

DETC98/DAC-5613

EXPLOITING DISCREPANCIES IN FUNCTION COMPUTATION TIME WITHIN OPTIMIZATION MODELS

Sigurd A. Nelson II*

Dept. of Mech. Engineering and Applied Mechanics
University of Michigan
Ann Arbor, Michigan 48109
Email: sigurd@engin.umich.edu

Panos Y. Papalambros

Dept. of Mech. Engineering and Applied Mechanics
University of Michigan
Ann Arbor, Michigan 48109
Email: pyp@umich.edu

ABSTRACT

Computational models in engineering often use a range of analysis functions within the same optimization problem, from finite element models to analytical expressions. The computational expense of these models often differs by orders of magnitude, and practical optimization algorithms should address this discrepancy. In this article, in an effort to reduce the number of expensive analyses, a technique is discussed for modifying trust region algorithms to utilize the inexpensive functions directly when calculating iterates. The technique is applied to a trust region algorithm of Yuan, and two numerical examples are given.

NOMENCLATURE

\mathbf{x} Vector of design variables.
 \mathbf{s} Algorithm step or proposed change in design variables.
 k Superscript iteration counter.
 $f(\mathbf{x})$ Objective function.
 $\mathbf{h}(\mathbf{x})$ Vector of equality constraints.
 $\mathbf{g}(\mathbf{x})$ Vector of inequality constraints.
 $\mathbf{c}(\mathbf{x})$ Vector of both inequality and equality constraints.
 i Subscript denoting that function is inexpensive to compute.
 e Subscript denoting that function is expensive to compute.
 j Subscript for indexing constraints.
 $\Phi(\mathbf{x})$ Penalty function.
 $\phi(\mathbf{x})$ Approximate penalty function.

\mathbf{B} Hessian estimate.
 Δ Trust region radius.
 σ Penalty parameter.
 γ Cauchy parameter.
 δ Cauchy parameter estimate.
 $+$ Superscript denoting constraint violation.
 $|||_{\infty}$ Infinity norm.

1 Introduction

The establishment of nonlinear programming as a common tool in engineering design has created the need to solve larger and more complex problems which use a growing range of computer models. For example, simulating the performance of an internal combustion engine may include a thermodynamic and combustion simulation to model combustion, a computational fluid dynamics code to model breathing processes, a finite element analysis to model the stresses and strains throughout the engine block, and so on.

Determining the best possible design must successfully incorporate all of the relevant models in an efficient, robust, and comprehensive manner, casting the optimal design problem in the form of a nonlinear program (NLP):

$$\begin{aligned} & \text{minimize } f(\mathbf{x}) \\ & \mathbf{x} \in \mathcal{R}^n \\ & \text{subject to } \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}(\mathbf{x}) = \mathbf{0} \end{aligned} \quad (1)$$

* Address all correspondence to this author.

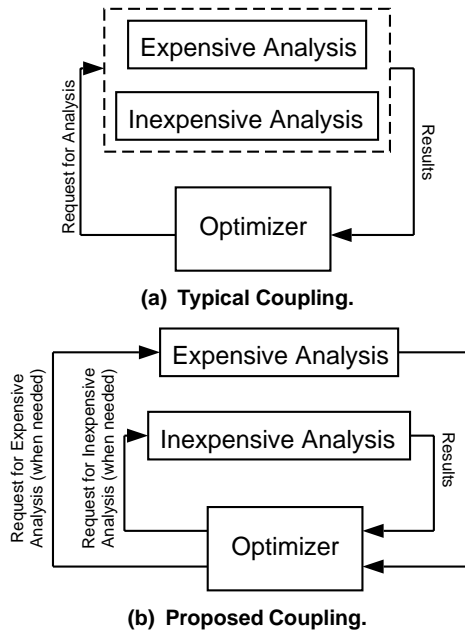


Figure 1. Couplings between an optimizer and the analyses that make up an optimal design problem.

