A Designer's Spreadsheet

R. Ramaswamy
Hewlett Packard Española, S.A.,
Barcelona, Spain

Karl Ulrich
The Wharton School,
University of Pennsylvania,
Philadelphia, PA

We have observed that many designers use spreadsheet programs for preliminary parametric design calculations. The primary reason for this appears to be the flexibility and ease of use of spreadsheets as compared to more advanced analysis tools. However, current commercially available spreadsheets were originally designed for finance and accounting and do not naturally support the analytical techniques used in design and engineering. In this paper, we propose a fundamentally different kind of spreadsheet for use in preliminary parametric design. We describe the features of this spreadsheet that make it especially suitable for preliminary parametric design, discuss the theoretical basis for implementing it and present the results of preliminary user tests.

1 Introduction

1.1 Use of Mathematical Modeling in Design and Engineering. It is common practice in design and engineering to use mathematical models as decision-making tools. To do this an engineer sets up a mathematical model that relates the design variables to the performance of a design, and uses this model to evaluate the effect of choosing a particular set of design variables. Three specific situations where mathematical models may be used in this way are:

(1) In the concept exploration stage, to test whether a concept is viable or to get a feel for the general performance of a concept.
(2) During system-level design, where a number of key design variables must be considered simultaneously in order to achieve a design that satisfies some criteria.
(3) In detail design, to size components or compute safety factors.

This is not an exhaustive list and mathematical models are used in a variety of other situations.

In this paper we concentrate primarily on models with a relatively small number of governing equations that are used frequently, probably several times a week. Models such as finite element analysis, or other large analysis programs that require a significant investment of resources and are used infrequently are excluded. We deal with problems that an engineer can formulate, code, and use to make a decision in a single session lasting a few hours. Though only a fraction of the mathematical models used in engineering satisfy these criteria, we believe that the use of simple models can have a far-reaching effect on a design and hence building tools to solve simple models is worthwhile.

1.2 Current Practice. In an industry-sponsored design project in which we participated, we observed that the ordinary computer-based spreadsheet was the computer tool most often used to evaluate designs. We repeatedly observed that even though a variety of more sophisticated tools were readily available, engineers would use spreadsheets for preliminary design calculations. This initial observation was confirmed by a more formal data collection effort (Ramaswamy, 1993) that included tracking all the written communication in a design project, interviewing designers at several local companies and examining existing survey data on computer tool usage (Dobson and Wolff, 1984; Kral, 1992; Slocum, 1992).

Our search revealed that spreadsheets are most often used in the preliminary design stage. For example, to compare several candidate design configurations, a common practice is to use simple spreadsheet models of each configuration. Configurations can then be compared by manipulating the inputs to the various models and observing the predicted performance. As the configurations are modeled separately, they can be efficiently compared without going into the problem of configuration selection (Ramaswamy et al., 1993). On other occasions, we found the spreadsheet being used to understand a specific design configuration better. For example, an engineer may use a spreadsheet model to decide levels for various design variables in order to achieve a specific performance. This may be in the context of preliminary design or redesign.

The reasons for widespread use of spreadsheets by designers appear to be consistent with the existing views on why spreadsheets are successful in general. One prevalent view is that in domains where exact solutions are not possible a "restricted but extremely flexible and easy to use set of computational tools" (Lai et al., 1988) is the best substitute for a tool that provides an exact solution. Of the various types of computer software available for designers today, the spreadsheet comes closest to being this set of tools, especially in the preliminary stages of design.

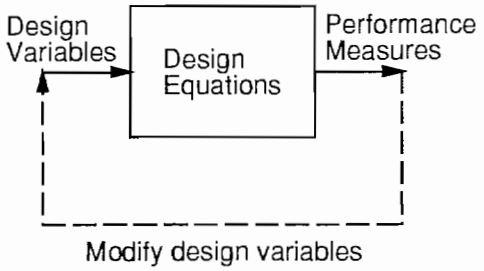
A number of existing systems attempt to provide a flexible design environment. For example, in several commercial CAD systems a user can specify a new geometry by modifying any of the available geometric variables. This may be done by either typing in new values for these variables from the keyboard or, in some systems, modifications can be made interactively on a computer screen by "pulling" on geometric entities using the mouse. However, the functionality is typically limited to geometric entities and constraints. Other systems (Gonda et al., 1992; Milley, 1972) have attempted to provide a more generic capability, but have focussed on large systems of equations and do not provide immediate feedback to the designer.

