

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0315906 1

ORNL/TM-11637

ornl

**OAK RIDGE
NATIONAL
LABORATORY**

**Modeling Histogram Data
with Piecewise Polynomials**

MARTIN MARIETTA

P. H. Worley

OAK RIDGE NATIONAL LABORATORY

CENTRAL RESEARCH LIBRARY

CIRCULATION SECTION

4500N ROOM 175

LIBRARY LOAN COPY

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this
report, send in name with report and
the library will arrange a loan.

UCN 7969 3 9-77

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161.

NTIS price codes—Printed Copy: A03 Microfiche A01

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Engineering Physics and Mathematics Division
Mathematical Sciences Section

MODELING HISTOGRAM DATA WITH PIECEWISE POLYNOMIALS

Patrick H. Worley

Oak Ridge National Laboratory
Mathematical Sciences Section
P.O. Box 2009, Bldg. 9207-A
Oak Ridge, TN 37831-8083

Date Published: August, 1990

Research was supported by the
Applied Mathematical Sciences Research Program
of the Office of Energy Research,
U.S. Department of Energy.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
operated by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC-05-84OR21400



3 4456 0315906 1

Contents

1	Introduction	1
2	Problem formulation	2
3	Algorithm formulation	5
3.1	Calculation of the partition	5
3.2	Calculation of the least-squares fit	7
3.3	Calculation of the piecewise polynomial model	10
3.4	Complexity	10
4	Examples	12
4.1	Parallel algorithm for solving a linear partial differential equation	13
4.1.1	Piecewise constant models	14
4.1.2	Piecewise linear models	14
4.1.3	Piecewise quadratic models	17
4.1.4	Summary	17
4.2	Parallel algorithm for solving a symmetric positive-definite matrix equation . .	17
4.2.1	Piecewise constant models	19
4.2.2	Piecewise linear models	20
4.2.3	Piecewise quadratic models	20
4.2.4	Summary	23
5	Generalizations	23
5.1	Heuristic for generating a mixed model	23
5.2	Examples	25
5.2.1	Differential equation example	25
5.2.2	Matrix equation example	25
6	Conclusions	28
	Acknowledgments	29
	References	29

MODELING HISTOGRAM DATA WITH PIECEWISE POLYNOMIALS

Patrick H. Worley

Abstract

As part of a research project on the performance characterization of parallel programs, piecewise polynomials are used to model histogram data that represents the processor utilization curve. In this paper an algorithm is described that generates a discontinuous piecewise polynomial model in time proportional to the amount of data.

1. Introduction

As part of a research project at Oak Ridge National Laboratory on the performance characterization of parallel programs, we are developing tools for the generation of scalable performance models. The rationale for this effort is that scalable models can be used to predict performance when problem or architecture parameters change. Similarly, a scalable model can be used to analyze the sensitivity of performance to a given parameter.

One focus of our research is on incorporating empirical performance data early in the modeling effort. We believe that the complexity of the model, and of the modeling process, can be minimized by modeling only the observed behavior. For example, the complexity of a typical scientific application code can make it difficult to use for prediction or sensitivity analysis. But much of this complexity may be independent of performance, for example, reflecting instead the complexity of the physics being simulated or simply poor coding. Thus, if the observed behavior is simple over a range of problem and architecture parameters, then a simple performance model may be sufficient, regardless of the complexity of the application code.

While observed performance is rarely simple, much of the detailed behavior is unimportant when measuring performance, and can be treated as “noise.” Also, many programs whose performance is a complex function of many parameters can be adequately modeled as a sequence of relatively simple *time-dependent models*, possibly representing a sequence of subroutine calls or other logical features of the program. Thus, identifying these *phases* of relatively simple behavior allows us to reduce the complexity of generating and using a performance model.

In this paper, we describe one of the tools we have developed for identifying phases in the *processor utilization curve*. The processor utilization curve is a histogram describing the number of processors that are computing at any given time. (If a processor is not computing, then either it is idle or it is actively involved in sending or receiving a message.) The processor utilization curve allows us to measure many of the important features in algorithm behavior: local and global speed-up and efficiency, and total execution time. Since the processor utilization curve does not explicitly take into account either interprocessor communication patterns or the logical structure of the program, it is unlikely that a phase analysis of the processor utilization curve will suffice to model the performance of a parallel program, but it is an important first step.

The goal of phase identification in the processor utilization curve is to break the curve into segments each of which can be well-approximated by a simple function. Thus, we need to identify the beginning and ending time of each phase and the underlying trend, or simple behavior, within the phase. We are currently pursuing two approaches to this problem: the first is a statistical analysis of the data to identify phase changes, and the second is a piecewise polynomial fit to the processor utilization data. In this paper, we describe one variant of the latter approach. For the rest of this paper, we will use the term *model* to refer to the piecewise polynomial fit, and not to the performance model that is the ultimate goal of this research.

Fitting piecewise polynomials to empirical data has a long history, but most standard techniques do not appear to be appropriate since our goal is to preserve abrupt changes in behavior, not smooth them away. In particular, there is no reason to require that a model be continuous across phases. The overall process is more complicated as well. Since we normally don't know how many phases there are, we generate fits assuming first that there is only one phase, then

two, then three, etc. To identify the correct number of phases requires a sensitivity analysis with respect to the number of assumed phases. A correct analysis will take into account how much the fit changes when an additional phase is assumed, and what features of the approximation are preserved when additional phases are assumed. Currently, the modeler simply picks the fits he/she likes best, using whatever heuristics seem natural at the time. While this technique for choosing the number of phases can be improved upon, the tools described here will never be more than aids to the modeler, and heuristics and user-interaction will always play a large role in the modeling process.

The simpler problem of fitting a piecewise polynomial with n pieces, or phases, to the histogram data representing the processor utilization curve is still a computationally difficult nonlinear problem. For example, if the error in the approximation is measured using the standard L_2 norm, then the problem is a nonlinear least-squares problem where both the breakpoints separating phases and the coefficients of the polynomial representing each phase must be calculated. We will refer to this as the *least-squares problem*. Previous work on the least-squares problem includes Friedman and Silverman [5], Hawkins [10],[11], and Vose [13]. Friedman and Silverman describe a linear complexity *heuristic* for solving the least-squares problem. Their heuristic adds one breakpoint at a time when generating an n -phase model, and, once a breakpoint has been placed, it is rarely moved. We have observed this to be a poor heuristic for our application since the optimal choice of breakpoints can vary wildly as a function of the number of phases. Hawkins uses a dynamic programming algorithm to solve the least-squares problem. The complexity of Hawkins' algorithm is quadratic in the amount of data, which can be too expensive for large data sets, especially for use in an interactive environment. In other work at Oak Ridge National Laboratory, Vose uses a hill-climbing solution technique to approximately solve the least-squares problem. Given a reasonable initial estimate of the breakpoints, Vose's algorithm is acceptably fast for small n , and it places no restrictions on the locations of the breakpoints. More importantly, he has proposed a new formulation of the least-squares problem that is more appropriate for our application. We refer the reader to his paper for more information.

In this paper, we develop a computationally efficient technique for fitting an n -phase piecewise polynomial to histogram data by using a special metric to define the error. This metric has no special suitability for our application other than that it is a metric, but it is intuitively no less meaningful than the L_2 norm that is traditionally used. The new technique is well-suited for computing fits for a sequence of n values, which is how the algorithm is used in practice. It is also simple to include many different types of functions in the fit with only a moderate increase in the computation time.

2. Problem formulation

Let T be a finite positive constant. Consider a nonnegative piecewise constant function h with a finite number of pieces that is defined on the interval $[0, T]$. This is our generic description of the processor utilization curve. Consider the metric space of vector-valued functions $\bar{f} = (f_1, \dots, f_n)$, where each f_i is a real-valued L_2 function on the domain $[0, T]$, with the metric

$$\|\bar{f}\|_n = \max_{i \in \{1, \dots, n\}} \left\{ \int_0^T f_i^2(x) dx \right\}.$$

Let $\bar{h} = (h, \dots, h)$, i.e. the vector of n copies of h .

Consider the set P_k of vector-valued functions $\bar{p} = (p_1, \dots, p_n)$ defined in the following way:

- 1) With each \bar{p} is associated an ordered set of $n - 1$ numbers $\{x_1, \dots, x_{n-1}\}$ in $[0, T]$ representing the endpoints of a (possibly degenerate) n -element partition of $[0, T]$. Let $x_0 = 0$ and let $x_n = T$.
- 2) Each p_i is a k th-degree polynomial in the interval $[x_{i-1}, x_i]$ and is identical to h in the rest of the interval $[0, T]$.

Thus, if $\bar{p} \in P_k$, then

$$\|\bar{p} - \bar{h}\|_n = \max_{i \in \{1, \dots, n\}} \left\{ \int_{x_{i-1}}^{x_i} (p_i(x) - h(x))^2 dx \right\}. \quad (1)$$

For any $\bar{p} \in P_k$ and $i \in \{1, \dots, n\}$, we will refer to

$$\|\bar{p} - \bar{h}\|_{n,i} = \int_{x_{i-1}}^{x_i} (p_i(x) - h(x))^2 dx$$

as the i th local error of approximating \bar{h} by \bar{p} .

By (1), the best approximation to \bar{h} by a member of P_k can be calculated by finding a piecewise polynomial of degree k and at most n phases that minimizes the maximum "piecewise" (or local) L_2 error. Since this is a metric in the space described above, the question of best approximation is well defined. Moreover, if n is large enough, then $\bar{h} \in P_k$. The advantage of using this formulation is that we can use an equidistribution property to identify a best approximation. First, we show that there exists a best approximation to \bar{h} in P_k .

Let X represent the $(n - 1)$ -dimensional cube $\prod_{i=1}^{n-1} [0, T]$. Then each $\bar{x} \in X$ represents a (possibly degenerate) n -element partition of $[0, T]$ upon ordering its components, and all n -element partitions of $[0, T]$ can be represented by some $\bar{x} \in X$. For any $\bar{x} \in X$, let $P_{k,\bar{x}}$ be the set of all functions $\bar{p} \in P_k$ associated with the partition corresponding to \bar{x} . Then there is a unique best approximation \bar{p} to \bar{h} in $P_{k,\bar{x}}$ defined by $p_i(x)$ being the (unique) best L_2 approximation to $h(x)$ by a k th-degree polynomial in the interval $[x_{i-1}, x_i]$. (The standard existence and uniqueness proofs for the best L_2 approximation to a continuous function [1],[3] also prove this result.) Call this function $\bar{p}_{\bar{x}}$, and let $p_{i,\bar{x}}$ represent its i th component. The following lemma establishes that there is a best approximation to \bar{h} in P_k .

Lemma 2.1. *If $\epsilon = \inf_{\bar{p} \in P_k} \|\bar{p} - \bar{h}\|_n$, then there exists a $\bar{q} \in P_k$ such that $\epsilon = \|\bar{q} - \bar{h}\|_n$.*

Proof. Let $\{\bar{p}_\nu\}$ represent a sequence such that

$$\epsilon = \liminf_{\nu \rightarrow \infty} \|\bar{p}_\nu - \bar{h}\|_n.$$

Since each \bar{p}_ν is associated with a given partition $\bar{x}_\nu \in X$, and since $\bar{p}_{\bar{x}_\nu}$ is at least as good an approximation to \bar{h} as is \bar{p}_ν ,

$$\epsilon \geq \liminf_{\nu \rightarrow \infty} \|\bar{p}_{\bar{x}_\nu} - \bar{h}\|_n \geq \inf_{\bar{x} \in X} \|\bar{p}_{\bar{x}} - \bar{h}\|_n. \quad (2)$$

Equality holds in (2) by the definition of ϵ . Since X is compact, this implies that the infimum is achieved for some $\bar{x}_* \in X$ and, thus, $\epsilon = \|\bar{p}_{\bar{x}_*} - \bar{h}\|_n$. \square

A similar argument proves the following lemma.