Although many general purpose algorithms exist, for large computationally expensive nonlinear programming problems the structure of the design problem must be utilized to find a solution reliably and efficiently. Many researchers have focused on identifying and exploiting certain aspects of model structure. For example, Sobieski and Haftka (1997) survey situations common to the aerospace industry and multidisciplinary optimization, Conn et al. (1992) exploit an algebraic structure known as group partial separability, and Papalambros (1995) makes use of a structure known as model-based decomposition.

In this article, the focus is on tailoring an algorithm to accommodate the relative computational expense of the different models within an NLP formulation. Most optimization algorithms require that all functions be evaluated at the same time, a scenario depicted in Figure 1 (a). However, if some functions within an optimal design problem may use, say, a finite element code (taking perhaps five minutes to hours or more to run) and other functions depend upon a second code containing only analytical expressions (taking perhaps less than a second to run), an algorithm should be allowed to evaluate the computationally inexpensive functions substantially more times than the computationally expensive model, in an effort to use fewer expensive analyses, as depicted in Figure 1 (b).

Algorithms produce iterates by estimating the results of analysis. For example, in sequential quadratic programming (SQP) all of the functions are approximated linearly or quadratically. The premise here is that, when calculating iterates, the ap-

proximations of the inexpensive functions can be *replaced* with the inexpensive functions *themselves*. Conceptually this is similar to the work of Alexandrov et al. (1997) and Rodriguez et al. (1998) in that novel approximations are used for some functions. However, the approximations in this article are determined explicitly by the computational expense in an effort to tailor the optimization algorithm to match the *expense structure* of the problem, making the algorithm more efficient.

In the next section, a general technique for taking advantage of such computational discrepancy between different models within an NLP is discussed. In Section 3 a trust region algorithm of Yuan (1995) is reviewed and then modified. In Section 4 two numerical examples are presented, followed by the conclusions in Section 5.

2 The General Technique

To frame the problem in a general manner, it is assumed that in the NLP (1) the scalar objective f and the components of the vector-valued constraints \mathbf{g} and \mathbf{h} are sufficiently smooth. The computationally expensive functions are denoted by a subscript e , and the computationally inexpensive functions are denoted by a subscript i , so that problem (1) is written as:

$$\begin{aligned}
 & \text{minimize } f_i(\mathbf{x}) + f_e(\mathbf{x}) \\
 & \mathbf{x} \in \mathcal{R}^n \\
 & \text{subject to } \begin{aligned} & \mathbf{g}_i(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}_i(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}_e(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{h}_e(\mathbf{x}) = \mathbf{0} \end{aligned} \quad (2)
 \end{aligned}$$

Often the computationally inexpensive functions f_i , \mathbf{g}_i and \mathbf{h}_i are simply analytical expressions or quick computer routines which can be evaluated separately from the expensive functions. However, the inexpensive functions may also be *practical constraints* which represent safeguards that keep iterations from wandering into areas where the expensive functions are known to be inaccurate, uncomputable, or where the analysis crashes.

Typically, during each iteration a simple but approximate model is formulated based on function and gradient values at \mathbf{x}^k . This approximate model is used to determine a candidate step \mathbf{s}^k , and some appropriate criteria (e. g., a decrease in a penalty function) must be met for the step to be accepted. Because the computationally inexpensive functions are themselves approximated, candidate steps may violate some constraints unnecessarily. Since the most expensive functions may take several orders of magnitude more time to compute than the least expensive functions, any reasonable modification which would reduce the number of expensive analyses is warranted. The very simple modification proposed here is that when candidate steps are

calculated, the inexpensive functions be no longer approximated. Instead, the approximations of the inexpensive functions should be *replaced* by the functions themselves.

