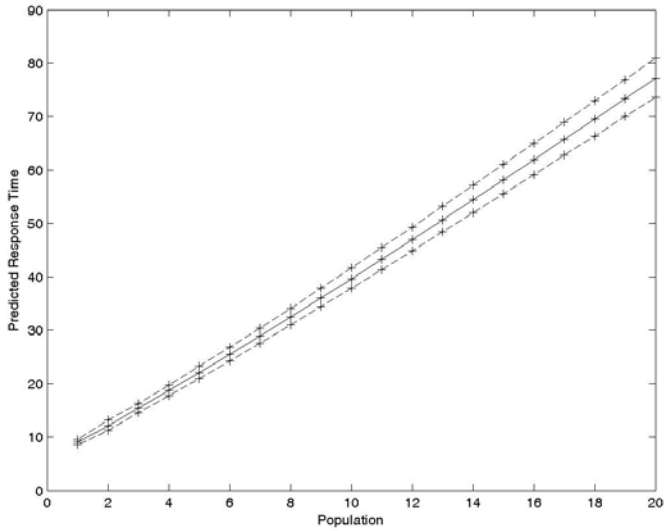


Fig 6. Output predictions of the queue model found for Case B5, with 95% confidence intervals

3.1 Structure

A recurring question is, have we captured all the structure of the system? In a classical queueing model, this means, is there another queueing resource that is not provided for in the model. Given the very straighforward structure of queueing models, one can easily add a node and fit its demand. The data used for the tests in this section was generated by a simulation of a three-queue model. If we attempt to fit a fourth queue, we can test to see if the fit is significantly better. Table 2 shows results for fitting four service demands to the same data as used in experiments B1 (long measurement runs) and B5 (short runs, giving larger measurement errors). The sums of squares criterion E(Ci) is given for these results to compare to the value E(Bi) for a three-queue model. We can see there is no reduction.

Table 2 Results for fitting models with four queues when the system has only three

Expt	$\hat{\mathbf{x}}$ (1)± CIx(1)		$\hat{\mathbf{x}}$ (2)± CIx(2)		$\hat{\mathbf{x}}$ (3)± CIx(3)		$\hat{\mathbf{x}}$ (4)± CIx(4)		E for Ci	E for Bi
C1(as B1)	2.005	0.009	2.994	0.010	4.010	0.009	0.005	0.201	2.577	2.579
C5(as B5)	1.858	0.231	2.808	0.263	4.006	0.203	0.470	4.127	1482	1481

It is easy to see that the fourth queue in this model is not useful or significant to the fit. One giveaway (that the fourth queue is not very significant) is that the confidence interval half-width is much larger than the fitted demand. When the data is very accurate, in C1, the fitted value is also near zero, but when it is less accurate as in C5, the fitted value is quite a lot more than zero. Some delay due to servers 1 and 2 is evidently being accounted for in this case by the extra ghost server.

The significance is tested using the sums of squared errors E. The test statistics are:

For C1: [(E(B1) - E(C1))/1]/[E(B1)/36] = [0.001]/[2.579/36] = 0.014
For C5: [(E(B5) - E(C5))/1]/[E(B5)/36] = [-1]/[1481/36] = -0.024

The test statistic cannot be negative in linear regression, where the model is a sum of terms each with a coefficient. However this is a nonlinear regression in which the model is not a sum of terms, and we are comparing the results of two independent approximate minimizations. Even so the model with additional parameters should give a smaller E; the fact that we obtained a larger sum of squares must be due to approximation error. The critical value at the 95% level is F(1,36,0.95) = 4.14. Case C1 is two orders of magnitude from significance, and case C2 also does not indicate significance.

4 Illustration with a layered model

Real software systems have a lot more structure to their resource use, than a classical queueing network model. The layered queueing network (LQN) formalism [5][6][12] captures the nested use of resources, including logical resources such as process threads, buffers, and locks, and does so in a formalism that represents large-scale aspects of the software architecture, such as concurrent processes.