Lemma 2.2. *If \bar{q} is any best approximation to \bar{h} in P_k , then $\bar{q} = \bar{p}_{\bar{x}}$ for some $\bar{x} \in X$.*

Proof. If $\bar{q} \in P_k$, then there is an associated partition $\bar{x} \in X$. Since $\|\bar{p}_{\bar{x}} - \bar{h}\|_n \leq \|\bar{q} - \bar{h}\|_n$, and since equality holds only if $\bar{p}_{\bar{x}} = \bar{q}$, this proves the lemma. \square

The following theorem states that there exists some best approximation to \bar{h} in P_k that satisfies an equidistribution principle.

Theorem 2.3. *Let $\epsilon = \inf_{\bar{p} \in P_k} \|\bar{p} - \bar{h}\|_n$. Then there exists a best approximation \bar{q} to \bar{h} in P_k such that $\epsilon = \|\bar{q} - \bar{h}\|_{n,i}$ for all $i \in \{1, \dots, n\}$.*

Proof. First, note that the i th local error of approximating \bar{h} by a function $\bar{p}_{\bar{x}}$, $\int_{x_{i-1}}^{x_i} (p_{i,\bar{x}}(x) - h(x))^2 dx$, is a continuous function of x_i that increases (decreases) monotonically as x_i increases (decreases). This follows from $p_{i,\bar{x}}$ being the best L_2 approximation to the piecewise constant function h in $[x_{i-1}, x_i]$. In particular, if x_i is small enough, then the error is zero since h will be constant on the interval $[x_{i-1}, x_i]$ and the constant function is k th-degree polynomial. Similarly, for fixed x_i , the error increases (decreases) monotonically as x_{i-1} decreases (increases).

Next, let \bar{q} be a best approximation to \bar{h} in P_k . Let $\bar{x} \in X$ represent the partition corresponding to \bar{q} , where the components of \bar{x} are ordered. (By Lemmas 2.1 and 2.2, \bar{q} exists and $\bar{q} = \bar{p}_{\bar{x}}$.) Let $\epsilon = \|\bar{q} - \bar{h}\|_n$. Assume that

$$\epsilon > \int_{x_{i-1}}^{x_i} (q_i(x) - h(x))^2 dx$$

for some $i \in \{1, \dots, n\}$. In particular, let i_* be an index such that

$$\epsilon = \int_{x_{i_*-1}}^{x_{i_*}} (q_{i_*}(x) - h(x))^2 dx$$

and either

$$\epsilon > \int_{x_{i_*}}^{x_{i_*+1}} (q_{i_*+1}(x) - h(x))^2 dx \quad (3)$$

or

$$\epsilon > \int_{x_{i_*-2}}^{x_{i_*-1}} (q_{i_*-1}(x) - h(x))^2 dx. \quad (4)$$

If (3) holds, i.e. that the next partition to the right has a local error less than ϵ , then decrease x_{i_*} , leaving all other partition endpoints fixed and recalculating the best L_2 fits over the intervals $[x_{i_*-1}, x_{i_*}]$ and $[x_{i_*}, x_{i_*+1}]$, until either the local error in $[x_{i_*-1}, x_{i_*}]$ is less than ϵ or the local error in $[x_{i_*}, x_{i_*+1}]$ is equal to ϵ . If (4) holds, then increase x_{i_*-1} until either the local

error in $[x_{i_*-1}, x_{i_*}]$ is less than ϵ or the local error in $[x_{i_*-2}, x_{i_*-1}]$ is equal to ϵ . Repeat this process until either all local errors are the same, or until the global error $\|\bar{p}_{\bar{x}'} - \bar{h}\|_n$ for the new partition \bar{x}' is less than ϵ . One of the two conditions must occur within $n - 1$ steps of this process. If the global error is less than ϵ , then this contradicts the assumption that \bar{q} is a best approximation to \bar{h} . If all local errors are the same, then $\bar{p}_{\bar{x}'}$ has the same global error as the original \bar{q} , and itself represents a best approximation to \bar{h} in P_k . \square

The following theorem represents a converse to the previous theorem, establishing that the equidistribution principle is sufficient to characterize a best approximation. In the next section, we will use this property to generate a best approximation.

Corollary 2.4. *Let $\bar{x} \in X$ represent an n -element partition of $[0, T]$. If there exists a fixed $\epsilon \geq 0$ such that the function $\bar{p}_{\bar{x}}$ satisfies the condition $\epsilon = \|\bar{p}_{\bar{x}} - \bar{h}\|_{n,i}$ for all $i \in \{1, \dots, n\}$, then $\bar{p}_{\bar{x}}$ is a best approximation to \bar{h} in P_k .*

Proof. Assume that there exists an n -element partition of $[0, T]$, represented by \bar{x} , such that $\bar{p}_{\bar{x}}$ satisfies the equidistribution property but is not a best approximation to \bar{h} in P_k . Let $\delta = \|\bar{p}_{\bar{x}} - \bar{h}\|_n$. By Theorem 2.3 and Lemma 2.2, there does exist an n -element partition of $[0, T]$, represented by \bar{y} , such that $\bar{p}_{\bar{y}}$ is a best approximation and does satisfy the equidistribution property. Let $\epsilon = \|\bar{p}_{\bar{y}} - \bar{h}\|_n$. Therefore, $\epsilon < \delta$. In particular,

$$\|\bar{p}_{\bar{y}} - \bar{h}\|_{i,n} \leq \epsilon < \delta = \|\bar{p}_{\bar{x}} - \bar{h}\|_{i,n}$$

for all i .

Without loss of generality, assume that the components of both \bar{x} and \bar{y} are ordered. Since the local error of $\bar{p}_{\bar{x}}$ in the interval $[0, x_1]$ is strictly greater than the local error of $\bar{p}_{\bar{y}}$ in the interval $[0, y_1]$, $y_1 < x_1$. This follows from the monotonic dependence of the local error on x_1 , as described in the proof of Theorem 2.3. In consequence, the best L_2 fit to h in the interval $[y_1, x_2]$ has an error that is at least as big as the local error of $\bar{p}_{\bar{x}}$ in the interval $[x_1, x_2]$. Therefore, y_2 must be strictly less than x_2 for the local error of $\bar{p}_{\bar{y}}$ in $[y_1, y_2]$ to be less than the local error of $\bar{p}_{\bar{x}}$ in $[x_1, x_2]$. Continuing this argument, it is clear that $y_i < x_i$ for all $i \in \{1, \dots, n - 1\}$. But, this then implies that the best L_2 fit to h in the interval $[y_{n-1}, T]$ is strictly less than the best L_2 fit to h in the interval $[x_{n-1}, T]$ even though $y_{n-1} < x_{n-1}$. By the monotonic dependence of the local error on x_{n-1} , this is impossible. Thus, our assumption is incorrect, and $\bar{p}_{\bar{x}}$ must itself be a best approximation to \bar{h} in P_k . \square

3. Algorithm formulation

3.1. Calculation of the partition

The algorithm to generate a best approximation is based on the following computational kernel, where the number of phases (n), the degree of the polynomials (k), and an estimated minimum error (ϵ) have already been specified:

- a) Set $x_0 = 0$ and $i = 1$.
- b) If $i = n$ or if the best L_2 fit of a k th-degree polynomial to h in the interval $[x_{i-1}, T]$ is less than ϵ , then stop. Otherwise, find the largest x such that the best L_2 fit of a k th-degree polynomial to h in the interval $[x_{i-1}, x]$ has an error of ϵ . Set $x_i = x$.
- c) Set $i = i + 1$ and go to b).

Figure 1: Algorithm K

Call this Algorithm K (for kernel). Note that the partition generated by Algorithm K may have fewer than n elements, but it will never have more than n . By Theorem 2.4, Algorithm K will generate the partition for a best approximation if the partition has the required n elements and if the local error in the last partition element is equal to ϵ . The goal is to find the value of ϵ for which this is true. We will refer to this “optimal” value of ϵ by ϵ_* .

Consider two error tolerances δ and ϵ , where $\delta > \epsilon$. Let \bar{x} represent the partition generated by using δ in Algorithm K, and let $m(\delta)$ be the number of (nontrivial) partition elements. Let \bar{y} represent the partition generated by using ϵ in Algorithm K, and let $m(\epsilon)$ be the number of partition elements. By the same argument used to prove Theorem 2.4, it is clear that $y_i < x_i$ for all $i \leq \min\{m(\delta), m(\epsilon)\}$. Therefore, either $m(\epsilon) > m(\delta)$, or $m(\epsilon) = m(\delta)$ and the local error in the interval $[y_{m(\epsilon)-1}, T]$ is greater than or equal to the local error in the interval $[x_{m(\delta)-1}, T]$. This “monotonic behavior” of the number of partition elements and the local error in the last partition element as a function of ϵ means that we can use a bisection type algorithm on ϵ to find the best approximation.

Let n be the desired number of phases. For a given $\epsilon \geq 0$, let $m(\epsilon)$ be the number of partition elements and let $\tau(\epsilon)$ be the local error in the last partition element generated by Algorithm K. Define the objective function $f(\epsilon)$ by

$$f(\epsilon) = \begin{cases} (n - m(\epsilon) + 1) - (\tau(\epsilon)/\epsilon), & \text{if } \epsilon > 0; \\ 0, & \text{if } \epsilon = 0 \text{ and } \tau(\epsilon) = 0; \\ -n, & \text{if } \epsilon = 0 \text{ and } \tau(\epsilon) \neq 0. \end{cases} \quad (5)$$

If ϵ_* generates the partition for the best approximation to \bar{h} , then $m(\epsilon_*) = n$, $\tau(\epsilon_*) = \epsilon_*$, and $f(\epsilon_*) = 0$. If $\epsilon < \epsilon_*$, then $m(\epsilon) = n$, $\tau(\epsilon) > \epsilon$, and $f(\epsilon) < 0$. If $\epsilon > \epsilon_*$, then either $m(\epsilon) < n$ or $\tau(\epsilon)/\epsilon < 1$, and $f(\epsilon) > 0$. Thus, the sign of f differs depending on whether ϵ is too large or too small, and bisection can be used, once upper and lower bounds for ϵ_* have been calculated.

Let ϵ_l and ϵ_u represent upper and lower bounds, respectively, for ϵ_* . It is clear that zero is a lower bound, so set $\epsilon_l = 0$. For ϵ_u we use the L_2 error in the best L_2 approximation to h by a single k th-degree polynomial. Thus, $f(\epsilon_l) = -n$ and $f(\epsilon_u) = n - 1$. At this point, we can now call a bisection routine to calculate ϵ_* . In practice, we use a modified version of bisection, `zeroin` by Brent [2], to find ϵ_* , since it often converges faster than bisection. To use `zeroin`, the user must also specify the amount of error that will be tolerated when calculating ϵ_* . We will refer to this value by ξ_1 .¹ The best choice for ξ_1 balances computational cost with the

¹The tolerance actually used by `zeroin` is $(2 \cdot \text{macheps} \cdot \bar{\epsilon}_* + 0.5 \cdot \xi_1)$, where `macheps` is the relative machine precision, or machine epsilon, and $\bar{\epsilon}_*$ is the approximation to ϵ_* calculated by `zeroin`. Thus, setting $\xi_1 = 0$ will still work.

“sensitivity” of the equidistribution principle when calculating the “optimal” partition, which is problem dependent. We currently set $\xi_1 = .01$. This value is *very* conservative for our data since the minimum separation between successive times in the histogram data is normalized to be greater than or equal to one, and ϵ_* is significantly greater than one for all of our numerical experiments to date. Even for this small value of ξ_1 , the cost of the search for ϵ_* is reasonable.

3.2. Calculation of the least-squares fit

In this section, we briefly describe the algorithm used to find the best L_2 fit of a k th-degree polynomial to histogram data in a given interval $[x_{i-1}, x_i]$, and how this is used in Algorithm K. We assume that the histogram data is represented by a set of M ordered pairs $\{(z_j, \eta_j) \mid j = 0, \dots, M-1\}$ that have been ordered by their first coordinate, denoting that $h(x) = \eta_j$ for $x \in [z_j, z_{j+1}]$ for all j . Here z_0 is assumed to be 0 and z_M is defined to be T .