We believe that a need exists for a tool that is intended specifically for use on smaller problems. This tool will minimize the time needed to set up a problem, accommodate a broad variety of equation types, not require the designer to pre-specify inputs and outputs and still provide virtually instant feedback to the designer. In this paper, we present preliminary evidence to support the assertion that such a system will enhance both designer productivity and design quality.

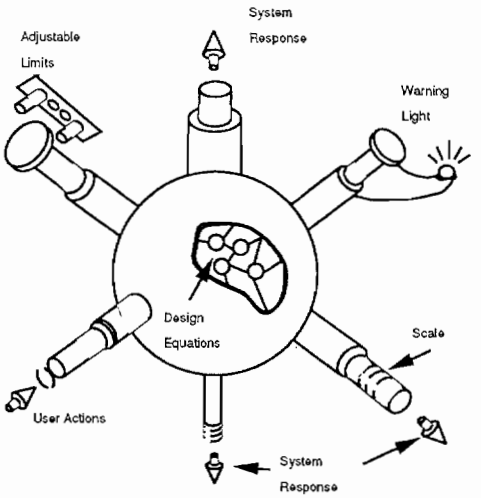
1.3 Conceptual Approach. The relationship between a financial spreadsheet and a design spreadsheet is analogous to the relationship between the conventional and modern view of parametric design. This is displayed graphically in Fig. 1.

The conventional view, shown in part (a), rigidly defines design variables, performance measures and the manner in which design iterations are performed. Most commercial

Contributed by the Design Automation Committee for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received June 1993; revised May 1996. Associate Technical Editor: D. A. Hoeltzel.



(a) Conventional View of Parametric Design



(b) Causal-Dependency View (reproduced from [Ser87])

Fig. 1 Different views of parametric design

spreadsheet programs support this "input-output" mode of functioning very well and this explains their utility as a "what-if" tool for designers. However, in most real design situations, the "what-if" question can be far more complex. For example, the designer may want to change some of the output variables directly or to designate some of the constants as variables. As a result, practical parametric design is a dynamic activity where the inputs, the variable bounds and causality are constantly changing. A model to describe this, the causal dependency sphere, was conceived by Prof. Woodie Flowers at MIT and has been described by Serrano (1987). It is shown in Fig. 1(b). We believe this is a better way of viewing parametric design than the input-output model in Fig. 1(a). In this paper we discuss the problem of creating and testing a computer tool to support the causal dependency view of parametric design.

1.4 An Introductory Example. Consider the problem of designing an elastic, rectangular cross-section beam to support a given bending moment. The equations governing this beam are as follows:

$$\sigma = \frac{Mh}{2I}; \quad I = \frac{bh^3}{12}$$

M Bending Moment
σ Maximum Allowable Normal Stress
I Moment of Inertia of Cross Section
b Width of Cross Section
h Height of Cross Section

This formulation of the problem implicitly assumes the variables b, h and M as the inputs. If these values are specified, the other two variables, I and σ, may be computed (in that order) and displayed. This can be easily coded in a spreadsheet for convenience. However, if the designer then finds a material of known σ and wants to know the maximum permissible moment, the first equation must be rewritten as $M = 2\sigma I/h$ and the spreadsheet modified to accommodate this new choice of design variables.

We propose that an alternative and superior mode of functioning is one in which the designer can directly control the value of any of the variables while the computer ensures that the design equations are enforced. An interface to such a tool, which we call the Designer's Spreadsheet, is shown in Fig. 2. The designer can use the up/down arrows to directly increase or decrease the value of any of the variables, even those that were originally outputs. As the designer modifies one variable, the computer adjusts the values of the other variables so that the design equations remain satisfied. The designer also has the ability to lock a variable at its current value using the button with the lock icon on it. For example, in Fig. 2 the designer has fixed the moment of inertia, I, of the beam and can experiment with various combinations of b and h that preserve the value of I. This type of non-directional system allows the designer to interact with the design equations far more flexibly than when using a conventional spreadsheet. The other important facility provided by the Designer's Spreadsheet is the ability to monitor and edit the constraints. The constraint window shows all the constraints. Inequalities may be not satisfied, labeled as "VIOL" (for violation), or satisfied, labeled as "OK". This is indicated by the label to the left of each inequality constraint. Equality constraints are always satisfied. Inequalities may be coerced into equalities by using the "Enforce" button to the right of each inequality. Another window, which is not shown in Fig. 2, is used for notifying the user of various system actions.