Figure 7 shows an example LQN representing a web application. The bold rectangles represent concurrent processes (called *tasks* in LQN terminology) with the attached rectangles representing their externally invoked operations (called *entries*), and associated to oval symbols representing their host processors. Entries are labeled with their CPU demand, in suitable time units, and in the figure the entry labels show both a symbolic name prefixed by $, and a value. The name identifies a parameter which was estimated,

and the value is the value used in the simulation to generate the data. We can later judge the estimation process by how close it comes to the original value. Entries call or request service from other entries, indicated by arrows; the solid arrowheads here indicate that the caller waits for the response (blocking calls). An entry may have a second phase or delayed operation, after it sends a response to its requester, so here the entry appOp shows host demands and database requests for two phases.

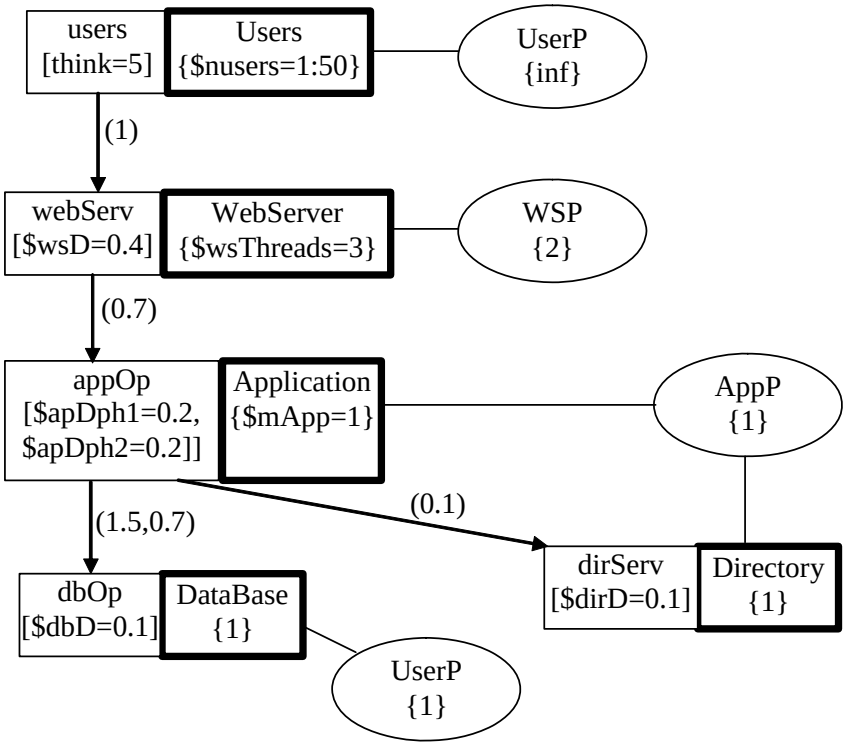


Fig 7. Layered Queueing model used for the illustration. The parameters with $name were estimated; the numbers show the values used in the simulation

The servers in an LQN include both the tasks and the processors. Processors can have a relative rate (not shown) and a multiplicity, in curly brackets (to represent multicore or symmetric multiprocessors), and tasks can also have a multiplicity (shown in curly brackets) representing the size of a finite thread pool. The Users are a special class of task, that does not serve any requests, and represent the customers to the system. An infinite processor represents one processor per task.

This model was simulated to generate data for 10 trials, using numbers of users from 5 to 50 in steps of 10. For estimation of the seven parameters, the configuration vector **u** again consisted of the number of users. Estimation was by Gauss-Newton iteration of the nonlinear regression problem, as described in theAppendix. The derivatives needed for the H matrix were computed by finite differences, after re-running the analytic solver for a slightly (1%) perturbed value of the parameter, for parameters with real values. Parameters with integer values ($wsThreads and $mApp) were perturbed by 1 unit.

Table 3 shows the results for the parameters with the half confidence interval below each result, for two experiments. In experiment D1 there were 10 trials of duration 1000 units, in experiment D2 there were 10 trials of 100000 units. Under the name of each estimated variable is given, in brackets, the value of the variable in the simulation. This is the "correct" value for the variable, and the estimate should be close to this value.