Consider approximating the histogram data on an interval $[x_{i-1}, x_i]$ by a polynomial² $p_i(x) = \sum_{l=0}^k \alpha_l \cdot (x - x_{i-1})^l$. Then, the square of the L_2 error in this approximation is

$$\begin{aligned} \int_{x_{i-1}}^{x_i} (p_i(x) - h(x))^2 dx &= \int_{x_{i-1}}^{z_{j_{i-1}+1}} (p_i(x) - \eta_{j_{i-1}})^2 dx \\ &+ \sum_{j=j_{i-1}+1}^{j_i-1} \int_{z_j}^{z_{j+1}} (p_i(x) - \eta_j)^2 dx \\ &+ \int_{z_{j_i}}^{x_i} (p_i(x) - \eta_{j_i})^2 dx, \end{aligned} \quad (6)$$

where z_{j_i} is defined to be the element of the set $\{z_j\}$ that is closest, but still less than, x_i . Since $p_i(x)$ is (at most) a k th-degree polynomial, the expression $(p_i(x) - \eta_j)^2$ is (at most) a $(2k)$ th-degree polynomial for each j . Thus, each integral in the right-hand side of (6) can be evaluated exactly using a Newton-Cotes quadrature formula [1] with $2k + 1$ sampling locations in the corresponding interval. Let the weights and sample locations of the quadrature formula over $[z_j, z_{j+1}]$ be $\{w_{m,j,i}\}$ and $\{\xi_{m,j,i}\}$, respectively, for $m \in \{0, \dots, 2k\}$. Similarly, let the weights and sample locations over $[x_{i-1}, z_{j_{i-1}+1}]$ be $\{w_{m,j_{i-1},i}\}$ and $\{\xi_{m,j_{i-1},i}\}$, respectively, and let the weights and sample locations over $[z_{j_i}, x_i]$ be $\{w_{m,j_i,i}\}$ and $\{\xi_{m,j_i,i}\}$, respectively. Then, (6) becomes

$$\begin{aligned} \int_{x_{i-1}}^{x_i} (p_i(x) - h(x))^2 dx &= \sum_{m=0}^{2k} w_{m,j_{i-1},i} \cdot (p_i(\xi_{m,j_{i-1},i}) - \eta_{j_{i-1}})^2 \\ &+ \sum_{j=j_{i-1}+1}^{j_i-1} \left(\sum_{m=0}^{2k} w_{m,j,i} \cdot (p_i(\xi_{m,j,i}) - \eta_j)^2 \right) \\ &+ \sum_{m=0}^{2k} w_{m,j_i,i} \cdot (p_i(\xi_{m,j_i,i}) - \eta_{j_i})^2. \end{aligned} \quad (7)$$

²In practice, we deal only with low degree polynomials, so we do not need to worry about the ill-conditioning associated with using the basis functions $\{(x - x_i)^l\}$ to represent the polynomial approximation.

Since $p_i(x)$ is a linear function of its coefficients $\{\alpha_i\}$, finding the coefficients of $p_i(x)$ that minimize (7) is a linear weighted least-squares problem *if the weights $w_{m,j}$ are all positive*. (For example, see Golub and Van Loan [9].) For $k \leq 3$, the weights will be positive if the sample locations are equidistributed in the corresponding interval [4, pp. 885-886]. Since we are only interested in using low degree polynomials in our piecewise models, equidistributed sample locations are sufficient for this application.³

The weighted linear least-squares problem can be described as finding a vector

$$\bar{\alpha} = (\alpha_0, \dots, \alpha_k)$$

that minimizes the error (in the least-squares sense) of a matrix equation $A \cdot \bar{\alpha} = \bar{b}$, where \bar{b} is a vector and A is a rectangular matrix with $k + 1$ columns. We apply Givens rotations [9] to both sides of the matrix equation to zero all elements of A except for the upper triangular matrix residing in the first $k + 1$ rows of the matrix, after which the solution $\bar{\alpha}$ can be evaluated by solving the resulting triangular matrix equation. The advantage of using Givens rotations over Householder transformations when introducing the zeroes into A is the ease by which the solution can be updated when adding new rows to A .

In Step b) of Algorithm K , x_{i-1} is fixed and we keep increasing the value of x_i until the least-squares error over $[x_{i-1}, x_i]$ is equal to the stipulated value ϵ , or until $x_i = T$. In practice, we set x_i equal to z_j for an increasing sequence of j until the squared error exceeds the squared error tolerance. Each step of this process involves adding $2k + 1$ extra rows to the bottom of A and b , zeroing these new elements in A using Givens rotations (which also alters the elements in the upper triangle of the first $k + 1$ rows in A , the first $k + 1$ elements of b , and the last $(2k + 1)$ elements of b), and calculating the new squared error. The squared error is the sum of squares of all but the first $k + 1$ elements of b , and is easily updated as each new interval is brought into the approximation.

Once the error tolerance has been exceeded, we know how to bracket the desired value of x_i by two successive z_j values, and we also know the values of j_i and η_{j_i} that correspond to this value of x_i . Trying different x_i values within this interval corresponds to changing the last $2k + 1$ rows of A and b and updating the error accordingly. The changes in the values of the last rows is a continuous function of x_i alone since z_j , and η_{j_i} are now determined, and the error is (still) a continuous monotonic function of x_i . Thus, bisection can be used to find x_i . As in §3.1, we use `zeroin`, the variant of bisection developed by Brent [2]. We again need to set an error tolerance when calling `zeroin`. We will refer to this value by ξ_2 . The best value for ξ_2 is again problem dependent. We currently use $\xi_2 = .1$, which is relatively conservative since $z_{j_i} - z_{j_i+1} \geq 1$ for our data.

To summarize, the algorithm used to implement Step b) of Algorithm K is described in Fig. 2.

³For larger k , we can sample at the zeroes of the k th-order Legendre polynomials, translated and scaled for the correct interval, in order to get positive weights [4, p. 887]. Using these sampling locations, we need only k sampling locations since we would then be using a Gaussian quadrature formula.

- i) Calculate the first $2k + 1$ rows of A and \bar{b} , corresponding to using a Newton-Cotes quadrature rule to calculate the integral

$$\int_{x_{i-1}}^{x_{i-1}+1} (p_i(x) - h(x))^2 dx.$$

Use Givens rotations to reduce the matrix A to upper triangular form. Set $\tau = 0$ and set $j = j_{i-1}$.

- ii) Set $j = j + 1$.

- iii) If $z_j = T$, then go to vii).

- iv) Add $2k + 1$ rows to the bottom of A and \bar{b} , corresponding to using a Newton-Cotes quadrature rule to calculate the integral

$$\int_{z_j}^{z_{j+1}} (p_i(x) - h(x))^2 dx.$$

Use Givens rotations to zero these new elements in A . Calculate the squared sum of the last $2k + 1$ elements of \bar{b} , and add this value to τ .

- v) If $\tau < \epsilon^2$ or $i = n$, then go to ii).

- vi) Use `zeroin` to find x_i such that $\tau = \epsilon^2$. This value is known to lie between z_j and z_{j+1} . Trying different values corresponds to modifying the last $2k + 1$ rows of A and \bar{b} to be consistent with using a Newton-Cotes quadrature rule to calculate the integral

$$\int_{z_j}^{x_i} (p_i(x) - h(x))^2 dx,$$

using Givens rotations to zero out the modifications in A , and recalculating τ .

- vii) If $i = n$ or $z_j = T$, then set $\tau = \sqrt{\tau}$ and stop.

Figure 2: Step b) of Algorithm K

3.3. Calculation of the piecewise polynomial model

The algorithm described in §3.1 and §3.2 generates the partition associated with an optimal piecewise polynomial model for a given set of histogram data, where optimality is defined in terms of minimizing the error function (1). The algorithm calculates a zero of the function $f(\epsilon)$ defined by (5) by first calculating an upper bound on the zero, and then using `zeroin` to determine the zero.

Each evaluation of f requires the execution of Algorithm K, described in Fig. 1. Figure 2 describes the logic used to implement Step b) of Algorithm K. This logic involves solving and updating solutions to a weighted linear least-squares problem and using `zeroin` to identify the endpoints of the partition generated by Algorithm K.

Evaluating $f(\epsilon)$ at its zero automatically generates both the desired partition and n triangular matrix equations whose solutions describe the coefficients of the optimal piecewise polynomial model. Thus, upon determining the zero using `zeroin`, determining the model requires only the solution of these triangular matrix equations. We will refer to the resulting algorithm as Algorithm M (for model).

3.4. Complexity

In this section, we calculate a simple upper bound on the number of “primitive” floating point operations in Algorithm M, which we will refer to as the *floating point complexity*. The set of primitive operations is made up of floating point addition, subtraction, multiplication, division, and square root. The floating point complexity represents the dominant term in the execution time of the algorithm. For brevity, we will also refer to the floating point complexity simply as the complexity.

As before, let $[0, T]$ be the interval over which the processor utilization curve is defined, and let M be the number of ordered pairs in the histogram data. Let Δ represent the maximum separation between successive ordered pairs in the histogram data,

$$\Delta = \max_{j \in \{0, \dots, M-1\}} (z_{j+1} - z_j),$$

where z_M is again defined to be T . Let Ω represent the maximum height in the histogram data,

$$\Omega = \max_{j \in \{0, \dots, M-1\}} \eta_j.$$

Let I be the L_2 norm of the data,

$$I = \left(\sum_{j=0}^{M-1} \eta_j^2 \cdot (z_{j+1} - z_j) \right)^{\frac{1}{2}}.$$

Let n be the number of phases in the piecewise polynomial to be fit to the data. Let ξ_1 be the error tolerance used in the call to `zeroin` when calculating ϵ_* . Let ξ_2 be the error tolerance used in the call to `zeroin` when calculating x_i in Algorithm K.

The basic operation in Step b) of Algorithm K is the addition of $2k + 1$ rows to the matrix

A and the vector b , and the elimination of the new entries in A using Givens rotations. We will refer to this operation as the *least-squares update*. The complexity of the calculation of the entries is linear in the number of entries, which, in this case, is $(2k+1) \cdot (k+2)$. For example, 2 floating point operations⁴ are sufficient when $k=0$, 10 floating point operations are sufficient when $k=1$, and 27 floating point operations are sufficient when $k=2$. The elimination of the $(2k+1) \cdot (k+1)$ new entries in A and the update to the squared error requires $4 \cdot k^3$ multiplications, $2 \cdot k^3$ additions, and $O(k^2)$ multiplications, divisions, additions, and square root calculations. Thus, for fixed k , the floating point complexity of the least-squares update is a constant, which we will refer to as $c(k)$. The complexity of adding and triangularizing $2k+1$ rows, as in Step i) of Fig. 2, is strictly less than this value.