There are at least two important and distinct questions to be answered about the Designer's Spreadsheet. These are:

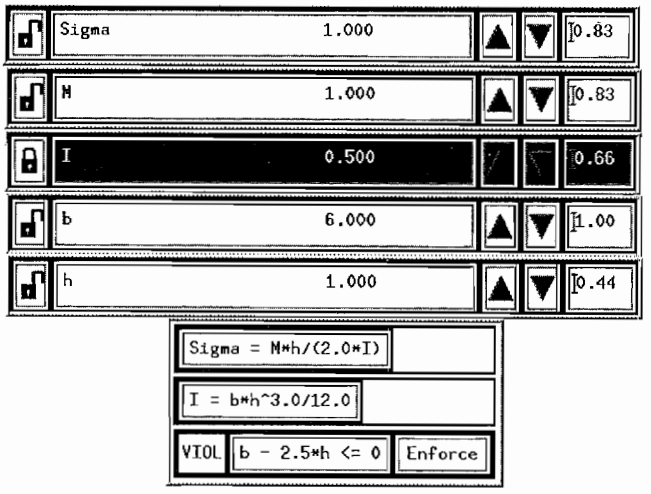


Fig. 2 Screen dump of designer's spreadsheet interface

(1) What are the theoretical obstacles to implementing this tool?
(2) How will such a tool affect designer productivity and design quality?

In this paper we address both of these questions, though we concentrate on the first one.

Before continuing, it is important to state some restrictions on the types of problems that we consider. These restrictions are a product of the fact that this tool is intended only as a preliminary design tool and not as a substitute for subsequent analysis tools.

(1) Functional forms must be either algebraic or transcendental functions (either explicit or implicit form). Other types of functions such as recursive functions and iterative computer programs are explicitly excluded.
(2) At this time our approach assumes continuous variables. We are investigating strategies for handling integer and discrete variables under special circumstances.
(3) Explicit consideration of geometry is excluded, though geometric parameters may occur as variables in any equation.
(4) Size is restricted. The design spreadsheet is not intended for problems where there are more than approximately fifteen equations and a similar number of variables.

Data we have collected indicates that despite the above restrictions there is a large class of important design problems to which the techniques we are about to discuss are applicable (Ramaswamy, 1993).

1.5 Roadmap. The next section is a discussion of the theoretical basis needed for providing the desired features in a computer program along with references to the relevant prior work. This is followed by a description, with examples, of the computer implementation of the Designer's Spreadsheet. The final section contains some preliminary user test results and a plan for future work. All the issues discussed in this paper, and some others, are discussed in greater detail in the reference (Ramaswamy, 1993).

2 Theory

The basic theoretical problem faced in this research is to find paths in the design space that are composed of points that satisfy both the design constraints and other constraints imposed by the designer. Hence, as the designer changes the value of a variable, the system must adjust all the remaining variables so that the constraints remain satisfied. The obvious approach of repeatedly solving the nonlinear equations using numerical methods such as Newton-Raphson iteration is too slow and unreliable. In this work we attempt to avoid this by using two major strategies:

- Exploit problem structure to create a plan for re-satisfying the constraints when a variable value is changed by the designer.
- If a suitable plan cannot be found, constrain the user to making continuous changes in variable values and use differential numerical computation to maintain rather than to solve the constraints.

We will now discuss each of the above strategies in greater detail.

2.1 Exploiting Problem Structure. In a conventional spreadsheet, the structure of the problem must be such that the variables V, are divided into two sets, the inputs or knowns K, and the outputs or unknowns U. The input-output behavior of the entire spreadsheet can be thus described by an equation of the form $G(K) = U$, where G is a (vector) function. This partitioning of variables into knowns and unknowns is rigid, and if the designer wants to change the membership of these sets, the entire spreadsheet must be recoded specifically for the new K and U.

