OPTIMIZATION OF AUTOMOTIVE VALVE TRAIN COMPONENTS WITH IMPLICT FILTERING *

T. D. CHOI†, O. J. ELSINGER†, C. T. KELLEY†, J. W. DAVID‡, AND M. ETHERIDGE‡

Abstract. In this paper we show how the implicit filtering algorithm can be parallelized and applied to problems in parameter identification and optimization in automotive valve train design. We extend our previous work by using a more refined model of the valve train and exploiting parallelism in a new way. We apply the parameter identification results to obtain optimal profiles for camshaft lobes.

Key words. Noisy Optimization, Implicit Filtering, Mechanical Systems, Automotive Valve Trains

AMS subject classifications. 65K05, 65K10, 65L05, 65Y05

1. Introduciton. In this paper we report on a parallel implementation of the implicit filtering [17], [19] algorithm and its application to problems in parameter identification and optimization in automotive valve train design. We extend our previous work [11], [10] on parameter identification by using a more refined model of the valve train and exploiting parallelism in a new way. We then apply the parameter identification results to obtain optimal profiles for camshaft lobes.

We begin in § 2 by reviewing implicit filtering and discussing the those aspects of the algorithm that were used for the first time in this application. In § 3 we present the details of the valve train model and the optimization problems to be solved. The parallel computing issues are discussed in § 4 and some representative results from [12] are presented in § 5.

2. Implicit Filtering. In this section we give a brief description of implicit filtering. We used the IFFCO code as described in [16]. We will state a convergence result from [4] and [19] and describe some, but not all, of the implementation details. We refer the reader to [16], [27], [17], and [19] for more details and statements and proofs of several convergence results.

2.1. General Description. Implicit filtering is an algorithm for solving bound constrained minimization problems

$$\min_{x\in\Omega} f(x)$$

where

$$\Omega = \{x \in R^N \mid L_i \le (x)_i \le U_i\}. \tag{2.1}$$

In (2.1) we denote the ith component of a vector x by $(x)_i$ to distinguish between the ith element of an iteration x_i.

Our paradigm for the objective function is that

$$f(x) = f_s(x) + \phi(x). \tag{2.2}$$

*Version of December 9, 1998.

† North Carolina State University, Center for Research in Scientific Computation and Department of Mathematics, Box 8205, Raleigh, N. C. 27695-8205, USA (tdchoi@unity.ncsu.edu, ojesling@eos.ncsu.edu, Tim_Kelley@ncsu.edu,). The research of these authors was supported by National Science Foundation grant #DMS-9700569 and a U. S. Department of Education GAANN fellowship. Computing activity was partially supported by an allocation from the North Carolina Supercomputing Center.

‡ North Carolina State University, Department of Mechanical and Aerospace Engineering, Box 7910, Raleigh, N. C. 27695-7910, USA (david@eos.ncsu.edu).

Here f_s is smooth and ϕ is a small-amplitude noise term. Implicit filtering is designed to avoid the local minima caused by oscillations or discontinuities in ϕ. Implicit filtering is designed to solve problems in which the amplitude of the noise ϕ decreases near the optimal point.

Implicit filtering is a difference-gradient implementation of the gradient projection algorithm [2] in which the difference increment is reduced in size as the iteration progresses. By using a sequence of difference increments, called *scales*, one hopes to jump over local minima and "implicitly" filter the noise. The algorithm was originally described and applied to problems in electrical engineering in [29], [28], [30], and [27].

In order to fully describe the algorithm as used in this paper, we must set notation by first describing the gradient projection algorithm from [2]. Let $\mathcal{P}$ denote the projection onto Ω, which is defined, for $x \in R^N$, by

$$\mathcal{P}(x)_i = \begin{cases} L_i & \text{if } (x)_i \le L_i \\ (x)_i & \text{if } L_i < (x)_i < U_i \\ U_i & \text{if } (x)_i \ge U_i \end{cases} \tag{2.3}$$

A gradient projection step transforms a current approximation x_c to a local minimizer x^* to a new approximation x_+ by

$$x_+ = \mathcal{P}(x_c - \lambda \nabla f(x_c)) \tag{2.4}$$

where the line search parameter $\lambda \in (0, 1]$ is selected using a backtracking Armijo, [1], rule, to force the sufficient decrease condition,

$$f(x(\lambda)) - f(x) \le \alpha \nabla f(x)^T (x - x(\lambda)) \tag{2.5}$$

to hold. In (2.5), α is a small parameter (usually 10^{-4}) and

$$x(\lambda) = \mathcal{P}(x - \lambda \nabla f(x)).$$

Algorithm `gradproj` is a description of an implementation of the gradient projection algorithm. The inputs are an initial iterate x, the objective f, and a limit on the number of iterations. On exit x is the approximate solution. We terminate the iteration when

$$\|x - x(1)\| \le \epsilon, \tag{2.6}$$

for some small ϵ.

ALGORITHM 2.1. `gradproj`$(x, f, nmax)$

1. *For* $n = 1, \ldots, nmax$
   - *(a) Compute* f *and* ∇f*; test for termination.*
   - *(b) Find the least integer* m *such that* (2.5) *holds for* $\lambda = \beta^m$.
   - *(c)* $x = x(\lambda)$.
2. *If* $n = nmax$ *and the termination test has failed, signal failure.*

Under standard nondegeneracy assumptions, [2], [19], one can show that the gradient projection iterates $\{x_n\}$ satisfy

$$\lim_{n \to \infty} \mathcal{P}(x_n - \nabla f(x_n)) = 0.$$

Therefore, every limit point of the sequence satisfies the first order necessary conditions.

Let $\nabla_h f$ be the difference gradient with an increment of h. We will consider both one sided differences

$$(\nabla_h f(x))_i = \frac{f(x \pm he_i) - f(x)}{\pm h}, \tag{2.7}$$

and centered differences

$$(\nabla_h f(x))_i = \frac{f(x + he_i) - f(x - he_i)}{2h}. \tag{2.8}$$

In (2.7) and (2.8) e_i is the unit vector in the ith coordinate direction. Based on both previous experience [27], [19] and testing for this specific problem, we use centered differences whenever possible (*i. e.* when both $x \pm he_i$ are feasible) and one sided differences when only one of $x \pm he_i$ is feasible. The centered difference implementation of implicit filtering has consistently given overall better performance. To guarantee that at least one of $x \pm he_i$ is feasible we scale the variables so that the feasible set is the unit cube in N dimensions and use a maximum h of $1/2$.