Step b) of Algorithm K adds and eliminates (or triangularizes) $2k+1$ rows for every ordered pair in the histogram data, representing a complexity of no more than $M \cdot c(k)$. It also modifies and eliminates $2k+1$ rows every time x_i is modified within a call to `zeroin`. Using bisection to calculate x_i requires at most $\log_2(\Delta/\xi_2)$ modifications [1, pp. 42-44], and `zeroin` is called at most $n-1$ times during Algorithm K. Every modification of x_i in `zeroin` also incurs a small fixed computational cost not related to modifying the rows of A and \bar{b} . If we denote this “overhead” by ζ , then the complexity of Algorithm K is bounded from above by

$$M \cdot c(k) + (n-1) \cdot \log_2 \left(\frac{\Delta}{\xi_2} \right) \cdot (c(k) + \zeta). \quad (8)$$

Using bisection to calculate ϵ_* requires at most $\log_2(I/\xi_1)$ evaluations of $f(\epsilon)$. This bound on the number of evaluations comes from setting $\epsilon_u = I$ and $\epsilon_l = 0$, i.e. assuming that the zero function is the best k th-degree polynomial approximation to h . Each evaluation of $f(\epsilon)$ involves executing Algorithm K and computing 3 additions and a division. Calculating the best k th-degree polynomial approximation to h , in order to calculate ϵ_u , requires $M \cdot c(k)$ floating point operations. Thus, the complexity of calculating ϵ_* is no more than

$$\left(\log_2 \left(\frac{I}{\xi_1} \right) \right) \cdot \left(M \cdot c(k) + (n-1) \cdot \log_2 \left(\frac{\Delta}{\xi_2} \right) \cdot (c(k) + \zeta) + 3 + \zeta \right) + M \cdot c(k).$$

Once we have calculated ϵ_* , all that is left is to calculate the coefficients of the piecewise polynomial model. This requires the solution of n triangular matrix equations, at a cost of $k \cdot n^2$ floating point operations [9]. Thus, the floating point complexity of Algorithm M is bounded from above by

$$M \cdot c(k) \cdot \left(\log_2 \left(\frac{I}{\xi_1} \right) + 1 \right) + C(k, n, \Delta, \xi_1, \xi_2, \zeta), \quad (9)$$

where

$$C(k, n, \Delta, \xi_1, \xi_2, \zeta) = \left(\log_2 \left(\frac{I}{\xi_1} \right) \right) \cdot \left((n-1) \cdot \log_2 \left(\frac{\Delta}{\xi_2} \right) \cdot (c(k) + \zeta) + 3 + \zeta \right) + k \cdot n^2. \quad (10)$$

Assume that ξ_1 and ξ_2 are fixed, and that k and Ω are bounded independent of the histogram data. (The constants ξ_1 and ξ_2 are usually chosen by the modeler to be appropriate for the

⁴In all cases, the operation count includes exactly one square root calculation, with the rest of the operations being floating point additions, multiplications, or divisions.

underlying resolution of the data, and are independent of any particular data set. The constant k is never more than three in our applications because we are looking for “simple” phases. The constant Ω is bounded by the maximum number of processors in the multiprocessor used to generate the processor utilization curve.) Since $n \leq M$, $\Delta < T$, and $I \leq \Omega \cdot \sqrt{T}$, the complexity of Algorithm M is $O(M \cdot \log_2^2 T)$. This follows directly from (9) and (10). For most of our applications, $M \gg n$, Δ is bounded from above independent of T , and

$$M \cdot c(k) \cdot \left(\log_2 \left(\frac{I}{\xi_1} \right) + 1 \right) \gg C(k, n, \Delta, \xi_1, \xi_2, \zeta).$$

In this case, the second term in (9) can be ignored when estimating the complexity, and the complexity grows as $O(M \cdot \log_2 T)$. The $\log_2 T$ factor in this expression can be eliminated if a relative error bound is used for ξ_1 , but this may affect the robustness of the algorithm.

Note that the constants in (9) and (10) are very conservative, as noted by the following examples:

- The routine `zeroin` often converges faster than bisection.
- The interval $[z_j, z_{j+1}]$ can be significantly shorter than Δ , decreasing the complexity of calculating x_i in `zeroin`.
- For a given ϵ , fewer than n phases may be generated, reducing the number of “inner” calls to `zeroin`.
- Usually, the constant I is a very poor approximation to the actual value used for ϵ_u . Moreover, when solving for ϵ_* for an increasing sequence of n , ϵ_* for the previous value of n is an upper bound on ϵ_* for the current value of n . Thus, we can continually improve our value for ϵ_u , which decreases the complexity of calculating ϵ_* in `zeroin`.

4. Examples

This section describes applications of our algorithm to two examples. Both are processor utilization curves for parallel programs that were executed on the 1024-processor Ncube/3200 multiprocessor at Sandia National Laboratories in Albuquerque. The programs are written in C and use PICL [7], [8], a portable instrumented communication library, for interprocessor communication and to collect performance data. The first example represents a relatively small data set where the underlying histogram nature of the data is still visible. We show that a piecewise constant model ($k = 0$) leads to a very good model for this example, and that higher degree piecewise polynomials do not significantly improve upon the piecewise constant model. The second example represents a larger data set, with a different nature. The data appears to describe a smooth function, but with a fair amount of “noise” that may or may not be important. We show that higher degree polynomials are necessary to adequately model this data.

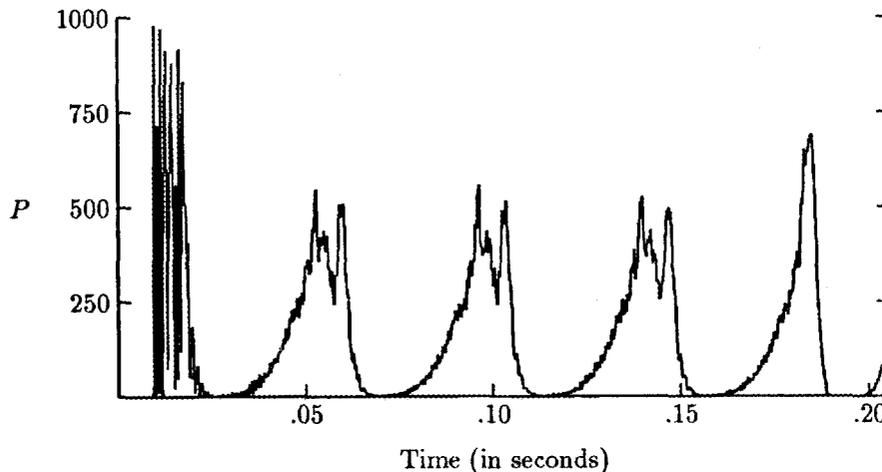


Figure 3: Processor utilization curve for differential equation example.

The piecewise polynomial models were generated on an IBM RS/6000 workstation. Performance statistics were collected, verifying that the complexity estimates made in §3.4 are valid.

Note that vertical lines are added to the graphs of all piecewise polynomial models to connect the pieces of the model. This makes the models easier to interpret.

4.1. Parallel algorithm for solving a linear partial differential equation

The first program is a parallel implementation of a large timestep algorithm for numerically solving the homogeneous wave equation in one space dimension with periodic boundary conditions and Dirichlet initial conditions:

$$\begin{aligned} \frac{\partial^2}{\partial t^2} u(x, t) - \frac{\partial^2}{\partial x^2} u(x, t) &= 0 \quad \text{for } (x, t) \in [0, 1] \times [0, 1] \\ u(0, t) &= u(1, t) \quad \text{for } t \in [0, 1] \\ u(x, 0) &= u_0(x) \quad \text{for } x \in [0, 1]. \end{aligned}$$

Values for the initial data u_0 are given on a mesh of equally spaced locations in $[0, 1]$, and a second-order approximation in space is used when advancing the solution in time. A single timestep can be used to calculate the solution at time $t = 1.00$, but, if the solution is needed at intermediate times, then multiple timesteps are calculated, with the solution at the previous timestep used as the data for calculating the next timestep. The complexity of the serial algorithm is a linear function of the number of mesh locations and of the number of timesteps.

Figure 3 contains the processor utilization curve for this program when 2048 mesh locations and 1024 processors are used to calculate the solution at times $t = .25, .50, .75, 1.00$. The histogram data is made up of 1208 ordered pairs, where the first coordinate is in microseconds. The algorithm takes $T = 205,824$ microseconds to execute, the maximum separation between consecutive data is $\Delta = 9216$ microseconds, and the average separation is approximately 200

microseconds.

The efficiency of this parallel program is very low because the amount of interprocessor communication is roughly equal to the complexity of the serial algorithm for this ratio of the number of processors to the number of mesh locations. Despite the low efficiency, this number of processors minimizes the execution time on this multiprocessor. An initialization phase and the four timesteps are easily identified in the processor utilization curve. One requirement for any reasonable model of this curve is that it duplicate this structure.

4.1.1. Piecewise constant models

We begin by modeling the processor utilization curve with piecewise constant functions. The smallest number of phases needed to accurately represent the structure of the processor utilization curve is eleven. This model is depicted in Fig. 4. Depending on the desired detail of the performance model, this type of representation may be sufficient guidance for generating a performance model. One useful feature of the piecewise constant model is the “preservation” of features in the model when more phases are calculated. This can be seen in the twenty-phase model depicted in Fig. 5. The additional detail does not mask the features that are present in the eleven-phase model.

Figure 6 describes the execution time, the optimal error tolerance, the number of objective function evaluations (in the outer call to `zeroin`), and the number of least-squares updates when calculating a sequence of models. Since a sequence was calculated, ϵ_* for the n -phase model was used as an upper bound when calculating ϵ_* for the $(n + 1)$ -phase model. This causes the cost per phase to remain relatively constant as the number of phases increases. In comparison, Fig. 7 describes the cost of calculating the 20-phase model directly, which is greater than the cost of calculating any single model in Fig. 6. Note that, for the piecewise constant model, `zeroin` was not needed to calculate \mathbf{x} ; since an analytic expression for this value exists.

4.1.2. Piecewise linear models

Next, we model the processor utilization curve with piecewise linear functions. Only six phases are required to model the structure of the curve. But, by using an eight-phase model, we are also able to capture the shape of the curve corresponding to each timestep of the algorithm, as shown in Fig. 8. This eight-phase model better represents the positions of these features of the curve than does the eleven-phase piecewise constant model. But the piecewise linear model also takes on negative values, which never occurs in the data. Moreover, while the phase information is not lost for larger numbers of phases, it can be obscured as more phases are added to the piecewise linear models.

The cost of calculating the piecewise linear models is consistent with the cost of calculating the piecewise constant models and with the model of the complexity described in §3.4. The execution time to calculate the piecewise linear models lies between 1.576 seconds and 1.966 seconds when calculating two-phase through twenty-phase models, consecutively. The number of f evaluations is between 22 and 26, and the number of least-squares updates is between 29,632 and 33,734. The execution time is fairly constant over the range of numbers of phases, when calculated consecutively, possibly increasing slightly for large numbers of phases.

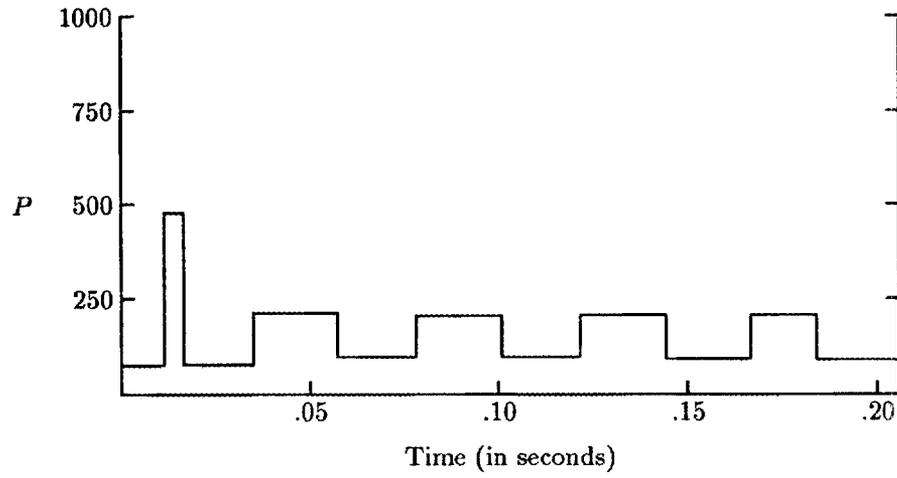


Figure 4: Eleven-phase piecewise constant model for differential equation example.