Conceptually this is simple. The SQP algorithm of Wilson (1963), Han (1976), and Powell (1978) uses a quadratic program to calculate candidate iterates, so the step calculation

$$\begin{aligned} & \text{minimize } \nabla f^k \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{B}^k \mathbf{s} \\ & \mathbf{s} \in \mathfrak{R}^n \\ & \text{subject to } \nabla \mathbf{g}^k \mathbf{s} + \mathbf{g}^k \leq \mathbf{0} \\ & \nabla \mathbf{h}^k \mathbf{s} + \mathbf{h}^k = \mathbf{0} \end{aligned} \quad (3)$$

would be replaced with

$$\begin{aligned} & \text{minimize } f_i(\mathbf{x}^k + \mathbf{s}) + \nabla f_e^k \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{B}^k \mathbf{s} \\ & \mathbf{s} \in \mathfrak{R}^n \\ & \text{subject to } \mathbf{g}_i(\mathbf{x}^k + \mathbf{s}) \leq \mathbf{0} \\ & \mathbf{h}_i(\mathbf{x}^k + \mathbf{s}) = \mathbf{0} \\ & \nabla \mathbf{g}_e^k \mathbf{s} + \mathbf{g}_e^k \leq \mathbf{0} \\ & \nabla \mathbf{h}_e^k \mathbf{s} + \mathbf{h}_e^k = \mathbf{0} \end{aligned} \quad (4)$$

As a convention, the superscript k is used to denote a function evaluated at \mathbf{x}^k , thus ∇f_e^k represents $\nabla f_e(\mathbf{x}^k)$. When a search direction \mathbf{s}^k is based on a quadratic program, one can guarantee that there exists some point along the line $\mathbf{x}^k + \alpha \mathbf{s}^k$, $0 < \alpha \leq 1$ which represents an improved design. However, because (4) may be highly nonlinear, there is no longer any guarantee that a line search will be successful. For this reason and others, the proposed application here is for trust region algorithms.

In a trust region algorithm the solutions to the approximate problem are restricted to some region around the current iterate \mathbf{x}^k . If $\mathbf{x}^k + \mathbf{s}^k$ does not produce the expected improvement, then the size of the region is reduced. For example, the trust region algorithm reviewed in Section 3 uses

$$\begin{aligned} & \text{minimize } \phi(\mathbf{s}) = \nabla f^k \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{B}^k \mathbf{s} + \sigma \|(\nabla \mathbf{c}^k \mathbf{s} + \mathbf{c}^k)^+\|_\infty \\ & \mathbf{s} \in \mathfrak{R}^n \\ & \text{subject to } \|\mathbf{s}\|_\infty \leq \Delta \end{aligned} \quad (5)$$

to calculate iterates. For brevity, the inequalities \mathbf{g} and equalities \mathbf{h} have been concatenated into a single vector $\mathbf{c}^T = [\mathbf{g}^T, \mathbf{h}^T]$, σ is a penalty parameter and the superscript $+$ denotes the amount of violation of a constraint. If the element c_j corresponds to an equality constraint, then $c_j^+ = |c_j|$; if c_j corresponds to an inequality constraint, then $c_j^+ = \max\{0, c_j\}$, and

$$\|\mathbf{c}^+(\mathbf{x})\|_\infty = \max\{0, \max_{j=1 \dots m_{\text{ineq}}} \{g_j\}, \max_{j=1 \dots m_{\text{eq}}} \{|h_j|\}\} \quad (6)$$

The framework for trust region algorithms allows for some additional theoretical benefits over line search methods, but such a discussion is not necessary here. The reader is referred to Dennis and Schnabel (1983) or Moré (1983).

When the linear and quadratic approximations of the inexpensive functions are replaced with the inexpensive functions themselves, (5) becomes:

$$\begin{aligned} & \text{minimize } \phi(\mathbf{s}) = f_i(\mathbf{x}^k + \mathbf{s}) \\ & \quad + \nabla f_e^k \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{B}^k \mathbf{s} \\ & \quad + \sigma \max \left(\begin{array}{l} \|\mathbf{c}_i^+(\mathbf{x}^k + \mathbf{s})\|_\infty \\ \|(\nabla \mathbf{c}_e^k \mathbf{s} + \mathbf{c}_e^k)^+\|_\infty \end{array} \right) \\ & \mathbf{s} \in \mathfrak{R}^n \\ & \text{subject to } \|\mathbf{s}\|_\infty \leq \Delta \end{aligned} \quad (7)$$