To obtain the finite difference gradient projection algorithm we replace the gradient ∇f with $\nabla_h f$ and change the sufficient decrease condition to

$$f(x_h(\lambda)) - f(x) \leq \alpha \nabla_h f(x)^T (x_h(\lambda) - x), \tag{2.9}$$

where

$$x_h(\lambda) = \mathcal{P}(x - \lambda \nabla_h f(x)).$$

Truncation error in the finite difference means that the line search in step 1b of Algorithm `gradproj` could fail because $-\nabla_h f$ might not be a descent direction. As described in [17], this implies that the resolution of a difference increment of h has been exhausted and the iteration should terminate. A test for this must be a part of any practical implementation. The difference error is also reflected in the termination criterion for the optimization since we can not expect to drive $\|x - x(1)\|$ below the truncation error in the difference, even for a smooth problem. In the code that we used for this work, we terminate the difference gradient projection method when

$$\|x - x(1)\| \leq \tau h, \tag{2.10}$$

where τ is a parameter.

With this in mind, a finite difference gradient projection method can be simply described. The new inputs are the difference increment h and a limit $mmax$ on the number of times the step will be reduced in the line search.

ALGORITHM 2.2. `fdgradproj`$(x, f, nmax, h, mmax)$

1. *For* $n = 1, \ldots, nmax$
   - (a) *Compute* f *and* $\nabla_h f$*; test for termination.*
   - (b) *Find the least integer* $m \leq mmax$ *such that* (2.9) *holds for* $\lambda = \beta^m$*. If no such* m *exists, terminate.*
   - (c) $x = x_h(\lambda)$.
2. *If* $n = nmax$ *and the termination test is failed, signal failure.*

Implicit filtering simply calls `fdgradproj` repeatedly for a sequence of difference increments $\{h_k\}_{k=0}^K$, called scales. Algorithm `imfil1` is a straightforward version.

ALGORITHM 2.3. `imfil1`$(x, f, nmax, \{h_j\}, mmax)$

1. *For* $k = 0, \ldots, K$
   Call `fdgradproj`$(x, f, nmax, h_k, mmax)$

2.2. Quasi-Newton Formulation. The performance of implicit filtering is significantly improved by using a quasi-Newton model of the reduced Hessian. To explain how this is done we must introduce some notation. We define the active set $\mathcal{A}(x)$ and inactive set $\mathcal{I}(x)$ of indices at a point $x \in \Omega$ by

$$\mathcal{A} = \mathcal{A}(x) = \{i \,|\, (x)_i = U_i \text{ and } (\nabla_h f(x))_i < 0 \text{ or } (x)_i = L_i \text{ and } (\nabla_h f(x))_i > 0\}$$

and let $\mathcal{I} = \mathcal{I}(x)$ be the complement of $\mathcal{A}$. We will often omit the explicit dependence on x.

We define projections $\mathcal{P}_\mathcal{A}$ and $\mathcal{P}_\mathcal{I}$ by

$$(\mathcal{P}_\mathcal{A} x)_i = \begin{cases} (x)_i & \text{if } i \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases}$$

and

$$(\mathcal{P}_\mathcal{I} x)_i = \begin{cases} (x)_i & \text{if } i \in \mathcal{I} \\ 0 & \text{otherwise} \end{cases}$$

Note that $\mathcal{P}_\mathcal{I} + \mathcal{P}_\mathcal{A} = I$.

The reduced Hessian at x is

$$\nabla^2_R f(x) = \mathcal{P}_\mathcal{A} + \mathcal{P}_\mathcal{I} \nabla^2 f(x) \mathcal{P}_\mathcal{I}.$$

For smooth problems, it is known, [3], [19], that if one is near to a local minimizer x^* that satisfies certain nondegeneracy conditions then the projected Newton iteration

$$x_+ = \mathcal{P}(x_c - (\nabla^2_R f(x_c))^{-1} \nabla f(x_c))$$

can be modified to converge quadratically to x^* by, for example, replacing $\mathcal{A}(x_n)$ by

$$\mathcal{A}_n = \{i \,|\, U_i - (x_n)_i \le \epsilon_n \text{ or } (x_n)_i - L_i \le \epsilon_n\}, \tag{2.11}$$

and letting $\epsilon_n \to 0$ in the appropriate way. The role of ϵ is to keep the iteration from stagnating near the solution; the level of noise in our application is so large that $\epsilon = 0$ can be used.

For noisy problems with expensive function evaluations, difference Hessians not only are too expensive to compute, but also have very large errors. For this reason we use a projected quasi-Newton iteration. If we let $\mathcal{A}_c = \mathcal{A}(x_c)$ and $\mathcal{I}_c = \mathcal{I}(x_c)$ the new iteration is found by a line search using

$$x(\lambda) = \mathcal{P}(x_c - \lambda(\mathcal{P}_{\mathcal{A}_c} + \mathcal{P}_{\mathcal{I}_c} B_c \mathcal{P}_{\mathcal{I}_c})^{-1} \nabla_h f(x_c)) \tag{2.12}$$

and requiring that (2.9) hold. After x_+, $\mathcal{A}_+$, and $\mathcal{I}_+$ have been determined, we then update the matrix B_c to get B_+ and continue the iteration.

Clearly from (2.12) we need only update

$$A = \mathcal{P}_\mathcal{I} B \mathcal{P}_\mathcal{I}$$

to completely determine the iteration. If the active set has not changed, *i. e.* $\mathcal{A}_c = \mathcal{A}_+$, then we can apply any quasi-Newton update for unconstrained minimization. We tested two updates for this application, the SR1 [5], [13] and BFGS [6], [14], [18], [26] updates. The SR1 update performed better on the bound constrained problems considered in [27] while the BFGS update was better on

some simple unconstrained examples from [19]. In the context of this application, the SR1 update performed better in our testing and we confine our discussion to that update.