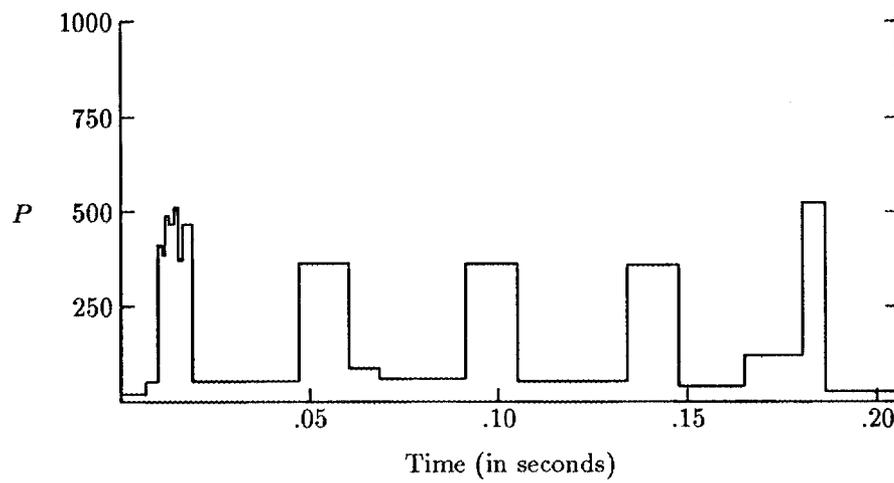


Figure 5: Twenty-phase piecewise constant model for differential equation example.

Phases	seconds	ϵ_*	f evaluations	least-squares updates
1	0.011	83109.4	1	1208
2	0.248	58761.1	25	30225
3	0.248	47913.8	25	30249
4	0.239	40414.8	24	29064
5	0.239	36784.3	24	29087
6	0.220	33807.9	22	26685
7	0.230	31050.0	23	27921
8	0.231	28064.5	23	27945
9	0.231	25772.7	23	27968
10	0.242	24066.5	24	29207
11	0.233	22054.7	23	28013
12	0.242	18030.0	24	29256
13	0.233	14960.6	23	28059
14	0.249	13369.3	23	28082
15	0.244	12163.9	24	29327
16	0.225	11891.1	22	26906
17	0.245	10664.7	23	28152
18	0.246	10359.8	23	28174
19	0.235	10156.3	22	26971
20	0.237	10116.7	22	26993

Figure 6: Performance statistics for calculating successively larger piecewise constant models for the differential equation example.

Phases	seconds	ϵ_*	f evaluations	least-squares updates
1	0.011	83109.4	1	1208
20	0.266	10116.7	26	31854

Figure 7: Performance statistics when calculating a 20-phase piecewise constant model for the differential equation example.

The increase in the execution time over that of the piecewise constant models is primarily due to the increase in $c(k)$, the cost of a single least-squares update. For example, calculating a one-phase linear model requires the same number of least-squares updates as calculating a one-phase constant model, but is six times more expensive (0.066 seconds for the one-phase linear model and .011 seconds for the one-phase constant model). The number of least-squares updates also increases somewhat since `zeroin` is required to calculate x_i .

4.1.3. Piecewise quadratic models

Finally, we model the processor utilization curve with piecewise quadratic models. Six phases are required to represent the structure of the curve, while nine-phases are sufficient to represent the shape and position of the internal features, as shown in Fig. 9. This model is not obviously better than the eight-phase linear model. The quadratic models also take on negative values, but they seem less prone to introducing detail that obscures structure as the number of phases increases than the linear models are.

The execution time to calculate the piecewise quadratic models lies between 5.881 seconds and 6.792 seconds when calculating two-phase through twenty-phase models, consecutively. The number of f evaluations is between 22 and 25, and the number of least-squares updates is between 29,307 and 32,778. The execution time is fairly constant over the range of numbers of phases, when calculated consecutively.

The increase in the execution time over that of the piecewise constant and piecewise linear models is again due to the increase in $c(k)$, the cost of a single least-squares update. The cost of calculating a one-phase quadratic model is 0.278 seconds, which is approximately four times the cost of calculating a one-phase linear model.

4.1.4. Summary

In summary, Algorithm M behaves as predicted. The models expose the underlying structure of the processor utilization curve once the correct number of phases has been identified. The piecewise constant model is the most robust in the sense that the number of phases calculated is not important to identifying the important features once the number of phases is large enough. The piecewise linear and quadratic models do bring out added detail, but they are significantly more expensive to calculate without being particularly illuminating.

4.2. Parallel algorithm for solving a symmetric positive-definite matrix equation

The second program is a parallel implementation of a numerical algorithm to solve a linear system $A \cdot \bar{x} = \bar{b}$ where A is a dense symmetric positive-definite matrix and \bar{x} and \bar{b} are vectors. The algorithm first factors A into $L \cdot L^t$, where L is a lower triangular matrix, and then solves the triangular systems

$$L \cdot \bar{y} = \bar{b} \quad \text{and} \quad L^t \cdot \bar{x} = \bar{y},$$

called the *forward solve* and *backward solve*, respectively. See Geist and Heath [6] for a description of the parallel implementation of the factorization, and see Heath and Romine [12] for a description of the parallel implementation of the forward and backward solves.

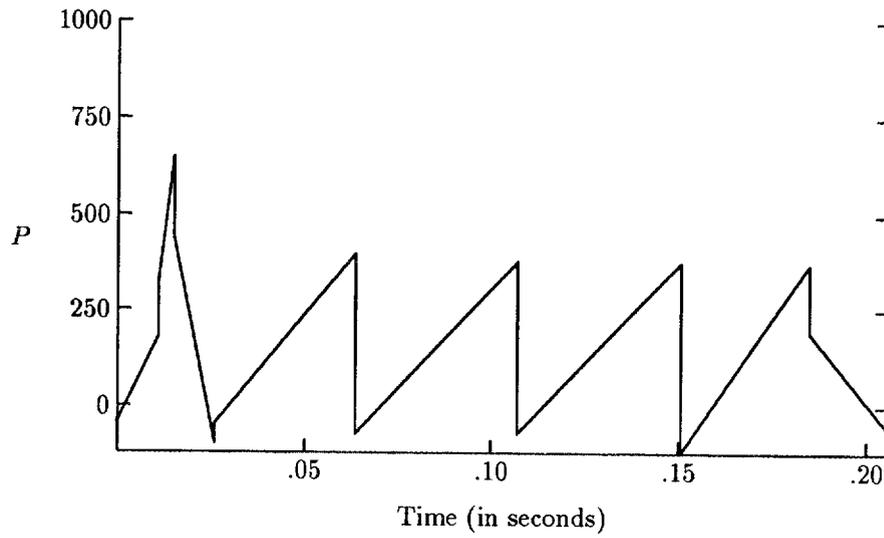


Figure 8: Eight-phase piecewise linear model for differential equation example.

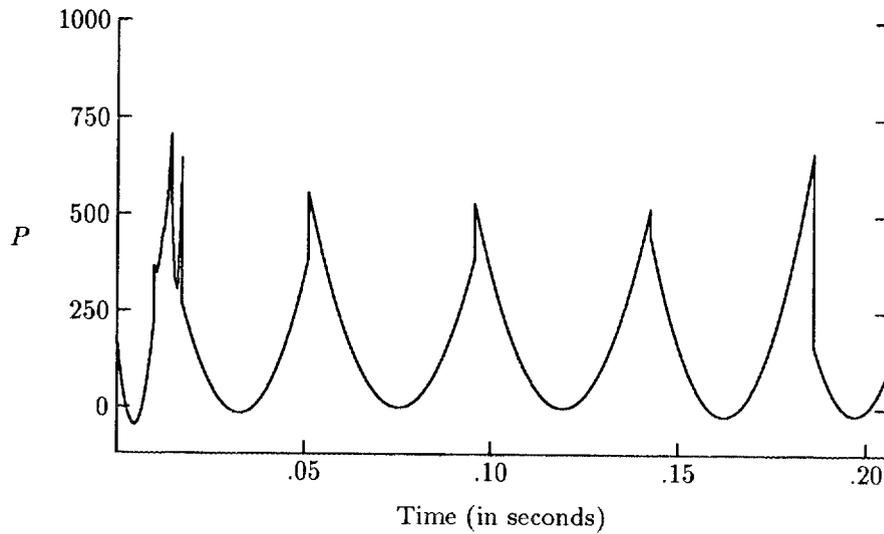


Figure 9: Nine-phase piecewise quadratic model for differential equation example.

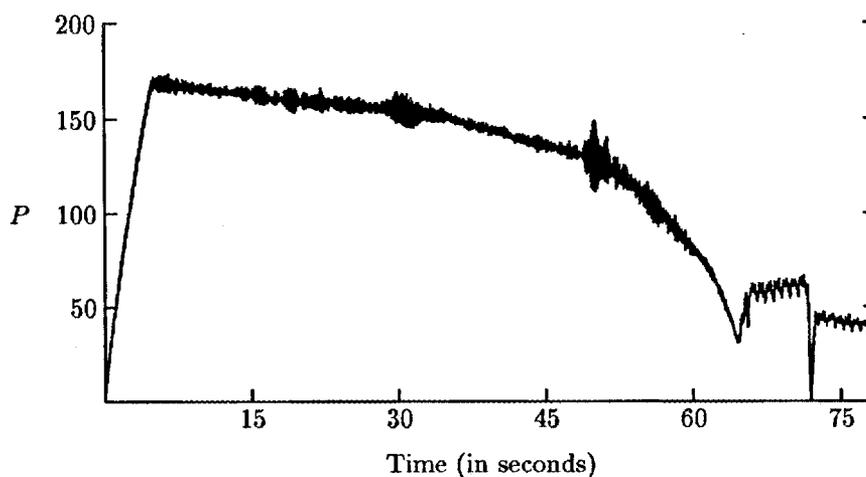


Figure 10: Processor utilization curve for matrix equation example.

Figure 10 contains the processor utilization curve for this program when 256 processors are used to solve a problem where the matrix A has 1024 rows (and columns). The histogram data is made up of 7865 ordered pairs, where the first coordinate is in microseconds. The algorithm takes $T = 78,640,281$ microseconds to execute and the maximum separation between consecutive data is $\Delta = 10,000$ microseconds, which is also the average separation.

The *apparent* underlying trend is much smoother than for the example in §4.1, but it also is contaminated by what looks like noise. Logically, there are 3 phases: the factorization phase, the forward solve phase, and the backward solve phase. The processor utilization curve also indicates a short start-up phase. From knowledge of the algorithm, we expect the part of the curve corresponding to the factorization to be quadratic, and so expect higher degree piecewise polynomial models to do better on this example.

4.2.1. Piecewise constant models

The piecewise constant approximations are not very satisfactory models for this curve. For example, Figure 11 contains the twenty-phase model. This model does a good job of approximating the curve, but it doesn't provide much insight into the behavior of the curve, with the exception that the forward and backward solves are correctly identified. To the extent that the piecewise constant curve does do a good job approximating the curve, the resulting model might be used as the input for a more expensive approximation.

The cost of calculating the piecewise constant models lies between 1.490 seconds and 1.823 seconds when calculating two-phase through twenty-phase models, consecutively, and the variation in execution time diminishes as the number of phases increases. The increase over the cost of calculating the piecewise constant models for the example in §4.1 corresponds almost exactly to the increase in the amount of histogram data. Thus, in correspondence with the complexity analysis in §3.4, the complexity varies most strongly as a linear function of the amount of histogram data when k is fixed.

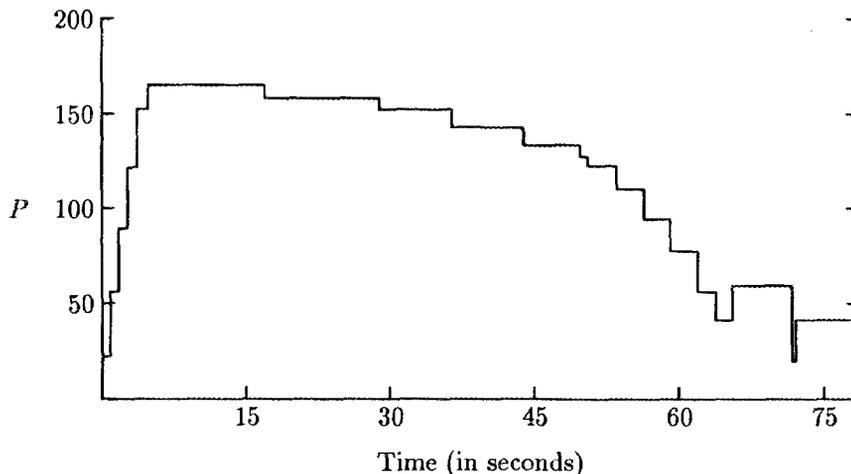


Figure 11: Twenty-phase piecewise constant model for matrix equation example.