Table 3 Results for fitting seven parameters of the LQN in Fig 6 to data generated by simulation

Expt./ Trial length	$dirD	$WSD	$appDph1	$appDph2	$dbD	$mApp	$wsThreads	E
Correct value	0.1	0.4	0.2	0.2	0.1	1	3	
D1/1000	0.695	0.559	0.192	0.234	0.101	1.000	3.000	0.5935
Confidence Int. ±	0.693	0.138	0.021	0.232	0.002	0.207	0.050	
Conf. Int. as %	100%	25%	11%	100%	2%	21%	16%	
D2/100000	0.378	0.608	0.195	0.111	0.100	1.000	3.000	0.1960
Confidence Int. ±	0.367	0.059	0.011	0.202	0.001	0.140	0.024	
Conf. Int. as %	97%	10%	6%	182%	1%	14%	8%	

The results show that

- all the confidence intervals cover the correct values of the parameters. That is, the estimation technique was able to recover the parameter values, given the correct model structure. This is a basic requirement for a trustworthy estimator.
- The longer experiments produced only moderately more accurate results. In estimating a mean value, 100 times as many samples (as in experiment C2) would give a confidence interval only 1/10 as wide; here the ratios are no better than 1/3 and mostly worse than that. This suggests that more short experiments are more worthwhile.
- Curiously, one result has lower accuracy for the longer experiment, the second phase demand of the application task App. Since the second phase does not block the web server, it only affects the visible performance through congestion of App. However as App is quite busy (96% utilized, see below), the performance might be expected to be sensitive to this parameter.

- The most accurately estimated parameters were the demands for the database and application phase 1. We can see that lower-layer parameters are more accurately estimated. Since they block higher-layer tasks, they influence
- The resource utilizations show push-back between the layers, as is well-known for software bottlenecks. The utilizations with 50 users are:
    - processor utilizations:
        - WSP: 0.88 (for a dual processor, that is maximum utilization of 2.0)
        - AppP: 0.61
        - DBP: 0.34
    - task utilizations:
        - WS: 3.0 (for three threads)
        - App: 0.96
        - DB: 0.34
        - Dir: 0.04

  The App task is virtually saturated, and waiting for it introduces delays which make the WS task also saturated (the push-back). But the servers below the App task (its processor and the database server) are not heavily utilized.

  The most sensitive parameters (indicated by the percentage confidence interval widths) are the App processor and database server, which determine the holding time of the App task for one service, and thus the system throughput and delay.
- one parameter dirD is of marginal significance, based on its confidence interval. If we examine the LQN we see that the directory service is a relatively light load on processor AppP, so apparently it doesn't have enough impact to be accurately estimated. The accuracy of estimation, relative to the actual value of 0.1, is also poor.
- the estimation of integer parameters was successful. The confidence intervals are found in the conventional way, but are only of interest if the half-width is near to or greater than 1.

Experiment D2 was repeated for a test with a longer user think time (15 units instead of 5). Provided the correct think time was used as a controlled parameter in the calculations for fitting, the estimated parameters and confidence intervals were identical. The predicted performance for think times of 5 and 15 units are plotted in Figure 8 with their 95% confidence intervals.

Equally, the model can be used to extrapolate from these experimental conditions to different values of population, think time, and demands of operations for resources.