The problem of automatically computing partitions with different properties has been addressed by several researchers (Serrano, 1987; Light, 1980). Light examined the problem in the context of creating a variational geometry system and Serrano addressed the problem of general constraint management when the membership of sets K and U above is changing (Serrano, 1987). Serrano's program would take a specific partitioning of V into K and U and then use graph searching to compute a plan such that all the members of U could be computed without solving any simultaneous equations. This is not always possible, but when possible it avoids the typical problems associated with the numerical solution of nonlinear simultaneous equations. We now discuss ways of extending Serrano's results to reduce the numerical computation in the Designer's Spreadsheet.

First, imagine a conventional spreadsheet with the capability that every time the user wants to modify a variable the spreadsheet is reformulated internally such that the variable being modified is one of the inputs of the spreadsheet. Having the modified variable as an input of the spreadsheet is very convenient because if this input is changed, the changes can be propagated simply by recomputing the spreadsheet. To do this we find for each variable x_i a partitioning of the set of variables V into knowns K and unknowns U such that $x_i \in K$ and such that $U = G(K)$ can be solved in closed form. If it is possible to perform such a partitioning, the values of all the variables in the spreadsheet can be updated without any numerical search. In a real design situation, it is also necessary to account for a set of variables F that the designer wants to keep fixed and the above condition must be expanded to state that the variables in F should be members of K, i.e. $F \subset K$.

This observation that fixing a variable essentially corresponds to including it in the set of knowns makes it possible to use Serrano's partitioning algorithm to precompute plans for updating each variable subject to different sets of fixed variables. When a variable is changed, if a suitable plan is available it is retrieved and used. Otherwise, the differential numerical techniques described in the next section must be applied. While Serrano's work considered the case where the sets K and U were specified completely, we deal with a situation where only some members of the set K have been specified and the others can be chosen to suit our convenience.

The strategy of precomputing plans is limited by the fact that the number of plans that must be precomputed grows quite rapidly as the number of variables increases. We can avoid any pre-computation by using a simple heuristic algorithm that attempts to find plans on-the-fly based on the adjacency matrix of the equation set. We have chosen to use the adjacency matrix as it is one of the most easily derived global characteristics of an equation set. The basic idea is that when a variable in an equation is changed we can use another variable in the same equation to compensate for the change and thus keep the equation satisfied. Since variables may appear in several equations, the effects of changing any variable must be propagated. The algorithm we describe attempts to do this while modifying the least possible numbers of variables in the process.

Consider the following equations for a beam:

$$\sigma - \frac{MY}{I} = 0 \tag{1}$$

$$M - FL = 0 \tag{2}$$

$$I - \frac{BH^3}{12} = 0 \tag{3}$$

$$Y - \frac{H}{2} = 0 \tag{4}$$

$$K - \frac{3EI}{L^3} = 0 \tag{5}$$

The adjacency matrix for these equations is shown below and has rows corresponding to equations and columns corresponding to variables. A "1" indicates occurrence of a variable and a "0" indicates non-occurrence.

	σ	M	Y	F	L	I	B	H	K	E
Eq 1	1	1	1	0	0	1	0	0	0	0
Eq 2	0	1	0	1	1	0	0	0	0	0
Eq 3	0	0	0	0	0	1	1	1	0	0
Eq 4	0	0	1	0	0	0	0	1	0	0
Eq 5	0	0	0	0	1	1	0	0	1	1

Based on this adjacency matrix, it is possible to generate by inspection the following plan for maintaining the constraints under a perturbation of M.

- Recompute σ in equation 1.
- Recompute F in Eq. (2).

The plan is found using the heuristic that the best variables to change are the ones that appear in the least number of equations. The logic of the search used to identify this plan is:

(1) The variable M appears in Eqs. (1) and (2). Hence, we need to recompute at least one variable in each of these equations.
(2) Processing Eq. (1) first and using the heuristic given above, the algorithm picks variable σ whose effect is limited to Eq. (1). Hence, no new equations need to be added to the list of equations to be processed.
(3) Next processing Eq. (2), it is found that variable F is not in any other equation and hence this variable is changed to keep Eq. (2) satisfied. This terminates the search.

Of course, this is only one of several alternative plans that can be found. For example, an alternative (and somewhat longer) plan is:

(1) Recompute σ in Eq. (1).
(2) Recompute L in Eq. (2).
(3) Recompute K in Eq. (5).