4.2.2. Piecewise linear models

The piecewise linear models are more interesting for this processor utilization curve. The first one to capture all of the relevant details is the five-phase model depicted in Fig. 12: the start-up, factorization, forward solve, and backward solve are all identified as distinct phases. Fig. 13 contains the twenty-phase model. While this is not particularly useful when generating a performance model, it does demonstrate that the higher degree piecewise polynomial models do a good job of approximating “smooth” functions, and are a reasonable way of smoothing noisy data.

The cost of calculating the piecewise linear models lies between 10.116 seconds and 12.516 seconds when calculating two-phase through twenty-phase models, consecutively. The execution time is fairly constant over the range of numbers of phases. The relative increase over the execution time of calculating piecewise linear models in §4.1 again agrees with the relative increase in the amount of histogram data.

4.2.3. Piecewise quadratic models

We expected the piecewise quadratic model to be well suited for modeling this curve. For small numbers of phases, this is true. In particular, even the one-phase model, depicted in Fig. 14, does a reasonable job of capturing the overall behavior of the curve. The five-phase model depicted in Fig. 15 makes clear each of the 4 logical phases, as well as identifying a possible change of behavior during the factorization. It is unclear from the data whether the break is due to the large amount of “noise” at that point in the curve, or whether the behavior actually does change. It does indicate that someone generating a performance model for this algorithm should look closely at the behavior of the program at that point in time. For larger numbers of phases, the approximation to the factorization part of the curve is improved only marginally, while the approximation to the rest of the curve becomes more difficult to interpret. In particular, a piecewise quadratic model does not seem well-suited for approximating the

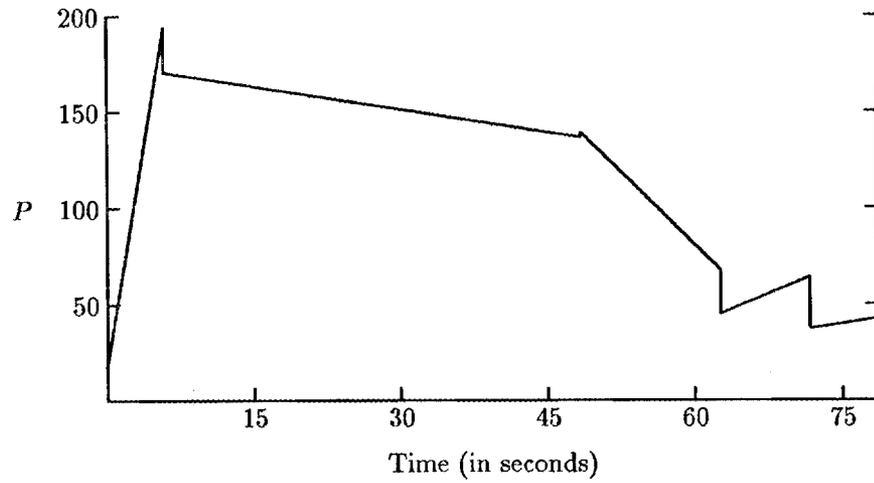


Figure 12: Five-phase piecewise linear model for matrix equation example.

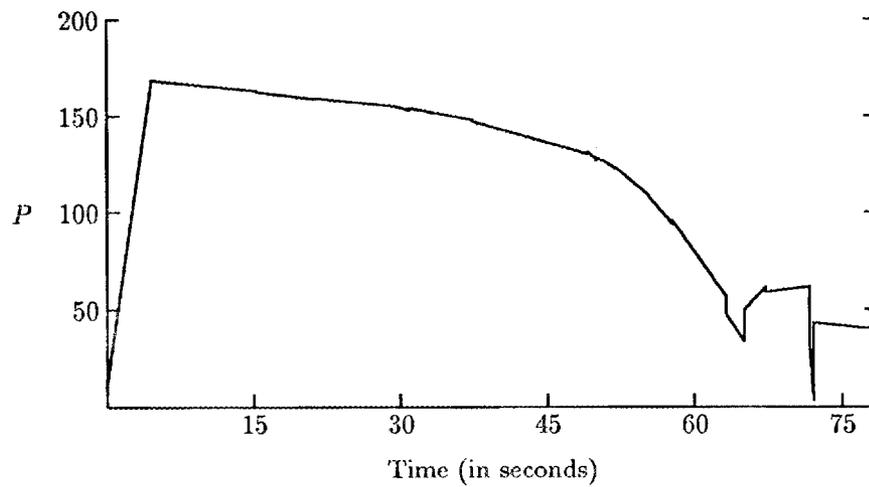


Figure 13: Twenty-phase piecewise linear model for matrix equation example.

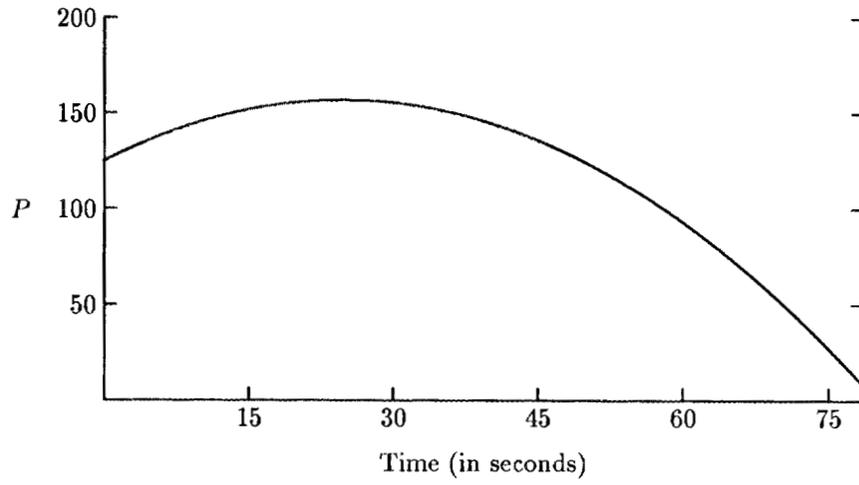


Figure 14: One-phase piecewise quadratic model for matrix equation example.

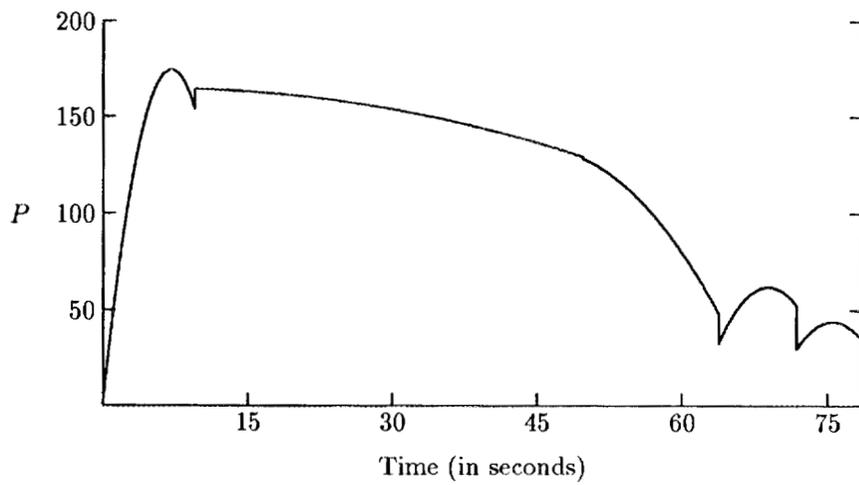


Figure 15: Five-phase piecewise quadratic model for matrix equation example.

forward and backward solves.

The cost of calculating the piecewise quadratic models lies between 35.521 seconds and 41.133 seconds when calculating two-phase through twenty-phase models, consecutively. This is relatively expensive compared to the other models for this example and for the models for the previous example, but it is consistent with the complexity being (primarily) a linear function of $M \cdot c(k)$. For example, on the average, 24 least-squares updates are required per histogram datum for all models for these two examples. A good rule-of-thumb estimate for the execution time is $24 \cdot M \cdot c(k)$, where the following values for $c(k)$ are derived from our numerical experiments.

k	$c(k)$
0	.000009 seconds
1	.000057 seconds
2	.000216 seconds

4.2.4. Summary

For the matrix equation example, the higher degree piecewise polynomial approximations generate better models, at least for small numbers of phases. The large amount of data makes the cost of the piecewise quadratic model relatively expensive, without providing significantly more information than does the piecewise linear model. The piecewise quadratic model does seem to verify that the factorization part of the curve is essentially quadratic in nature. An empirically-derived model of the execution time indicates that a good approximation to the complexity of modeling both the differential equation example and the matrix equation example is $24 \cdot M \cdot c(k)$ for each number of phases, when they are computed consecutively. This is in general agreement with the complexity analysis described in §3.4.

5. Generalizations

For the example in §4.1, the piecewise constant models seem most appropriate, while higher degree piecewise polynomials are needed to adequately model the example in §4.2. If we consider a processor utilization curve that is a concatenation of both of these, scaled in a reasonable way, none of the models using fixed k may produce satisfactory models. What we want is some way to vary k within the same model.

5.1. Heuristic for generating a mixed model

In any direct comparison between two polynomial approximations with different degrees, the polynomial with the higher degree will have the lower error when approximating a given curve. What this comparison does not take into account is that an even better approximation may result from breaking up the curve into two pieces and using a different low degree polynomial to approximate each of the two pieces. This is a reasonable approach if the resulting piecewise model is no more complex than the single high degree approximation. Thus, what we want to do is to take into account the “simplicity” of the model as well as the ability to approximate the data.

For piecewise polynomials, the obvious measure of simplicity is the number of parameters needed to define the polynomial. Thus, a piecewise constant function requires two parameters to define each piece, the starting location and the function value. A piecewise linear function requires three parameters per phase, the starting location and the parameters of the linear function, $a \cdot x + b$. Similarly, a piecewise quadratic function requires four parameters per phase. We currently incorporate the following heuristic into Step b) of Algorithm K) in order to generate a “mixed” constant/linear/quadratic piecewise polynomial model:

- Given ϵ , the current error tolerance, and x_i , the position of the beginning of the i th element of the partition, do the following to calculate x_{i+1} :
 - 1) Find the largest value x_c such that the best L_2 fit of a constant function to h in the interval $[x_i, x_c]$ has a squared error of $\epsilon^2/2$.
 - 2) Find the largest value x_l such that the best L_2 fit of a linear function to h in the interval $[x_i, x_l]$ has a squared error of $3 \cdot \epsilon^2/4$.
 - 3) Find the largest value x_q such that the best L_2 fit of a quadratic function to h in the interval $[x_i, x_q]$ has a squared error of ϵ^2 .
 - 4) If $(x_q - x_i) > 2 * (x_c - x_i)$ and $(x_q - x_i) > 4 * (x_l - x_i)/3$, then use the quadratic fit to h in the interval $[x_i, x_q]$, and set $x_{i+1} = x_q$. Go to 7).
 - 5) If $(x_l - x_i) > 3 * (x_c - x_i)/2$, then use the linear fit to h in the interval $[x_i, x_l]$, and set $x_{i+1} = x_l$. Go to 7).
 - 6) Use the constant fit to h in the interval $[x_i, x_c]$, and set $x_{i+1} = x_c$.
 - 7) Set $i = i + 1$.