The implementation of SR1 in IFFCO is one of several possibilities (see [19] for details). If $\mathcal{A}_c = \mathcal{A}_+$ then we set

$$y = \mathcal{P}_{\mathcal{I}}(\nabla f(x_+) - \nabla f(x_c))$$

and $s = \mathcal{P}_{\mathcal{I}}(x_+ - x_c)$. The SR1 update is

$$A_+ = A_c + \frac{(y - A_c s)(y - A_c s)^T}{(y - A_c s)^T s}.$$

The SR1 update is skiped if $(y - A_c s)^T s = 0$ and the reduced Hessian is reset to the identity after a scale change.

If $\mathcal{A}_c \neq \mathcal{A}_+$ then IFFCO attempts to perform an update on that part of Hessian corresponding to the intersection of $\mathcal{I}_+$ and $\mathcal{I}_c$. To do this IFFCO sets

$$y = \mathcal{P}_{\mathcal{I}_+} \nabla f(x_+) - \mathcal{P}_{\mathcal{I}_c} \nabla f(x_c),$$

$$B_c = \mathcal{P}_{\mathcal{A}_c} + A_c,$$

and uses the SR1 update,

$$A_+ = \mathcal{P}_{\mathcal{I}_+} \left(B_c + \frac{(y - B_c s)(y - B_c s)^T}{(y - B_c s)^T s} \right) \mathcal{P}_{\mathcal{I}_+}. \tag{2.13}$$

One then verifies the resulting approximation to $\nabla_R f$

$$B_+ = \mathcal{P}_{\mathcal{A}_+} + A_+$$

is positive definite, reseting the model reduced Hessian to the identity if positivity fails.

For a given scale h our bound constrained quasi-Newton algorithm is

ALGORITHM 2.4. `projquasi`$(x, f, nmax, h, mmax)$

1. *For* $n = 1, \ldots, nmax$
   - (a) *Compute* f *and* $\nabla_h f$*; test for termination.*
   - (b) *Find the least integer* $m \leq mmax$ *such that* (2.9) *holds for* $\lambda = \beta^m$ *and* $x(\lambda)$ *given by* (2.12)*. If no such* m *exists, terminate.*
   - (c) $x = x_h(\lambda)$.
   - (d) *Update* A.
2. *If* $n = nmax$ *and the termination test is failed, signal failure.*

Algorithm `imfilq` is a description of the quasi-Newton form of implicit filtering that uses `projquasi`.

ALGORITHM 2.5. `imfilq`$(x, f, nmax, \{h_j\}, mmax)$

1. *For* $k = 0, \ldots, K$
   Call `projquasi`$(x, f, nmax, h_k, mmax)$

2.3. A Convergence Result. In this section we state a convergence result from [19], which is representative of the more complex results from [17] and [19]. The purpose of this is to illustrate the gap between the problems for which theory is available and those problems for which the algorithm has been observed to work well.

There is not at present a convergence theory for form of implicit filtering used in this paper. The benficial effects of quasi-Newton methods, while consistently observed in practice, have not been explained in theory and the results that address bound constraints from [17] and [15] are not simple to state. Hence we consider an unconstrained problem (*i. e.* $\mathcal{P}$ is the identity matrix) using a finite difference form of steepest descent (*i. e.* the model Hessian is the identity matrix). The restriction to an unconstrained problem is not severe because once the active set has been identified the algorithm is the same as that for an unconstrained problem for the remaining variables.

We assume that ϕ, the noise in f, is everywhere defined and bounded in order to make the statement of the results simpler. As the implicit filtering iteration progresses we measure the size of ϕ by

$$\|\phi\|_{x,h} = \max_{z=x\pm he_k} |\phi(z)|.$$

So, for unconstrained problems, $\|\phi\|_{x,h}$ is the maximum of the perturbation on the difference stencil.

In the unconstrained case the sufficient decrease condition is the difference form of the classical Armijo [1] rule

$$f(x - \lambda \nabla_h f(x)) - f(x) < -\frac{\alpha}{\lambda}\|\nabla_h f(x)\|^2 = \alpha \nabla_h f(x)^T (x(\lambda) - x) \tag{2.14}$$

THEOREM 2.1. *Let f satisfy* (2.2) *and let ∇f_s be Lipschitz continuous. Let $h_k \to 0$, $\{x_k\}$ be the implicit filtering sequence. Assume that* (2.14) *holds (*i. e. *there is no line search failure) for all but finitely many k. Then if*

$$\lim_{k\to\infty} (h_k + h_k^{-1}\|\phi\|_{x_k,h_k}) = 0 \tag{2.15}$$

then any limit point of the sequence $\{x_k\}$ is a critical point of f_s.

2.4. Algorithmic Issues. The problems solved in this paper all used implicit filtering with the SR1 quasi-Newton method. We also followed the procedure, first reported in [11], of using as the new iterate the the best value from the difference stencil if that value was lower than the iterate found by implicit filtering and terminating the iteration when a target value of f was reached.

Restarting `imfil` after termination has the theoretical benefit [17], [19] that one can characterize the final output if the algorithm is restarted until the output of each call to `fdsteep` or `projquasi` is unchanged. In the actual computations reported in this paper, the algorithm was repeatedly restarted to accomodate the laboratory and computing environment, but the formal restarting protocol from [17] was not used.

In our current implementation of implicit filtering [16] the scale is reduced if either (2.10) holds or if too many reductions in the step length are taken. The former case is successful completion of the iteration, the latter is a signal that the difference gradient with the current scale is a poor indicator of descent.

3. Valve Train Optimization Problems.

3.1. Engineering Setting. The model is based on a pushrod valve train of the type still commonly found in production and in racing organizations such as NASCAR and NHRA. With a model that accurately simulates the dynamic behavior of the valve train, we can optimize component properties using computers instead of traditional experimental testing.