This model is itself a nonlinear programming problem that must be solved. Even if the computationally inexpensive functions take extra time to evaluate, it may be possible to solve this new model in much less time than it takes for a single evaluation of the expensive functions. Interestingly, (7) is a non-smooth minimization problem that can be solved by a simplified version of Yuan's algorithm. Care must be taken to ascertain that such complications are accounted for and do not detract from the overall computational savings.

The next section describes the basic elements of a specific trust region algorithm and the required modifications.

3 Modifying the Trust Region Algorithm

In order to balance improvements in the objective function and constraint violations, Yuan's algorithm (1995) uses the non-smooth penalty function

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + \sigma \|\mathbf{c}(\mathbf{x})^+\|_\infty \quad (8)$$

because a solution to (1) is also a minimizer of (8) if the penalty parameter σ is large enough and a few other mild assumptions hold.

Candidate steps are calculated by solving the previously mentioned approximate model (5), where each step is restricted to a region of radius Δ . A step is accepted if it decreases the penalty function $\Phi(\mathbf{x})$, and the trust region radius is altered at

each iteration according to how accurately the approximate function ϕ predicts changes in the actual penalty function Φ using the rule:

$$\Delta^{k+1} = \begin{cases} \max\{2\Delta, 4\|\mathbf{s}^k\|_\infty\} & \text{if } 0.9 < r \\ \Delta^k & \text{if } 0.1 \leq r \leq 0.9 \\ \min\{\Delta/4, \|\mathbf{s}^k\|_\infty/2\} & \text{if } r < 0.1 \end{cases} \quad (9)$$

where $r = \frac{\Phi(\mathbf{x}^k) - \Phi(\mathbf{x}^k + \mathbf{s}^k)}{\phi(\mathbf{0}) - \phi(\mathbf{s}^k)}$

If the approximate model ϕ gives a good prediction of Φ , the size of the region is increased, otherwise the region is made smaller in order to ensure convergence.

For convergence, each iteration must also satisfy the fractional Cauchy descent property (Powell 1975), which for the unconstrained case establishes that there exists some positive constant γ such that

$$\phi(\mathbf{0}) - \phi(\mathbf{s}^k) < \gamma \|\nabla f(\mathbf{x}^k)\| \min\{\Delta^k, \frac{\|\nabla f(\mathbf{x}^k)\|}{\|\mathbf{B}^k\|}\} \quad (10)$$

Yuan uses the parameter δ^k as an estimate for γ and the related condition:

$$\phi(\mathbf{0}) - \phi(\mathbf{s}^k) < \delta^k \sigma^k \min\{\Delta^k, \|\mathbf{c}^+(\mathbf{x}^k)\|_\infty\} \quad (11)$$

If (11) does not hold for some iteration, then σ is increased and δ is decreased for all future iterations.

The proposed change is that the approximate problem (5) used to define the step \mathbf{s}^k should be modified to directly include the computationally inexpensive functions, as in (7). This substitution, which occurs in Step 2, is the only necessary modification. The resulting algorithm is given below.

Algorithm 1. *The Modified Trust Region Algorithm.*

1. Set the outer iteration counter $k = 1$. Choose some \mathbf{x}^1 as an initial point and \mathbf{B}^1 as an initial Hessian estimate. Define the penalty parameter $\sigma^1 > 0$, trust region radius $\Delta^1 > 0$, and an estimate of the Cauchy parameter $\delta^1 > 0$.
2. Solve the approximate problem (7) and denote the solution \mathbf{s}^k . If $\mathbf{s}^k = \mathbf{0}$ then stop.
3. Calculate the penalty function Φ , approximate penalty function ϕ and ratio r at the point $\mathbf{x}^k + \mathbf{s}^k$. If $r > 0$ go to Step 4, otherwise set $\Delta^k = \|\mathbf{s}^k\|_\infty/4$, $\mathbf{x}^{k+1} = \mathbf{x}^k$, $k = k + 1$, and go to Step 2.
4. Alter the trust region radius according to (9).