Using this logic, if a two-phase piecewise constant function approximates h to within an error tolerance of ϵ over an interval $[x_i, x_{2c}]$, if a single quadratic function approximates h to within an error tolerance of ϵ over an interval $[x_i, x_q]$, and if $x_{2c} > x_q$, then the two-phase piecewise constant function is considered the better fit. The same number of parameters are required by both models, but the two-phase piecewise constant model covers a longer interval when satisfying the same error bound. A similar analysis holds when comparing constant fits to linear fits and linear fits to quadratic fits. The heuristic is more general than this comparison implies since the best two-phase fit of the data may have one constant piece and one quadratic piece. That is, constant (or linear) fits need not be “bundled” together when generating the approximation.

The heuristic as stated is not complete since the fit in the final element of the partition is not treated. If $x_q = T$, then the quadratic fit is used only if its squared error is less than half the squared error of the constant fit and less than three-fourths the squared error of the linear fit. If $x_l = T$ and the quadratic fit is not used, then the linear fit is used only if its squared error is less than two-thirds the squared error of the constant fit.

The complexity of evaluating $f(\epsilon)$ after this modification to Algorithm K is at least as great as the total complexity of evaluating $f(\epsilon/\sqrt{2})$ for $k = 0$, evaluating $f(\sqrt{3} \cdot \epsilon/2)$ for $k = 1$, and evaluating $f(\epsilon)$ for $k = 2$ in the original algorithm. It can be greater since choosing a constant

model for a given phase may require the linear and quadratic models to reprocess some of the histogram data they used when generating their own candidate models for the phase. But even in the worst case, the complexity of the modified algorithm is never more than n times as great as the sum of the three “simple” evaluations, where n is again the desired number of phases. Thus, for fixed n , this upper bound has the same form as that described in §3.4, with only the constant factors being different.

5.2. Examples

In this section we repeat the modeling of the examples described in §4.1 and §4.2 using mixed constant/linear/quadratic piecewise polynomial models.

5.2.1. Differential equation example

A seven-phase mixed model is required to capture the nature of the processor utilization curve for the differential equation example, while a nine-phase model also correctly captures information about the shape and locations of the features of the curve, as shown in Fig. 16. The nine-phase mixed model is a slight improvement over the nine-phase quadratic model in that the beginning and end of the curve are better characterized, the models for the internal features are consistent, and fewer negative values are generated. Unfortunately, the mixed models seem to be more sensitive to choosing the “correct” number of phases than are the piecewise quadratic models, in the sense of sometimes obscuring the underlying structure when increasing the number of phases. But, when the number of phases is chosen correctly, the mixed models are very illuminating. For example, the seventeen-phase model is depicted in Fig. 17. While this model “tries” very hard to resolve the detail in the beginning of the curve, and generates some very large positive and negative numbers in the process, resolving this part of the curve does not contaminate the model over the rest of the curve. In particular, the fact that simple models characterize the major features in the model strongly indicates that these features are real and should be incorporated into a performance model. Thus, the mixed model can improve the approximation over any of the “pure” models, but the identification of the correct number of phases to use in the model is crucial.

Performance statistics for calculating these models are listed in Fig. 18. The number of least-squares updates were measured for the quadratic model only, but this value will generally represent an upper bound on the number of least-squares updates in generating the linear and constant models. The number of least-squares updates is never more than 2000 more than the corresponding calculation for the “pure” piecewise quadratic model, and the execution time is never more than twice that of generating the piecewise quadratic model over the range examined. But, the cost is increasing as a function of n , unlike that of generating the piecewise quadratic model.

5.2.2. Matrix equation example

Even though the piecewise quadratic model does a good job of approximating the processor utilization curve for the matrix equation example, the mixed model provides additional insight.

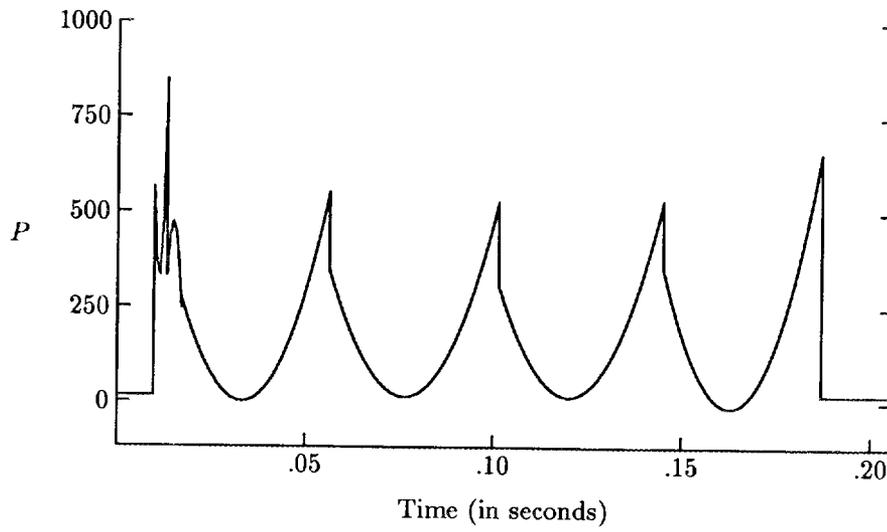


Figure 16: Nine-phase mixed constant/linear/quadratic model for differential equation example.

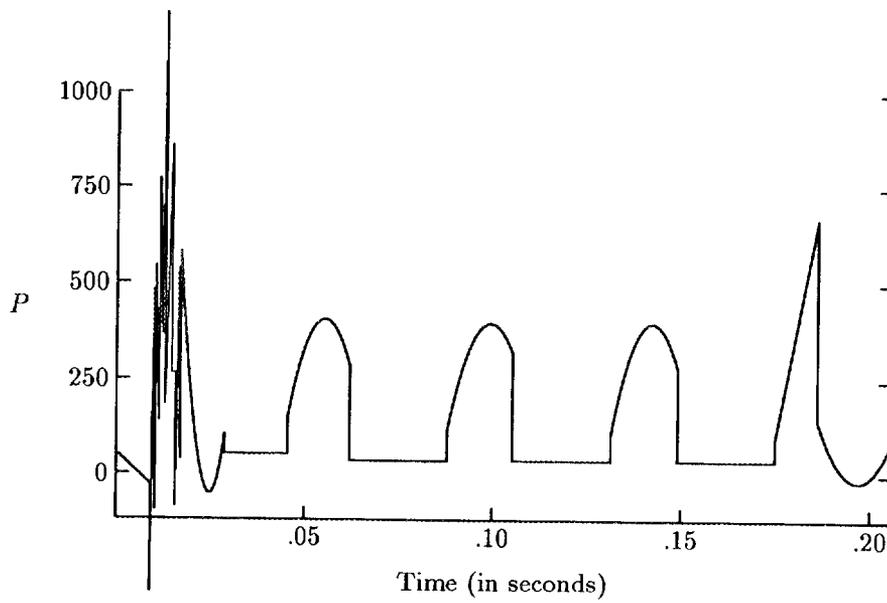


Figure 17: Seventeen-phase mixed constant/linear/quadratic model for differential equation example.

Phases	seconds	ϵ_*	f evaluations	least-squares updates
1	0.346	117534.4	1	1208
2	7.629	67833.1	26	31739
3	8.818	56146.5	26	32084
4	8.413	43531.2	25	31182
5	8.107	32303.6	24	30273
6	8.004	29091.8	24	30595
7	8.263	22109.8	24	30879
8	8.305	19661.5	24	31176
9	8.149	17450.5	24	31436
10	9.611	17450.5	23	30503
11	8.412	12599.6	22	29513
12	8.622	12475.9	23	31127
13	8.923	12405.9	23	31486
14	9.423	11369.9	23	31689
15	9.494	10153.6	23	32014
16	8.452	9442.2	22	30926
17	9.863	9096.4	22	31325
18	9.489	8824.0	22	31617
19	10.100	8362.1	23	33390
20	10.010	7965.7	22	32150

Figure 18: Performance statistics for calculating successively larger mixed constant/linear/quadratic models for the differential equation example.

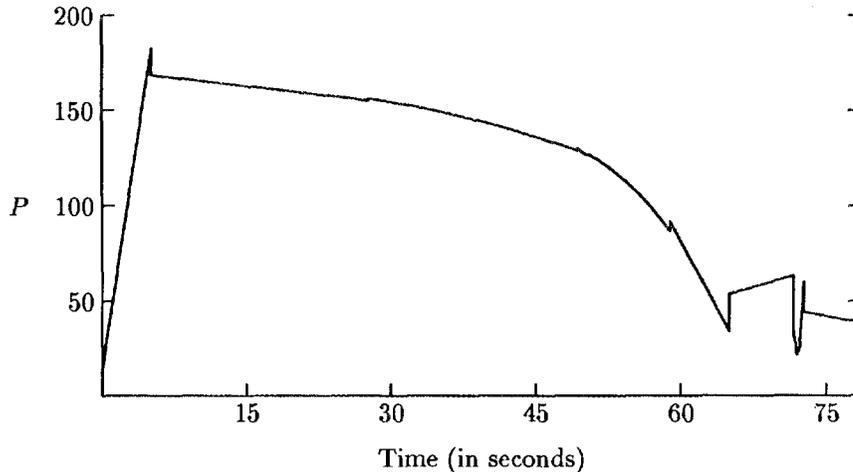


Figure 19: Nine-phase mixed constant/linear/quadratic model for matrix equation example.

For example, the nine-phase model depicted in Fig. 19 is as good as any of the other pure or mixed models. The start-up, the forward solve, and the backward solve are all well approximated by simple (linear) functions, and the factorization phase is well approximated by a small number of linear and quadratic models. Larger numbers of phases only lead to attempts to model the “noise,” and do not add additional insight into the behavior of the algorithm.

The cost of generating the mixed models for this example is somewhat less than twice that of calculating the corresponding piecewise quadratic model, between 45 seconds and 66 seconds per model. Unlike the previous example, the execution time is not an increasing function of n for this example.

6. Conclusions

We described an algorithm designed to approximate histogram data by discontinuous piecewise polynomials. The complexity of the algorithm is linear in the amount of data, and different types of approximations can be mixed in a single model with only a moderate increase in the complexity of the algorithm. The models generated by the algorithm have proven to be useful when attempting to understand parallel programs from their processor utilization curves.

In practice, we generate the models in increasing order of model complexity. This follows from our desire to identify phases with *simple* behavior. Thus, we generate piecewise constant models first. Only if these prove unsatisfactory do we, first, generate piecewise linear models, and then generate mixed constant/linear/quadratic piecewise models. If the mixed models are not too expensive to compute, they generally produce better models than do the piecewise quadratic models.

The algorithm described here can also be used to generate models based on other functions, with the same general complexity, as long as there exists a quadrature rule with positive weights that exactly determines the integral of the square of the expression “function minus constant”.

Acknowledgments

We thank George Ostrouchov and Michael Leuze for their helpful suggestions on the presentation of the material in this paper.

References

- [1] K. E. ATKINSON, *An Introduction to Numerical Analysis*, John Wiley, New York, 1978.
- [2] R. P. BRENT, *Algorithms for minimization without derivatives*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
- [3] E. W. CHENEY, *Introduction to Approximation Theory*, Chelsea Publishing Company, New York, Second Edition, 1982.
- [4] P. J. DAVIS AND I. POLONSKY, *Numerical interpolation, differentiation, and integration*, in Handbook of Mathematical Functions, M. Abramowitz and I. A. Stegun, eds., Dover Publications, New York, 1972, Ch. 25, pp. 875–924.
- [5] J. H. FRIEDMAN AND B. W. SILVERMAN, *Flexible parsimonious smoothing and additive modeling*, *Technometrics*, 31 (1989), pp. 3–21.
- [6] G. A. GEIST AND M. T. HEATH, *Matrix factorization on a hypercube multiprocessor*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Philadelphia, PA, 1987, Society for Industrial and Applied Mathematics, pp. 161–180.
- [7] G. A. GEIST, M. T. HEATH, B. W. PEYTON, AND P. H. WORLEY, *PICL: a portable instrumented communication library, C reference manual*, Tech. Rep. ORNL/TM-11130, Oak Ridge National Laboratory, Oak Ridge, TN, July 1990.
- [8] ———, *A users' guide to PICL: a portable instrumented communication library*, Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.
- [9] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The John Hopkins University Press, Baltimore, Maryland, 1983.
- [10] D. M. HAWKINS, *Point estimation of the parameters of piecewise regression models*, *Appl. Statist.*, 25 (1976), pp. 51–57.
- [11] ———, *Discussion*, *Technometrics*, 31 (1989), pp. 31–34.
- [12] M. T. HEATH AND C. H. ROMINE, *Parallel solution of triangular systems on distributed-memory multiprocessors*, *SIAM J. Sci. Statist. Comput.*, 9 (1988), pp. 558–588.
- [13] M. D. VOSE, *Piecewise linear models of processor utilization*, Tech. Rep. ORNL/TM-11566, Oak Ridge National Laboratory, Oak Ridge, TN, May 1990.