The purpose of the valve train is to open and close both the intake and exhaust valves. The intake valve allows air-fuel mixture into the chamber for combustion before compression and, the exhaust valve allows spent gases to leave the combustion chamber after the power stroke. A typical pushrod valve train contains the following components: cam lobe, lifter, pushrod, rocker arm, valve and valve springs. As the camshaft rotates, the cam lobe imparts a translation motion to the follower and pushrod. The pushrod then pivots the rocker arm which in turn opens the valve. The valve springs provide the restoring force to drive the valve closed after maximum lift is obtained and, they also keep the components in contact with each other so that the motion of the mechanism reflects the motion imparted by the cam lobe.

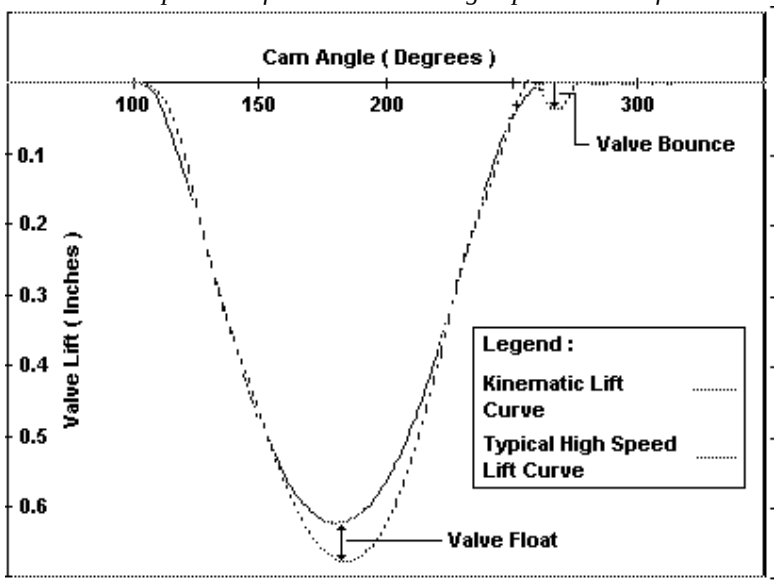
However, in high speed internal combustion engines the motion of the valve train can deviate from the ideal kinematic motion due to the inertia of the components and surge of the valve springs. These phenomenon lead to valve float and valve bounce which are detrimental to engine performance because the valve remains open during the compression stage of the combustion cycle, thus allowing leakage of the air-fuel mixture out of the combustion chamber. Valve float occurs when the inertial force of the valve train components exceeds the spring force of the valve springs thus allowing components to separate. In addition, valve float causes the valve to exceed the maximum lift of the kinematic motion and close with an abnormally high velocity as illustrated in Figure 3.1. Valve bounce occurs when the valve closes against the seat with a sufficiently high velocity that it physically bounces off the seat and remains open as the piston begins the compression cycle as illustrated in Figure 3.1. At a certain engine speed termed limit speed, the valve bounce amplitude increases dramatically thereby resulting in what appears to be chaotic valve motion and loss of engine performance. Since measurements of valve float and valve bounce are good indicators of the performance characteristics of a valve train at high engine speeds, reduction in valve bounce and/or valve float is established as the primary goal of valve train design.

3.2. Valve Train Model. The valve train model consists of a combination of a lumped mass model of the cam, cam-lifter, pushrod, rocker arm and valve with a Finite Element Analysis of the valve springs [12] [20] [21] [22] [31] as shown in Figure 3.2. The cam shaft, rocker arm center pivot shaft, and valve stem are assumed rigid. Coulomb friction acts at the rocker arm center pivot, valve stem and along each valve spring due to the presence of a friction damper within the valve spring nest. The nonlinear effects of component contact and separation are included by setting corresponding damping and stiffness coefficients to zero during the separation events. By summing forces on masses, m_1 and m_3 , while summing moments on the rocker arm equivalent inertia, I_2, we can form the equations of motion of the valve train excluding the valve springs and seat:

$$m_1\ddot{y_1} + (c_1 + c_2 + c_5)\dot{y_1} + ac_2\dot{\theta_2} + (k_1 + k_2 + k_5)y_1 + ak_2\theta_2 = k_1 d(t) + c_1\dot{d}(t) \tag{3.1}$$

$$I_2\ddot{\theta_2} + ac_2\dot{y_1} + (a^2c_2 + b^2c_3)\dot{\theta_2} - bc_3\dot{y_3} + ak_2y_1 +$$

FIG. 3.1. *Comparison of Kinematic and High Speed Valve Lift Curves.*



$$(a^2k_2 + b^2k_3)\theta_2 - bk_3y_3 = \pm\eta_{fric} - bk_3\nu_{lash} \tag{3.2}$$

$$m_3\ddot{y_3} - bc_3\dot{\theta_2} + c_3\dot{y_3} - bk_3\theta_2 + (k_3 + k_\nu)y_3 = f_{spring} \pm f_{fric} + k_3\nu_{lash} \tag{3.3}$$

Using helical rod elements we can write the valve springs' equations of motion in the form:

$$[M]\left\{\ddot{\vec{z}}\right\} + \zeta\,[K]\left\{\dot{\vec{z}}\right\} + [K]\,\{\vec{z}\} = \{F\} \tag{3.4}$$

where $[M]$ and $[K]$ are the total lumped mass and stiffness matrices, respectively. We assume that the boundary conditions for each valve spring are fixed at the base and top node, except for vertical displacements of the top node. We also assume that the damping in the valve springs is a combination of both viscous and Coulomb damping and that the viscous damping is proportional to the stiffness matrix. The external force vector $\{F\}$ consists of the valve spring reaction force at the top node and the Coulomb friction force along the valve springs. The retainer, keepers and valve stem are assumed rigid, hence the vertical displacement of the valve stem is equal to the vertical displacement of each of the top nodal points in each valve spring.