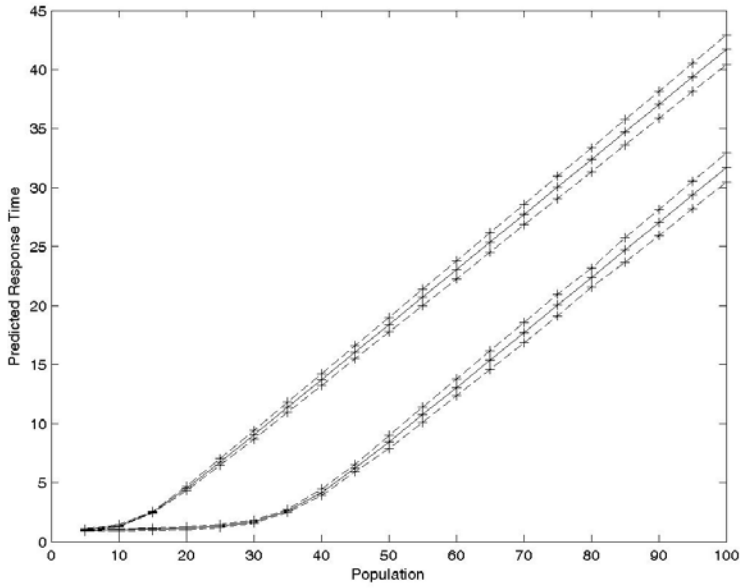


Fig. 8 Response time predictions for two values of the user think time (5 units and 15 units)

5 Exploiting the Model

A brief summary of uses of the model is (see also Figure 1):

1. Performance test design: the equation for confidence intervals (in the appendix) shows that they depend on the covariances R and sensitivities H (which come from the model). H can be affected by the configuration used to gather test data, and should be chosen so that configurations that give high sensitivities to all the desired parameters, should be combined. The duration of performance tests can also be chosen, to give adequate accuracy.
2. Performance problem diagnosis and analysis of improvements: where performance is inadequate, solutions at the architecture level can be analysed by changes to the performance model.
3. Scalability analysis: scope out the space of probably configurations to identify scalability limitations, by creating models of large configurations, built from submodels for components. This is straightforward for LQN models, where each process has a submodel. Different kinds of scaling strategy can be analysed.

4. Product configurations for individual clients can be evaluated, using a mature calibrated model of the product.
5. Product upgrade analysis: when new features are planned, estimates for their workload can be included in an existing model for a quick evaluation of their impact; this can be refined using prototypes.
6. Product performance management: some techniques for adaptive control of service configurations use a model of the application, with parameters that are updated online to track changes in system usage, efficiency or load (e.g. [8]).

5.1 Validity of the Model

The fitting process effectively validates the model for the operational conditions under which the data is gathered, and for nearby conditions within the range of the configuration parameters. In this range we may say the model is used for interpolation in the data. Of course additional validation may be done by additional measurements within that range.

Many of the uses listed above require extrapolation from the data, and here is where performance models are most useful. The advantage of fitting a performance model instead of a polynomial or some other arbitrary function, comes from its realistic underlying resource-usage semantics. So, how can we maintain confidence in the predictions of the model.

Roughly speaking there are two ways the model may fail in extrapolation. First, as we move away from the fitted range the confidence interval of predictions becomes wider, as seen in Figures 6 and 8. This is understandable and can simply be recorded. Second, some resource which was not included in the model, or which was not heavily enough utilized to be accurately modeled, may become an active constraint in a larger configuration. Consider for instance if the directory server in Figure 7 were shared among 30 replicas of the application, it would be an important limitation. Or imagine a database buffer pool which is not restricting at the scales of the test, but which becomes a constraint in larger systems. Some comments follow:

1. Many parameters which are insensitive in small-scale testing are always insensitive, and therefore their accuracy is not critical. The model itself can be used to calculate the sensitivity in large-scale versions, if it is a concern.
2. Resources which may become critical in larger deployments should be identified based on system expertise, and perhaps tests can be constructed which stress those resources.
3. Traffic statistics, both the intervals between arrivals and the distribution of demands made on applications, can challenge the assumptions of the performance modeling tools. Long-tailed distributions for arrivals and demands, and correlations between successive demands, are examples. These possibilities should not be ignored in any case, as they typically require more reserve capacity to handle them with adequate performance. If they occur they also may require a more sophisticated model. However the methods for parameter estimation would be the same, as shown here.