5. Generate \mathbf{B}^{k+1} and if (11) does not hold then set $\sigma^{k+1} = 2\sigma^k$ and $\delta^{k+1} = \delta^k/4$, otherwise set $\sigma^{k+1} = \sigma^k$ and $\delta^{k+1} = \delta^k$.
6. Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k + 1$ and go to Step 2.

Some details omitted for clarity, such as the manner in which \mathbf{B} is used to estimate only the expensive functions, or the use of a correction step to ensure fast local convergence, can be found in Yuan (1995). Additionally, by modifying Algorithm 1 (specifically, restraining σ to be constant) it is possible to directly solve (7).

In the next section, two numerical examples are solved using both algorithms in order to make a comparison.

4 Example Applications

In this section, two numerical examples are discussed and solved with the original and modified algorithms. The first example is completely analytic and is used to give some insight into the performance of the proposed algorithm. The second example highlights typical situations in engineering design where the proposed algorithm should be found useful.

4.1 A Hock and Schittkowski Example

The example is Problem No. 7 from the Hock and Schittkowski collection (1981):

$$\begin{aligned} & \text{minimize } \ln(1 + x_1^2) - x_2 \\ & \mathbf{x} \in \mathfrak{R}^n \\ & \text{subject to } (1 + x_1^2)^2 + x_2^2 - 4 = 0 \end{aligned} \quad (12)$$

For illustrative purposes, the inexpensive functions will be the polynomials and the single ‘‘expensive’’ function will be the term $\ln(1 + x_1^2)$. The starting values of $\mathbf{x}^1 = [2, 2]^T$, $\sigma^1 = 1.0$, $\Delta^1 = 1.0$, and $\delta^1 = 0.1$ were used. Yuan’s original algorithm required a total of 39 separate analyses from both the expensive and the inexpensive functions (including those used for finite differences) to locate the solution $\mathbf{x}^* = [0, \sqrt{3}]^T$. In contrast, the proposed algorithm needed 29 expensive analyses and 176 inexpensive analyses.

The reason for this improvement can be found by examining the contour lines of the different approximate models ϕ in Figure 2. Figure 2 (b) portrays the approximate model used during the first iteration of Yuan’s original algorithm. Although minimizing ϕ in Figure 2 (b) will lead to an improvement in the actual penalty function Φ shown in Figure 2 (a), the two figures do not appear to be similar. Conversely, the proposed algorithm minimizes the ϕ shown in Figure 2 (c), which matches the actual penalty function much more closely. In fact, it is evident that the initial trust region radius of $\Delta = 1.0$ was conservative.

4.2 Valve Event Optimization of an IC Engine

MANDY (Chapman et. al. 1982) is a proprietary code of Ford Motor Co. used to analyze one-dimensional wave dynamics within the intake and exhaust manifold of gasoline engines. MANDY can be used to predict the torque at wide open throttle as a result of the compression waves inside the manifolds of the engine given a valve-lift profile and manifold geometry of an engine.

The six variables used in this study are the opening and closing times ($\theta_{i,o}$, $\theta_{i,c}$, $\theta_{x,o}$ and $\theta_{x,c}$) and maximum lifts (h_i and h_x) of both the intake and exhaust valve-lift profiles. In order to guarantee the life and durability of the valve train, the valve accelerations ($a_{i,max}$ and $a_{x,max}$) and decelerations ($d_{i,max}$ and $d_{x,max}$) must stay within certain predefined limits. However, the torque