The valve head which we assume has constant thickness is modeled as an equivalent stiffness determined from a thin-plate analysis [8] [9]. We assume that the boundary conditions for the support conditions along the contact edge with the valve seat is simply supported, and therefore the equivalent stiffness is given by,

$$k_\nu = \frac{1.8132E_\nu h^3}{r_\nu^2}. \tag{3.5}$$

If the valve is in contact with the valve seat then (3.5) is applied to determine the valve head stiffness, otherwise the valve is assumed rigid and the stiffness is set equal to zero. The valve seat is modeled with a linear spring and damper. There is no lumped mass present at the valve seat surface, hence the equation of motion from summing forces at the seat is first order:

$$c_4 \dot{y}_4 = k_\nu (y_3 - y_4) - k_4 y_4 . \tag{3.6}$$

The equations of motion for the entire valve train are derived by combining (3.1)-(3.4) and (3.6) by eliminating the valve spring reaction force from (3.3) and (3.4). The results can be written in the form:

$$\left\{\ddot{\tilde{z}}\right\} = \left[\tilde{M}\right]^{-1} \left(\left\{\tilde{F}\right\} - \left[\tilde{C}\right]\left\{\dot{\tilde{z}}\right\} - \left[\tilde{K}\right]\left\{\dot{\tilde{z}}\right\}\right). \tag{3.7}$$

Hence, if the cam lift $d(t)$ and velocity $\dot{d}(t)$ profiles are known then (3.7) can be integrated to determine the displacement of the valve over a steady state cycle. Steady state response is determined by integrating over multiple cam cycles starting with zero initial conditions until the following condition is satisfied :

$$\sum_{i=1}^{360} \left(\left(y_3^n(i) - y_3^{n-1}(i) \right)^2 + \left(y_3^{n-1}(i) - y_3^{n-2}(i) \right)^2 \right) < 10^{-4} \qquad \text{for } n > 4 \tag{3.8}$$

for the nth cam cycle or until the 20th cam cycle is reached. i is the cam's rotation angle in degrees and $y_3^n(i)$ is the simulated valve lift during the nth cam cycle at cam angle i. The integrator used to integrate (3.7) does not normally return y_3 at integer values of i. Therefore, we use linear interpolation between the last time step before i and the first time step after i in order to approximate y_3 at i. The integration routine involves integrating the accelerations of each degree of freedom to obtain the corresponding velocities and positions. The number of equations can vary from approximately 50 to 200 depending upon the number of valve springs in the system, as well as, the number of finite elements required to model the valve springs.

The Runge-Kutta Fehlberg 5-6 routine [24], [7] is employed to integrate the equations of motion. We selected a one step algorithm so changes in the differential equation (when components come in and out of contact) are easy to deal with. When component separation occurs stiffness and damping coefficients change values dramatically, requiring a large decrease in stepsize which is easily handled with a variable step algorithm. A stiff integrator, such as LSODE [25], requires restarts when the equations change and was much less efficient in our testing than the explicit method.

3.3. Optimization Problems.

3.3.1. Parameter Identification. Before we can optimize the valve train, certain system parameters must be determined. The finite element technique could be used to determine the various stiffness coefficients, however the damping and friction properties are much more difficult to measure. Hence, we use the implicit filtering algorithm to determine the various system parameters involving both experimental and theoretical methods. The experimental apparatus that was used in this investigation was constructed from a small block Chevrolet engine (see Figure 3.3) fitted with race type cylinder heads and a valve train assembly for a single valve. The crankshaft was replaced with a straight 'dummy' crankshaft (i.e. no crank arms) since piston motion is not of interest in this analysis. The block was then mounted to a rigid frame which housed a DC motor. The dummy crank is driven by the DC motor which powers the valve train. As the valve train is actuated by the DC motor, the valve lift is measured by a proximeter. The valve train is operated at a number of speeds, typically from 7000 rpm to limit speed, and the data is recorded on a personal computer.

FIG. 3.2. *Lumped Mass Model.*

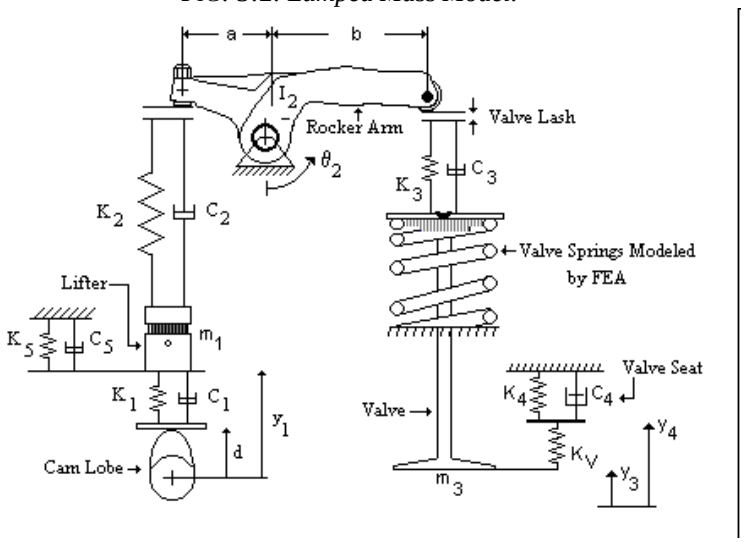
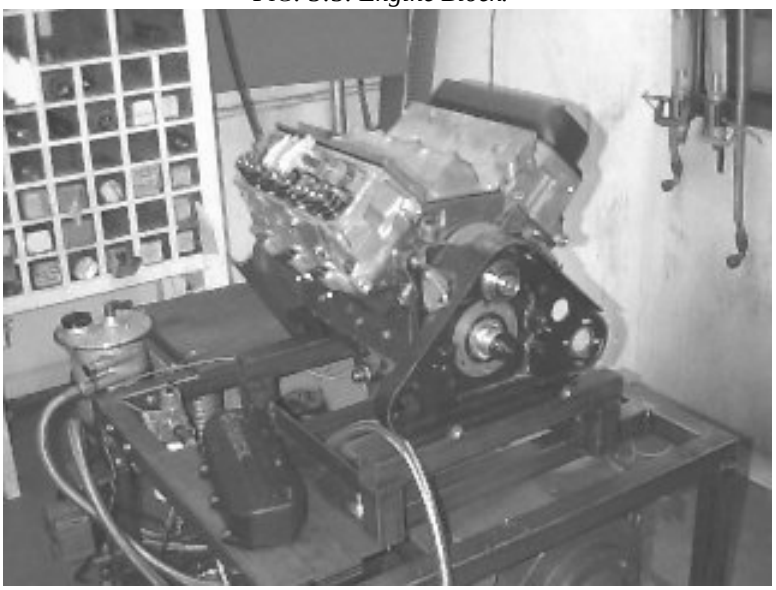


FIG. 3.3. *Engine Block.*



The experimental data is truncated due to the limited sensing range of the proximeter. This does not limit the validity of the results as the dynamic properties of the valve train are realized when the valve closes and strikes the seat.