5.2 Simplified Models

Regression modeling usually seeks to fit the simplest model that can explain the data. One indicator of having excessive model structure and parameters is, fitted values with large confidence intervals, as shown for queueing models in Table 2. However an LQN model is simplified not by removing a single parameter, but by removing or approximating a structural unit, a task or subset of tasks. A suitable approximation for elements left out of a model is to introduce a pure delay in the holding time of an affected resource, or in the response time. This delay can be fitted, and has a similar role to fitting a constant in linear regression.

6 Conclusions

Performance models can be maintained in sync with a developing product, by calibrating their parameters from performance test data. The estimation techniques described here use standard statistical methods, and non-intrusive monitoring to obtain test data. The limitations of prediction accuracy can also be estimated.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Centres of Excellence (OCE), and by the IBM Centre for Advanced Studies, Toronto.

References

[1] S. Balsamo, A. DiMarco, P. Inverardi, and M. Simeoni, "Model-based Performance Prediction in Software Development," *IEEE Trans. on Software Eng.*, vol. 30, no. 5 pp. 295-310, May 2004.

[2] S. Barber, “Beyond performance testing”, parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/4169.html

[3] A. Bogardi-Meszoly, T. Levendovszky, H. Charaf, T. Hashimoto, “Improved Evaluation Algorithm for Performance Prediction with Error Analysis”, Proc. 11th Int. Conf. on Intelligent Engineering Systems, 2007, pp. 301-306.

[4] IBM, IBM Rational PurifyPlus, Purify, PureCoverage, and Quantify: Getting Started, May 2002. G126-5339-00.

[5] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, "Performance Analysis of Distributed Server Systems," in Proc. Sixth International Conference on Software Quality (6ICSQ), Ottawa, 1996, pp. 15-26.

[6] G. Franks, D. Petriu, M. Woodside, J. Xu, and P. Tregunno, "Layered bottlenecks and their mitigation," in Proc of 3rd Int. Conference on Quantitative Evaluation of Systems QEST'2006, Riverside, CA, Sept 2006, pp. 103-114.

[7] R. Jain, The Art of Computer Systems Performance Analysis. John Wiley & Sons Inc., 1991.
[8] M.H. Kutner, C.J. Nachtsheim, J. Neter, W. Li, *Applied Linear Statistical Models*, 5th edition, McGraw Hill, 2005
[9] M. Litoiu, T. Zheng, and M. Woodside, "Service System Resource Management Based on a Tracked Layered Performance Model," in *Proc. IEEE Int. Conf. on Autonomic Computing*, Dublin, June 2006.
[10] Y. Liu, A. Fekete, and I. Gorton, "Design-Level Performance Prediction of Component-Based Applications," IEEE Trans. on Software Engineering, vol. 31, no. 11 pp. 928-941, Nov. 2005.
[11] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, vol. 28, no. 11 pp. 37-46, Nov. 1995.
[12] J. A. Rolia and K. C. Sevcik, "The Method of Layers," IEEE Trans. on Software Engineering, vol. 21, no. 8 pp. 689-700, August 1995.
[13] P. C. Roth and B. P. Miller, "On-line Automated Performance diagnosis on Thousands of Processes," in *ACM SigPLAN Symp. on Principles and Practices of Parallel Programming (PPOPP06)*, New York, Mar. 2006.
[14] C. U. Smith and L. G. Williams, *Performance Solutions.* Addison-Wesley, 2002.
[15] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra, "Adaptive self-tuning memory in DB2," in *Proc. 32nd Int. Conf. on Very large data bases*, Seoul, 2006, pp. 1081 - 1092.
[16] A. N. Tantawi, "Method and system for dynamic performance modeling of computer application services." USA, Patent Application 20070299638, 2007.
[17] Kay White Vugrin, Laura Painton Swiler, Randall M. Roberts, Nicholas J. Stucky-Mack, and Sean P. Sullivan, "Confidence Region Estimation: Techniques for Nonlinear Regression: Three Case Studies", Sandia Laboratories Report SAND2005-6893, October 2005.
[18] M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", *Proc. WOSP'2005*, Mallorca, pp 1-12.
[19] C. M. Woodside, T. Zheng, and M. Litoiu:, "The Use of Optimal Filters to Track Parameters of Performance Models.," in *Proc. 2nd Int. Conf. on Quantitative Evaluation of Systems*, Torino, Italy, 2005, pp. 74-84.
[20] Murray Woodside, Greg Franks, Dorina C. Petriu, "The Future of Software Performance Engineering", Proc Future of Software Engineering 2007, at ICSE 2007, IEEE Computer Society Order Number P2829, May 2007, pp 171-187.
[21] WOSP, the Proceedings of the ACM International Workshop on Software and Performance, ACM Press 1998-2007