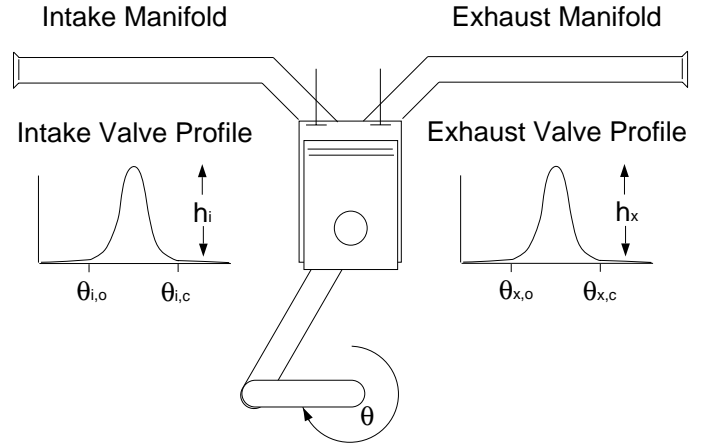


Figure 3. The MANDY analysis.

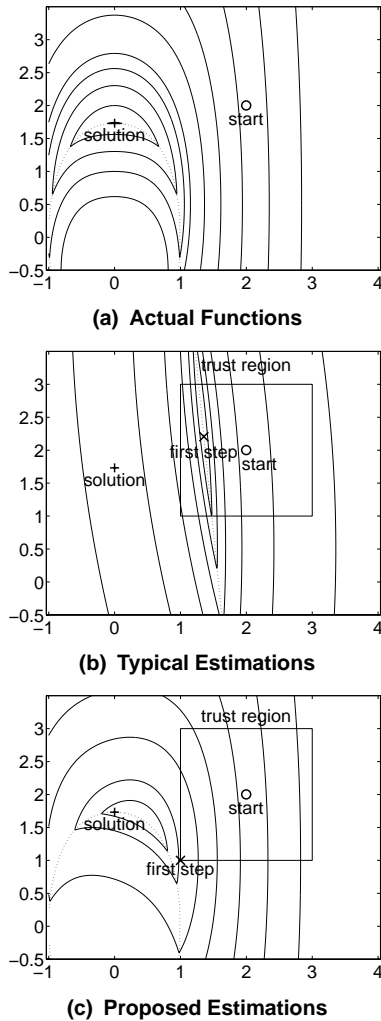


Figure 2. Contour lines for the penalty function Φ and the various approximations ϕ used to calculate iterates during the first iteration of the example in Section 4.1. The values $\sigma = 1$ and $\Delta = 1$ were used.

produced by the engine also depends on the valve events, so that the analyses is represented by Figure 3.

The goal of the study was to determine the valve events and maximum lifts which maximize the torque of a single-cylinder engine operating at 5000 RPM while keeping the valve-lift profile within acceptable limits. Thus, the optimization problem can be stated as

$$\begin{aligned}
 & \text{maximize} && T_{5000 \text{ RPM}}(\theta_{i,o}, \theta_{i,c}, \theta_{x,o}, \theta_{x,c}, h_i, h_x) \\
 & \theta_{i,o}, \theta_{i,c}, \theta_{x,o}, \theta_{x,c}, h_i, h_x \\
 & \text{subject to} && a_{i,max}(\theta_{i,o}, \theta_{i,c}, h_i) - a_{i,lim} \leq 0 \\
 & && a_{x,max}(\theta_{x,o}, \theta_{x,c}, h_x) - a_{x,lim} \leq 0 \\
 & && d_{i,lim} - d_{i,max}(\theta_{i,o}, \theta_{i,c}, h_i) \leq 0 \\
 & && d_{x,lim} - d_{x,max}(\theta_{x,o}, \theta_{x,c}, h_x) \leq 0
 \end{aligned} \quad (13)$$

Depending upon the computer architecture, the type of manifold, the number of cylinders and the simulated engine speeds, a single MANDY analysis may take anywhere between thirty seconds to an hour. However, another program that checks the velocities and accelerations of the valves takes much less than a second.