Now that we have the experimental valve lift curves, we can determine the system parameters by matching the simulated and experimental curves over the engine speed range mentioned above. A weighted least squares fit is used to specify the cost function :

$$cf = \sum_{nspeed} \left(\sum_{i=1}^{360} w\left(y_3^{exp}(i)\right) \left(y_3^{sim}(i) - y_3^{exp}(i)\right)^2 \right) \tag{3.9}$$

where $nspeed$ represents the number of engine speeds, and i is the cam's rotation angle in degrees.

#of variables	Variable description
4	contact stiffness coefficients(k_1, k_2, k_3, k_4)
4	contact damping coefficients(c_1, c_2, c_3, c_4)
1	rocker arm friction torque
1	coulomb friction force at valve springs
1	damping coefficient in valve spring nest
1	average valve head thickness
2	number of active coils of each valve spring

TABLE 3.1
System variables

$y_3^{sim}(i)$ is the simulated valve lift at cam angle i, and $y_3^{exp}(i)$ is the experimental valve lift. As before, the valve lift at cam angle i is not readily available and is therefore obtained by linear interpolation.

Hence, as the curves tend to overlap, the cost function approaches zero. The weighting function is normally equal to $.1$ except for data points that are out of the range of the proximeter in which case the weighting function is equal to zero. During the closing event of the valve, the weighting function is set equal to 1 because of the high accuracy required in predicting the valve bounce phenomenon. Therefore, the goal of system identification is to minimize the cost function thus yielding the system parameters. Listed in Table 3.1 are the system parameters to be identified.

After the system parameters are uniquely determined, the valve train properties can be optimized for improved performance and higher obtainable limit speed. In general, any or all the components can be analyzed for optimal design, however for this analysis, only the cam profile and valve springs' geometric properties are considered.

3.3.2. Cam Profile Design. The cam profile which defines the shape of the cam lobe is usually designed by determining the cam lift curve. The cam lift is the vertical distance measured from the base circle of the cam lobe to the surface of the lobe. Usually a single function is sufficient to describe the cam lift curve. However, with high performance engines there are many kinematic constraints which must be met and with a single function there is no easy way to specify these constraints. Therefore, the cam profile is broken up into 10 cam events each of which is described by a polynomial with boundary conditions that are constructed so that the kinematic constraints are met. This technique for designing cam profiles was developed by Park and David in [23]. A set of 15 parameters are chosen to completely describe the cam profile based on these polynomial functions, as well as, the maximum lift, velocity, and cam duration. Since the maximum lift and velocity, and the cam duration directly influence other properties of the engine, these parameters are held fixed after being measured from the original cam. An additional 5 parameters are selected for design from the valve springs' geometric properties. These properties are the number of active coils and wire diameter of each spring along with the stack height of the valve spring nest. This brings the total number of search parameters to 20. The goal for valve train design is to reduce both valve float and valve bounce. Hence, the optimal valve lift curve is the kinematic valve lift curve which contains no valve float or valve bounce. Therefore, we base the cost function for valve

train design on the squared difference between the simulated valve lift and the kinematic valve lift:

$$cf = \sum_{nspeeds} \sum_{i=1}^{360} \left(y_3^{sim}(i) - y_3^{kinem}(i) \right)^2 \tag{3.10}$$

where $nspeeds$ represents the number of engine speeds.

As the simulated valve lift overlaps the kinematic profile, the cost function approaches zero. Again, an implicit filtering algorithm can be utilized to minimize this cost function providing the best set of optimal parameters for cam profile and valve spring designs.

4. Parallelism. Below we discuss how we parallelize the model. We evaluate parallel performance by speedup and efficiency. Speedup is defined to be the ratio between the computational time of the serial (single processor) algorithm and the computational time of the parallel algorithm (ideally speedup $= n$ where n is the number of processors). Efficiency is defined to be the speedup divided by the number of processors (ideally efficiency $= 1$). Two necessary requirements for high speedup and efficiency are minimal communication times and an even load balance between the processors.

4.1. Valve Train Model Parallelism. Recall in § 3 that we model the valve train at a range of engine speeds. The modeling of the valve train at a given speed is completely independent of another speed, therefore they can be performed simultaneously.

The different processors communicate through a master-slave arrangement. The master first sends modeling information to each of the processors and then waits. Each slave simulates the valve train at a particular engine speed and sends simulation data back to the master. If more simulations need to be done, the master sends the modeling information for those speeds to the slaves which have already returned their simulation data to the master. Once all the simulation data for every speed is received by the master, the master then evaluates the objective function for the system identification (3.9) or the cam optimization (3.10).

The communication between the processors involves a large amount of information including multi-dimensional arrays with 10^4 elements. However, the total communication time is still dwarfed by the computational time of the simulation and does not have a significant effect on the speedup and efficiency.

If the number of engine speeds is a multiple of the number of slaves $(n-1)$ so that each processor has the same number of simulations to do, and if the cost of each simulation is approximately the same, then the load balancing will be excellent.

With the minimal communication and good load balancing described above we would expect a high speedup. Unfortunately, the cost of each simulation is not the same, because we are using a variable step size integrator within the simulation. Therefore, the length of time needed to complete the integration depends on the behavior of the step size. This causes a load imbalance which can lead to significant idle times for some of the processors and seriously hurt speedup.