Appendix: Nonlinear Regression

Various numerical minimization techniques can be used to find $\hat{\mathbf{x}}$. A simple one which gives some insight into the nature of the problem is a simple gradient descent. We form the derivative vector $\partial E(\mathbf{x})/\partial x$:

$$\partial E(\mathbf{x})/\partial \mathbf{x} = -2 \Sigma_t \mathbf{H}^T(\mathbf{x}; \mathbf{u}_t) \mathbf{R}^{-1} \mathbf{e}_t,$$

where $\mathbf{H}(\mathbf{x}; \mathbf{u}_t) = \partial \mathbf{h}(\mathbf{x}; \mathbf{u}_t)/ \partial \mathbf{x}$, and from any starting point $\mathbf{x}$ we make a *gradient descent step* proportional to the negative gradient:

$$\mathbf{x}^{new} = \mathbf{x} + \Delta\mathbf{x} = \mathbf{x} - a\, \partial E(\mathbf{x})/ \partial \mathbf{x} = \mathbf{x} + (2a)\, \Sigma_t\, \mathbf{H}^T(\mathbf{x}; \mathbf{u}_t)\, \mathbf{R}^{-1}\, [\mathbf{z}(t) - \mathbf{h}(\mathbf{x}; \mathbf{u}_t)]$$

where a is the constant of proportionality, or step size control parameter. If this is repeated until x converges, we have *simple gradient descent*.

Various quasi-Newton methods are better. They use an approximation to the Hessian matrix E_{xx} (the matrix of second derivatives of $E(\mathbf{x})$):

$$E_{xx} = \partial^2 E(\mathbf{x})/\partial \mathbf{x}^2 = -2\, \Sigma_t\, \partial \mathbf{H}^T(\mathbf{x}; \mathbf{u}_t)/\partial \mathbf{x}\, \mathbf{R}^{-1}\mathbf{e} + 2\, \Sigma_t\, \mathbf{H}(\mathbf{x}; \mathbf{u}_t)^T \mathbf{R}^{-1} \mathbf{H}(\mathbf{x}; \mathbf{u}_t)$$

$$\approx\ 2\, \Sigma_t\, \mathbf{H}(\mathbf{x}; \mathbf{u}_t)^T\, \mathbf{R}^{-1}\, \mathbf{H}(\mathbf{x}; \mathbf{u}_t)$$

The first term is ignored because it is small when the residuals are small, for instance near the best fit (if the fit is good). Then the step size is determined by E_{xx}. For example, by differentiating a quadratic approximation for $E(\mathbf{x})$ (a Taylor expansion around the point x)), we could choose the minimum of the approximation, which is at

$$\mathbf{x}^{new} = \mathbf{x} + \Delta\mathbf{x} = \mathbf{x} - E_{xx}\, \partial E(\mathbf{x})/\partial \mathbf{x}$$

Other quasi-Newton methods can also be applied to $E(\mathbf{x})$. Some of them build a Hessian approximation from information gathered over several steps.

Successive Linear Regressions

The same iterative algorithm is obtained by considering a sequence of weighted least-squares problems based on linearizing $\mathbf{h}(\mathbf{x}; \mathbf{u})$ about the current estimates of $\mathbf{x}$. One output of this interpretation is a standard calculation for the sampling covariance of the estimates $\mathbf{x}$, which gives confidence intervals.