Using the same starting values for both algorithms, Yuan's algorithm found the optimum using 274 MANDY analysis (including those used for finite differences) whereas the proposed algorithm only required 139 MANDY analyses. This improvement is very attractive in practice, allowing for optimization to be performed overnight instead of requiring a few days.

5 Conclusions

In optimal design it is common to use computer models of varying complexity to formulate the design problem. The

main premise of this article was that computationally inexpensive functions can be used directly when calculating iterates in an optimization algorithm. Subsequent iterates are then more accurate and the overall algorithm requires fewer expensive analyses.

Naturally, this will increase the number of inexpensive analyses required. Therefore some judgment is necessary when designating which functions are to be considered inexpensive vs. expensive. Nonetheless, the technique presented addresses a common problem in practical design situations and has the potential to increase the robustness and efficiency of some optimization algorithms.

ACKNOWLEDGMENT

This research has been partially supported by the Automotive Research Center at the University of Michigan, a US Army Center of Excellence in Modeling and Simulation of Ground Vehicles, under Contract No. DAAE07-94-C-R094. This support is gratefully acknowledged.

The authors also would like to thank Dr. James Novak and the Ford Motor Co. for their help and permission to use the MANDY example.

REFERENCES

- Alexandrov, N.; Dennis Jr., J. E.; Lewis, R. M.; Torczon, V. 1997: A trust region framework for managing the use of approximation models in optimization, Technical Report 97-50, ICASE, Langley, VA.
- Chapman, M.; Novak, J. M.; Stein, R. A. 1982: Numerical modeling of inlet and exhaust flows in multi-cylinder internal combustion engines, in *Flows in ICE Engines* (Uzkan, T., ed.), ASME, 9 – 19.
- Conn, A. R.; Gould, N. I. M.; Toint, P. L. 1992: *LANCELOT, A FORTRAN package for large-scale nonlinear optimization*, New York: Springer-Verlag.
- Dennis, J.; Schnabel, R. 1983: *Numerical methods for unconstrained optimization*, Englewood Cliffs, New Jersey: Prentice-Hall.
- Han, S.-P. 1976: Superlinearly convergent variable metric algorithms for general nonlinear programming problems, *Mathematical Programming* 11, 263 – 282.
- Hock, W.; Schittkowski, K. 1981: *Test Examples for Nonlinear Programming Codes*, No. 187 in Lecture Notes in Economics and Mathematical Systems, Berlin: Springer - Verlag.
- Moré, J. J. 1983: Recent developments in algorithms and software for trust region methods, in *Mathematical Programming: The State of the Art, Bonn 1982* (Bachem, A.; Grötschel, M.; Korte, B., eds.), New York: Springer - Verlag, 258 – 287.
- Papalambros, P. Y. 1995: Optimal design of mechanical engineering systems, *ASME Journal of Mechanical Design* 117, 55 – 62.
- Powell, M. J. D. 1975: Convergence properties of a class of minimization algorithms, in *Nonlinear Programming 2* (Mangasarian, O. L.; Meyer, R. R.; Robinson, S. M., eds.), New York: Academic Press, 1 – 27.
- Powell, M. J. D. 1978: The convergence of variable metric methods for nonlinear constrained optimization calculations, in *Nonlinear Programming 3* (Mangasarian, O. L.; Meyer, R. R.; Robinson, S. M., eds.), New York: Academic Press, 27 – 64.
- Rodriguez, J. F.; Renaud, J. E.; Watson, L. T. 1998: Trust region augmented lagrangian methods for sequential response surface approximation and optimization, *ASME Journal of Mechanical Design* 120, 58 – 66.
- Sobieszcanski-Sobieski, J.; Haftka, R. T. 1997: Multidisciplinary aerospace design optimization: a survey of recent developments, *Structural Optimization* 14, 1 – 23.
- Wilson, R. B. 1963: *A simplicial algorithm for concave programming*, Ph.D. thesis, Harvard University, Graduate School of Business Administration.
- Yuan, Y.-X. 1995: On the convergence of a new trust region algorithm, *Numerische Mathematik* 70, 515–539.