Here is an example of what can happen with a range of variable work loads. In this example, we are integrating over 12 engine speeds in order to evaluate the objective function for the parameter identification problem. We have 12 work processors at our disposal and the following times for integration at each engine speed:

A single processor would perform a function evaluation in 7.56 seconds. Twelve work processors would perform a function evaluation in no less than the time of the most costly engine speed

Engine Speed	Time (secs)
8600	0.312
8700	0.307
8800	0.298
8900	0.286
9000	0.463
9100	0.410
9200	1.550
9300	0.310
9400	1.000
9500	0.984
9600	0.749
9700	0.894

TABLE 4.1
Example of variable work loads.

Computer#	Processor	Operating System
1	Pentium 90	Windows95
2	Pentium 133	Windows95
3	Pentium 133	Windows95
4	Dual 133 Pentiums	Windows NT
5	166 MMX Pentium	Windows95
6	233 MMX Pentium II	Windows NT
7	233 MMX Pentium II	Windows NT

TABLE 5.1
Processors and Operating Systems

which in this case is 1.55 seconds. Therefore, the speedup for the twelve processors is

$$\text{speedup} = \frac{7.56}{1.55} \approx 5$$

while ideally the speedup would be twelve.

5. Results.

5.1. Computing Environment. Computations were performed on a heterogeneous network of seven Dell personal computers. The processors and operating systems of the computers are listed in Table 5.1. The communication environment we use is PVM (Parallel Virtual Machine) version 5.4 beta.

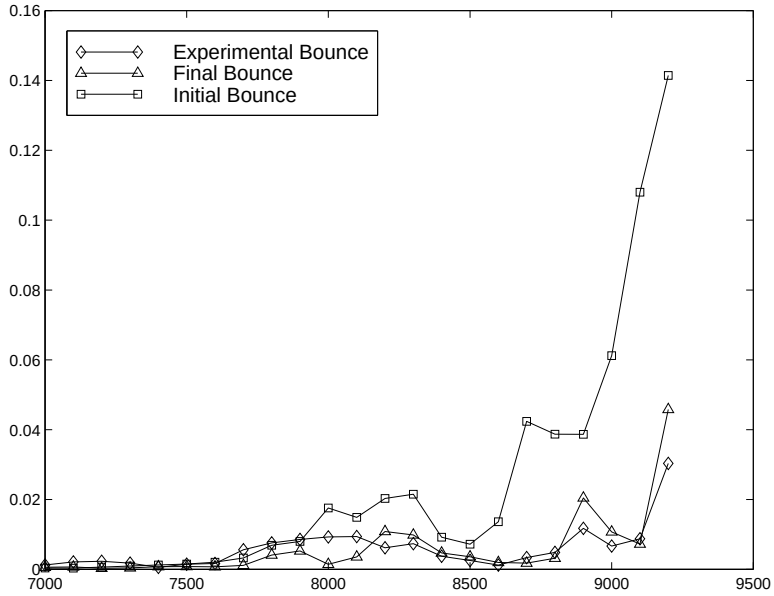


FIG. 5.1. *Ford Mechanical Valve Spring System.*

The network of computers was not dedicated to this one application, and therefore it became necessary to periodically stop the optimization in mid-iteration. The optimization was also interrupted when occurrences beyond our control intervened such as a thunderstorm or power surge. At the time of stoppage the current best point was saved and when the optimization was restarted it was used as the initial point.

5.2. System Identification. We show a typical system identification result for a mechanical valve train. The valve train consists of a Ford engine block containing a dual mechanical valve spring nest with a friction damper. The main cost of evaluating an objective function is due to the complexity involved in modeling the mechanical valve spring.

The boundary constraints for implicit filtering are dependent on the initial point. The upper constraints are

$$U_i = \frac{(x_0)_i}{s_i},$$

and the lower constraints are

$$L_i = (x_0)_i s_i,$$

where $0 < s_i < 1$.

The initial system parameters for this problem are taken from a previously converged result of an earlier model [8].

In Figure 5.1 we plot the experimental valve bounce, the simulated valve bounce for the initial iterate, and the bounce for the final result. The interested range of engine speeds is between 7000 to 9200 rpm, but we optimize over only 8 engine speeds from 8500 to 9200 rpm to reduce the computational costs. We see, however, that the valve bounce pattern is well predicted over the entire range of engine speeds. The simulation also predicts the limit speed, which occurs at approximately 9200 rpm, very well.

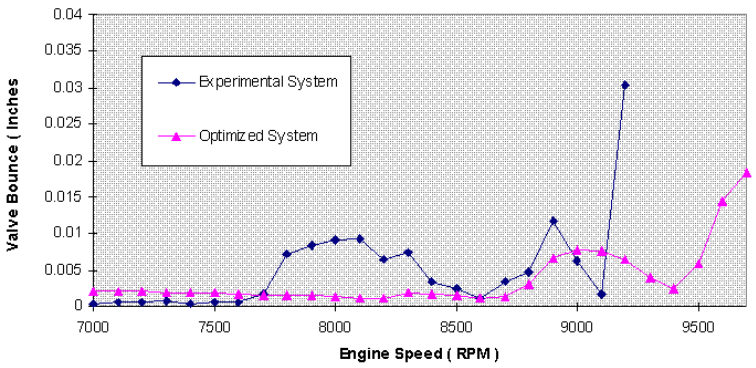


FIG. 5.2. *Lumped Mass Model.*

5.3. Camshaft Optimization. We optimize the cam profile and valve spring design for the Ford mechanical valve spring system. The goal is to reduce the valve bounce amplitude over the entire test range and increase limit speed. Eight engine speeds from 8500 to 9200 rpm are used in the optimization.

Large perturbations in the spline coefficients can result in a lobe profile that is non-convex and will cause the simulator to fail. IFFCO can deal with some failures of the objective function to return a value, treating them as violation of "hidden constraints" and assigning an artificial high value to the function to points at which the evaluation fails. In this case, however, many candidates for initial iterates cause the function to fail for every point that IFFCO tested and thereby caused the iteration to stagnate. We used a random search to find a usable initial point.