Ordinary regression equations can be written based on the Taylor expansion around any current estimate $\mathbf{x}$ as (referring to [8] Chapter 13):

$$\mathbf{z}_t = \mathbf{h}(\mathbf{x}; \mathbf{u}_t) + \mathbf{H}(\mathbf{x}; \mathbf{u}_t)(\Delta\mathbf{x}) + \mathbf{v}_t$$

(where $\mathbf{v}$ is still the hypothetical random error term) and finding the increment $\Delta\mathbf{x}$ to minimize the weighted sum of squares for this linear model. We put all the observations into a single partitioned vector of mT components, and similarly all the residuals $\mathbf{e}$, random errors $\mathbf{v}$, the predictions $\mathbf{h}$, and also put the sensitivities $\mathbf{H}$ into a single partitioned matrix, giving (using the notation of Kutner et al, chapter 13):

Y = vector combining the T error vectors $\mathbf{e}_t = \mathbf{z}_t - \mathbf{h}(\mathbf{x}; \mathbf{u}_t)$ at the current starting estimate **x**

D = matrix combining the **H** matrices computed at **x**

W = a matrix with T "$\mathbf{R}^{-1}$" matrices on its diagonal, one for each period

These partitioned matrices look like this:

$$Y = \begin{pmatrix} e_1 \\ \dots \\ e_t \\ \dots \\ e_T \end{pmatrix}, \qquad D = \begin{pmatrix} H(x,u_1) \\ \dots \\ H(x,u_t) \\ \dots \\ H(x,u_T) \end{pmatrix}, \qquad W = \begin{pmatrix} R^{-1} & 0 & \dots & 0 \\ 0 & R^{-1} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & R^{-1} \end{pmatrix}$$

and the model to be fitted in the neighborhood of some starting estimate **x** is

$$\mathbf{Y} = \mathbf{D}\,\Delta\mathbf{x} + \mathbf{v},$$

where the elements of **D** are treated as the observations. Now by conventional least-squares solution the value of $\Delta\mathbf{x}$ which minimizes the weighted sum of squares $(\mathbf{Y} - \mathbf{D}\,\Delta\mathbf{x})^T\mathbf{W}(\mathbf{Y} - \mathbf{D}\,\Delta\mathbf{x})$ is

$$\Delta\mathbf{x} = (\mathbf{D}^T\mathbf{W}\mathbf{D})^{-1}\,\mathbf{D}^T\mathbf{W}\mathbf{Y}$$
$$\hat{\mathbf{x}}^{\,new} = \hat{\mathbf{x}} + \Delta\mathbf{x}$$

and this is used to update $\hat{\mathbf{x}}$, and to iterate to convergence. It may be more convenient to see it in terms of sums over the matrices and vectors for each measurement period, by expanding the partitioned matrices:

$$\Delta\mathbf{x} = (\Sigma_t\,\mathbf{H}_t^T\mathbf{R}^{-1}\mathbf{H}_t)^{-1}\,\Sigma_t\mathbf{H}_t^T\mathbf{R}^{-1}\mathbf{e}_t$$

Remember that **H** and **e** are those values for the starting estimate **x** for this step, and $\mathbf{H}_t$ is different for each step because $\mathbf{u}_t$ is different.

The covariance matrix of $\Delta\mathbf{x}$ (and thus of the solution) is

$$\mathbf{P} = (mse)\,(\mathbf{D}^T\mathbf{W}\mathbf{D})^{-1} = (mse)\,(\Sigma_t\,\mathbf{H}_t^T\mathbf{R}^{-1}\mathbf{H}_t)^{-1}$$

where *mse* estimates a scale factor for the covariance, given by:

$$mse = (\mathbf{Y}^T\mathbf{W}\mathbf{Y} - \Delta\mathbf{x}^T\mathbf{D}^T\mathbf{W}\mathbf{Y})/(mT - n) = (\Sigma_t \mathbf{e}_t^T\mathbf{R}^{-1}\mathbf{e}_t - \Delta\mathbf{x}^T \Sigma_t\mathbf{H}_t^T\mathbf{R}^{-1}\mathbf{e}_t)/(mT - n)$$

which as the iteration converges and $\Delta\mathbf{x}$ goes to zero, becomes:

$$mse = \mathbf{Y}^T\mathbf{W}\mathbf{Y}/(mT - n) = (\Sigma_t \mathbf{e}_t^T\mathbf{R}^{-1}\mathbf{e}_t)/(mT - n)$$

The factor (mT − n) is the degrees of freedom remaining in the data after fitting n parameters to mT measured values.