INTERNAL DISTRIBUTION

- | | |
|--------------------------|--|
| 1. B. R. Appleton | 26. C. H. Romine |
| 2. E. F. D'Azevedo | 27. T. H. Rowan |
| 3. J. J. Dongarra | 28-32. R. C. Ward |
| 4. J. B. Drake | 33-37. P. H. Worley |
| 5. T. H. Dunigan | 38. J. J. Dorning (EPMD Advisory Committee) |
| 6. G. A. Geist | 39. R. M. Haralick (EPMD Advisory Committee) |
| 7-8. R. F. Harbison | 40. J. E. Leiss (EPMD Advisory Committee) |
| 9. M. T. Heath | 41. N. Moray (EPMD Advisory Committee) |
| 10. E. R. Jessup | 42. M. F. Wheeler (EPMD Advisory Committee) |
| 11. M. R. Leuze | 43. Central Research Library |
| 12-16. F. C. Maienschein | 44. ORNL Patent Office |
| 17. E. G. Ng | 45. K-25 Plant Library |
| 18. C. E. Oliver | 46. Y-12 Technical Library |
| 19. G. Ostrouchov | /Document Reference Station |
| 20. B. W. Peyton | 47. Laboratory Records - RC |
| 21-25. S. A. Raby | 48-49. Laboratory Records Department |

EXTERNAL DISTRIBUTION

50. Dr. Loyce M. Adams, Department of Applied Mathematics, University of Washington, Seattle, WA 98195
51. Dr. Christopher R. Anderson, Department of Mathematics, University of California, Los Angeles, CA 90024
52. Dr. Robert G. Babb, Department of Computer Science and Engineering, Oregon Graduate Center, 19600 N.W. Walker Road, Beaverton, OR 97006
53. Dr. David H. Bailey, NASA Ames, Mail Stop 258-5, NASA Ames Research Center, Moffet Field, CA 94035
54. Dr. Jesse L. Barlow, Department of Computer Science, Pennsylvania State University, University Park, PA 16802
55. Dr. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185
56. Dr. Robert E. Benner, Parallel Processing Division, 1413, Sandia National Laboratories, Albuquerque, NM 87185
57. Dr. Marsha J. Berger, Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, NY 10012

58. Prof. Ake Bjorck, Department of Mathematics, Linkoping University, Linkoping 58183, Sweden
59. Dr. John H. Bolstad, L-16, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
60. Dr. James C. Browne, Department of Computer Sciences, University of Texas, Austin, TX 78712
61. Dr. Bill L. Buzbee, Scientific Computing Division, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307
62. Dr. John Cavallini, Acting Director, Scientific Computing Staff, Applied Mathematical Sciences, Office of Energy Research, U.S. Department of Energy, Washington, DC 20545
63. Dr. Tony Chan, Department of Mathematics, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90024
64. Dr. Jagdish Chandra, Army Research Office, P. O. Box 12211, Research Triangle Park, North Carolina 27709
65. Dr. Melvyn Ciment, National Science Foundation, 1800 G Street, NW, Washington, DC 20550
66. Prof. Tom Coleman, Department of Computer Science, Cornell University, Ithaca, NY 14853
67. Dr. Paul Concus, Mathematics and Computing, Lawrence Berkeley Laboratory, Berkeley, CA 94720
68. Dr. Jane K. Cullum, IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598
69. Dr. George Cybenko, Center for Supercomputing Research & Development, 104 South Wright Street, Urbana, IL 61801-2932
70. Ms. Helen Davis, Computer Science Department, Stanford University, Stanford, CA 94305
71. Professor Larry Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235
72. Dr. Iain Duff, CSS Division, Harwell Laboratory, Didcot, Oxon OX11 0RA, ENGLAND
73. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
74. Professor Geoffrey C. Fox, Physics Department, MS 356-48, California Institute of Technology, Pasadena, CA 91125
75. Dr. Chris Fraley, Department of Mathematics and Statistics, Utah State University, Logan, UT 84322-3900

76. Dr. Paul O. Frederickson, RIACS, NASA Ames Research Center, Moffet Field, CA 94035
77. Dr. Robert E. Funderlic, Department of Computer Science, North Carolina State University, Raleigh, NC 27650
78. Professor Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401
79. Dr. W. Morven Gentleman, Division of Electrical Engineering, National Research Council, Building M-50, Room 344, Montreal Road, Ottawa, Ontario, Canada K1A 0R8
80. Dr. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
81. Dr. Gene Golub, Computer Science Department, Stanford University, Stanford, CA 94305
82. Dr. Joseph F. Grcar, Division 8331, Sandia National Laboratories, Livermore, CA 94550
83. Dr. William D. Gropp, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
84. Dr. Eric Grosse, 2C 471, 600 Mountain Avenue, Murray Hill, NJ 07922
85. Prof. John L. Gustafson, Ames Laboratory, 236 Wilhelm Hall, Iowa State University, Ames, IA 50011-3020
86. Dr. Gerald W. Hedstrom, L-71, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
87. Dr. Don E. Heller, Physics and Computer Science Department, Shell Development Co., P. O. Box 481, Houston, TX 77001
88. Dr. John L. Hennessy, CIS 208, Stanford University, Stanford, CA 94305
89. Dr. N. J. Higham, Department of Mathematics, University of Manchester, Gtr Manchester, M13 9PL, ENGLAND
90. Dr. Charles J. Holland, Air Force Office of Scientific Research, Building 410, Bolling Air Force Base, Washington, DC 20332
91. Dr. Robert E. Huddleston, Computation Department, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
92. Dr. Lennart S. Johnsson, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
93. Dr. Harry Jordan, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309
94. Dr. Bo Kagstrom, Institute of Information Processing, University of Umea, 5-901 87 Umea, Sweden

95. Professor Malvyn Kalos, Courant Institute for Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012
96. Dr. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439
97. Dr. Alan H. Karp, IBM Scientific Center, 1530 Page Mill Road, Palo Alto, CA 94304
98. Dr. Kenneth Kennedy, Department of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77001
99. Dr. Tom Kitchens, ER-7, Applied Mathematical Sciences, Scientific Computing Staff, Office of Energy Research, Office G-437 Germantown, Washington, DC 20545
100. Prof. Michael Langston, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
101. Dr. Richard Lau, Office of Naval Research, 1030 E. Green Street, Pasadena, CA 91101
102. Dr. Robert L. Launer, Army Research Office, P. O. Box 12211, Research Triangle Park, North Carolina 27709
103. Prof. Tom Leighton, Lab for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139
104. Dr. Randall J. LeVeque, Department of Mathematics, University of Washington, Seattle, WA 98195
105. Dr. John G. Lewis, Boeing Computer Services, P. O. Box 24346, M/S 7L-21, Seattle, WA 98124-0346
106. Dr. Heather M. Liddell, Director, Center for Parallel Computing, Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, ENGLAND
107. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3
108. Dr. Franklin Luk, Electrical Engineering Department, Cornell University, Ithaca, NY 14853
109. Dr. Thomas A. Manteuffel, Computing Division, Los Alamos National Laboratory, Los Alamos, NM 87545
110. Dr. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550
111. Dr. Paul C. Messina, California Institute of Technology, Mail Code 158-79, Pasadena, CA 91125
112. Dr. Cleve B. Moler, MathWorks, 325 Linfield Place, Menlo Park, CA 94025

113. Dr. William A. Mulder, Koninklijke Shell Exploratie en Produktie Laboratorium, Postbus 60, 2280 AB Rijswijk, THE NETHERLANDS
114. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742
115. Dr. Joseph Olinger, Computer Science Department, Stanford University, Stanford, CA 94305
116. Professor James M. Ortega, Department of Applied Mathematics, Thornton Hall, University of Virginia, Charlottesville, VA 22901
117. Prof. Merrell Patrick, Department of Computer Science, Duke University, Durham, NC 27706
118. Dr. Linda R. Petzold, L-316, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94550
119. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650
120. Dr. David A. Poplawski, Department of Computer Science, Michigan Technological University, Houghton, MI 49931
121. Professor Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801
122. Dr. John K. Reid, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon OX11 0RA, ENGLAND
123. Dr. John R. Rice, Computer Science Department, Purdue University, West Lafayette, IN 47907
124. Dr. Garry Rodrigue, Numerical Mathematics Group, Lawrence Livermore National Laboratory, Livermore, CA 94550
125. Dr. Donald J. Rose, Department of Computer Science, Duke University, Durham, NC 27706
126. Dr. Ahmed H. Sameh, Center for Supercomputing R&D, 1384 W. Springfield Avenue, University of Illinois, Urbana, IL 61801
127. Dr. Jorge Sanz, IBM Almaden Research Center, Department K53/802, 650 Harry Road, San Jose, CA 95120
128. Prof. Robert B. Schnabel, Department of Computer Science, University of Colorado at Boulder, ECOT 7-7 Engineering Center, Campus Box 430, Boulder, Colorado 80309-0430
129. Dr. Robert Schreiber, RIACS, MS 230-5, NASA Ames Research Center, Moffet Field, CA 94035

130. Dr. Martin H. Schultz, Department of Computer Science, Yale University, P. O. Box 2158 Yale Station, New Haven, CT 06520
131. Dr. David S. Scott, Intel Scientific Computers, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006
132. The Secretary, Department of Computer Science and Statistics, The University of Rhode Island, Kingston, RI 02881
133. Prof. Charles L. Seitz, Department of Computer Science, California Institute of Technology, Pasadena, CA 91125
134. Dr. Horst D. Simon, Mail Stop 258-5, NASA Ames Research Center, Moffett Field, CA 94035
135. Dr. William C. Skamarock, 3973 Escuela Court, Boulder, CO 80301
136. Dr. Burton Smith, Teracomputer Company, 400 North 34th Street, Suite 300, Seattle, WA 98103
137. Dr. Marc Snir, IBM T.J. Watson Research Center, Department 420/36-241, P. O. Box 218, Yorktown Heights, NY 10598
138. Prof. Larry Snyder, Department of Computer Science, FR-35, University of Washington, Seattle, WA 98195
139. Dr. Danny C. Sorensen, Department of Mathematical Sciences, Rice University, P. O. Box 1892, Houston, TX 77251
140. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742
141. Mr. Steven Suhr, Computer Science Department, Stanford University, Stanford, CA 94305
142. Dr. Wei Pai Tang, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
143. Dr. Lloyd N. Trefethen, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139
144. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853
145. Dr. Robert G. Voigt, ICASE, MS 132-C, NASA Langley Research Center, Hampton, VA 23665
146. Dr. Michael D. Vose, 107 Ayres Hall, Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301
147. Dr. A. J. Wathen, School of Mathematics, University Walk, Bristol BS8 1TW, ENGLAND

148. Dr. Andrew B. White, Los Alamos National Laboratory, P. O. Box 1663, MS-265, Los Alamos, NM 87545
149. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600
- 150-159. Office of Scientific & Technical Information, P. O. Box 62, Oak Ridge, TN 37831