The boundary constraints were determined to be physically reasonable ranges for the parameters. Using implicit filtering we reduced the cost function of .95234 to .01002. Figure 5.2 illustrates the improvement in the behavior of the optimized system. The limit speed is increased by over 400 rpm from 9200 to about 9700 rpm and the magnitude of the valve bounce is significantly less than the original system for most of the engine speeds.

REFERENCES

[1] L. ARMIJO, *Minimization of functions having Lipschitz-continuous first partial derivatives*, Pacific J. Math., 16 (1966), pp. 1–3.

[2] D. B. BERTSEKAS, *On the Goldstein-Levitin-Polyak gradient projection method*, IEEE Trans. Autom. Control, 21 (1976), pp. 174–184.

[3] ———, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221–246.

[4] D. M. BORTZ AND C. T. KELLEY, *The simplex gradient and noisy optimization problems*, in Computational Methods in Optimal Design and Control, J. T. Borggaard, J. Burns, E. Cliff, and S. Schreck, eds., vol. 24 of Progress in Systems and Control Theory, Birkhäuser, Boston, 1998, pp. 77–90.

[5] C. G. BROYDEN, *Quasi-Newton methods and their application to function minimization*, Math. Comp., 21 (1967), pp. 368–381.

[6] ———, *A new double-rank minimization algorithm*, AMS Notices, 16 (1969), p. 670.

[7] J. CASH AND A. KARP, ACM transactions on Mathematical Software, 16 (1990), pp. 201–222.

[8] C. CHENG, *Analysis and Optimal Design of Three-Dimensional Valve Train Sy stems*, PhD thesis, North Carolina State University, 1997.

[9] C. CHIA, *Nonlinear Analysis of Plates*, McGraw Hill International Book Company, 1980.

[10] J. W. DAVID, C. Y. CHENG, T. D. CHOI, C. T. KELLEY, AND J. GABLONSKY, *Optimal design of high speed mechanical systems*, Tech. Rep. CRSC-TR97-18, North Carolina State University, Center for Research in Scientific Computation, July 1997. Mathematical Modeling and Scientific Computing, to appear.

[11] J. W. DAVID, C. T. KELLEY, AND C. Y. CHENG, *Use of an implicit filtering algorithm for mechanical system parameter identification*. SAE Paper 960358, 1996 SAE International Congress and Exposition Conference Proceedings, Modeling of CI and SI Engines, pp. 189–194.

[12] M. ETHERIDGE, *Preliminary performance of carbon-carbon valves in high speed pushrod type valve trains*, Master's thesis, North Carolina State University, Raleigh, North Carolina, 1998.

[13] A. V. FIACCO AND G. P. MCCORMICK, *Nonlinear Programming*, no. 4 in Classics in Applied Mathematics, SIAM, Philadelphia, 1990.

[14] R. FLETCHER, *A new approach to variable metric methods*, Comput. J., 13 (1970), pp. 317–322.

[15] P. GILMORE, *An Algorithm for Optimizing Functions with Multiple Minima*, PhD thesis, North Carolina State University, Raleigh, North Carolina, 1993.

[16] ———, *IFFCO: Implicit Filtering for Constrained Optimization*, Tech. Rep. CRSC-TR93-7, Center for Research in Scientific Computation, North Carolina State University, May 1993.

[17] P. GILMORE AND C. T. KELLEY, *An implicit filtering algorithm for optimization of functions with many local minima*, SIAM J. Optim., 5 (1995), pp. 269–285.

[18] D. GOLDFARB, *A family of variable metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26.

[19] C. T. KELLEY, *Iterative Methods for Optimization*. Department of Mathematics, North Carolina State University, to be published by SIAM in 1999, 1998.

[20] D. KIM AND J. DAVID, *A combined model for high speed valve train dynamics (partly li near and partly nonlinear)*, SAE Technical Paper 901726, (1990).

[21] J. MOTTERSHEAD, *Finite elements for dynamical analysis of helical rods*, International Journal of Mechanical Science, 22 (1980), pp. 267–283.

[22] ———, *The large displacements and dynamic stability of springs using helical finite elements*, International Journal of Mechanical Science, 24 (1982), pp. 547–558.

[23] D. PARK, *Design and optimization of valve train cam profiles in high spe ed ic engines*, M.S. Thesis, North Carolina State University, (1994).

[24] W. PRESS, S. TEUKOLSKY, W. VETTERLING, AND B. FLANNERY, *Numerical Recipes*, Cambridge University Press, 1992.

[25] K. RADHAKRISHNAN AND A. C. HINDMARSH, *Description and use of LSODE, the Livermore solver for ordinary differential equations*, Tech. Rep. URCL-ID-113855, Lawrence Livermore National Laboratory, December 1993.

[26] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970),

pp. 647–657.

[27] D. STONEKING, G. BILBRO, R. TREW, P. GILMORE, AND C. T. KELLEY, *Yield optimization using a GaAs process simulator coupled to a physical device model*, IEEE Transactions on Microwave Theory and Techniques, 40 (1992), pp. 1353–1363.

[28] D. E. STONEKING, G. L. BILBRO, R. J. TREW, P. GILMORE, AND C. T. KELLEY, *Yield optimization using a GaAs process simulator coupled to a physical device model*, in Proceedings IEEE/Cornell Conference on Advanced Concepts in High Speed Devices and Circuits, IEEE, 1991, pp. 374–383.

[29] T. A. WINSLOW, R. J. TREW, P. GILMORE, AND C. T. KELLEY, *Doping profiles for optimum class B performance of GaAs mesfet amplifiers*, in Proceedings IEEE/Cornell Conference on Advanced Concepts in High Speed Devices and Circuits, IEEE, 1991, pp. 188–197.

[30] ——, *Simulated performance optimization of GaAs MESFET amplifiers*, in Proceedings IEEE/Cornell Conference on Advanced Concepts in High Speed Devices and Circuits, IEEE, 1991, pp. 393–402.

[31] W. WITTRICK, *On elastic wave propagation in helical springs*, International Journal of Mechanical Science, 8 (1966), pp. 25–47.