Confidence Intervals

The converged solution $\hat{\mathbf{x}}$ of the above linear approximation can be used to give conventional confidence interval estimates for each parameter separately. Under the assumptions we have made (normality of measurement errors, and approximate linearity of h(x,u)), the posterior distribution of **x** is normal with covariance matrix **P**. Then the confidence interval at level α for the kth parameter is

$$\hat{\mathbf{x}}_k \pm t(1-\alpha/2; mT-n) (sqrt(P_{kk}))$$

where t in this equation is the t-statistic with (mT-n) degrees of freedom (measured values - fitted parameters), and P_{kk} is the estimated variance of $\Delta\mathbf{x}_k$, a diagonal element of **P**.

According to the linearization of **h**, a small deviation $\Delta\mathbf{y}$ of the predicted performance **y**, due to a small deviation $\Delta\mathbf{x}$ in **x**, is given by

$$\Delta\mathbf{y} = \mathbf{H}\,\Delta\mathbf{x}$$

Therefore the approximate distribution of the predictions **y** of the performance model, for the fitted parameters $\hat{\mathbf{x}}$ and a given **u**, is also normal with mean $\hat{\mathbf{y}} = \mathbf{h}(\hat{\mathbf{x}},\mathbf{u})$ and covariance matrix **C**:

$$\mathbf{C} = \mathrm{Cov}(\mathbf{y}) = \mathbf{H}(\hat{\mathbf{x}}, \mathbf{u})\,\mathbf{P}\,\mathbf{H}^T(\hat{\mathbf{x}},\mathbf{u})$$

From this we can derive a confidence interval for the prediction y_i as

$$\hat{\mathbf{y}}_i + t(1-\alpha/2; mT-n) (sqrt(C_{ii}))$$

When we want to state a combined uncertainty interval for a set of parameters (e.g. all of them at once) it is called a *confidence region*. Based on the normality approximation for the estimated parameters and the predicted performance values, both of these vectors have ellipsoidal confidence regions, bounded by contours of the normal distribution, which are given by

for parameters: $(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{P}^{-1} (\mathbf{x} - \hat{\mathbf{x}}) = \text{constant}$

for predictions: $(\mathbf{y} - \hat{\mathbf{y}})^T \mathbf{C}^{-1} (\mathbf{y} - \hat{\mathbf{y}}) = \text{constant}$

A simpler approach is to consider a rectangular subspace or box bounded by the intersection of the separate intervals. The "Bonferroni" approximation assumes this has a probability given by the product of the probabilities associated with each parameter separately. For instance if there are three parameters with 95% confidence intervals, the intersection of these only has probability no greater than $0.95^3 = 0.857$. (It may be less because of interaction effects between the parameter estimates.) With this approach, a conservative confidence region for n parameters at level α is given by the intersection of separate estimates at level α/n. Examples of more exact confidence regions, showing interactions between the variables, and references are given in [17].

Comparison of Structures

Two models M1 and M2 of different structures may be compared, to evaluate if one is significantly better fit than the other, using a standard F test. Suppose the values of E are E1 and E2 (E2 < E1), and the number of fitted parameters are respectively n1 and n2 (n2 > n1), then M1 is judged to be significantly better at the level α if

$$[(E2 - E1)/(n2 - n1)]/[E2/(mT - n2 - 1)] > F(n2 - n1, mT - n - 1, \alpha)$$

where F( ) is the F-statistic with degrees of freedom (n2-n1, mT - n - 1) corresponding to the degrees of freedom in the two mean squares [8]. A model with more parameters should not be able to give a larger residual error, so the fraction should always be positive.