Our implementation is capable of locating all such plans. The algorithm also has to be augmented to account for the fact that there is a set of fixed variables and for equations where the causality cannot be changed arbitrarily. This can be achieved quite simply by introducing additional zeroes in the adjacency matrix. To have a plan for each variable is beneficial as it enables us to avoid differential numerical manipulation altogether.

While the availability of a suitable plan as described above simplifies computation, in most cases a combination of numerical and symbolic computation is necessary. When no plans can be found and the designer is allowed to change variables arbitrarily it is necessary to solve nonlinear simultaneous equations to compute a solution. In the following section we discuss an approach to performing constraint maintenance using differential numerical computation. This technique allows us to avoid solving nonlinear equations altogether. The price paid for this is the fact that the designer must now change variables continuously rather than in random increments. In many practical situations, this is not a big handicap since design changes tend to be incremental.

2.2 Numerical Computation. A lot of research has been performed in the area of computer graphics to achieve real-time motion of objects subject to constraints (Witkin et al., 1990; Gleicher and Witkin, 1991). This has resulted in many efficient ways of solving and, more importantly, of updating constraints. Though most graphics work is concentrated on cases where the constraints are equations of motion, the results can be generalized to any type of constraints, including design constraints. An early attempt at applying some of these results to a mechanical design problem is presented by Tidd et al. (1992). They describe a computer program that helps them design a device with a specified dynamic behavior. They report that the system had problems with performance and concluded that due to the fact that design problems are tightly constrained, their efforts were not very successful.

We believe that by systematically removing all the computational overhead and by considering somewhat simpler problems than the one attempted by Tidd, it is possible to build a useful computer tool with similar functionality to the application they describe.

The fundamental idea is to start at a feasible point and to use a local, linear approximation to the constraint functions to compute a new state such that the constraints remain satisfied under the perturbation. This approach itself is not new, having already been tried and tested quite extensively in classical optimization research. What is new is the realization that these techniques can be easily adapted for use in the type of interactive design tool that we are proposing. We now explain the mathematical details of our approach. Those readers who are not interested in this development may skip to the section on using the Designer's Spreadsheet.

Let the constraint equations be expressed as $C(X) = 0$, where X is a vector of variables. Assume that we begin at some state X_o where the constraints are satisfied. The first-order condition for a movement ΔX on the constraint surface defined by $C(X) = 0$ can be derived from the Taylor series expansion (TSE) of $C(X)$ about X_o. The TSE, to first order, is as follows:

$$C(X_o + \Delta X) \cong C(X_o) + (\nabla C)\Delta X$$

Hence, if the constraints must remain satisfied while effecting a change ΔX in X_o, the condition that must be satisfied is:

$$C(X_o + \Delta X) = C(X_o) = 0$$

Therefore,

$$(\nabla C)\Delta X = 0$$

Hence the first order condition is that the differential motion must lie in the nullspace of the constraint Jacobian, $J = \nabla C$. The nullspace is a vector space that spans all the possible solutions to the equation $(\nabla C)\Delta X = 0$.

Based on this observation, we propose a constrained motion algorithm with the following steps:

(1) Allow the user to pick a desired change in the variables (ΔX_d). In general, taking this step will not maintain the constraints (i.e. $(\nabla C)\Delta X_d \neq 0$).
(2) Transform ΔX_d into a change ΔX_a that preserves the constraints (i.e. $(\nabla C)\Delta X_a = 0$).
(3) Update the values of variables by augmenting them with the legal step ΔX_a. (i.e. $X_{n+1} = X_n + \Delta X_a$).
(4) Periodically monitor error that accumulates due to the linear approximation and correct it as necessary.

The design spreadsheet continuously executes the above algorithm as the user specifies ΔX_d via the user-interface. Continuously updating the values of the variables creates the impression that the values of the variables are changing smoothly in the direction specified by the user. In reality this is only an impression since ΔX_d and ΔX_a are not identical. We now discuss each of the steps in the above algorithm in more detail.

(1) Picking a desired direction: This consists of specifying an n-vector $[X_d]$ which indicates the relative changes in each of the design variables desired by the designer.

This vector corresponds to a direction in the design space and therefore has to be specified at each step by the user. There are several ways of doing this. The most obvious is to allow the user only to pick a single variable and to specify the sign of the desired change in that variable. This can be translated into the desired direction [0 0 0 . . .±1 . . .0 0] where the only non-zero element of the n-vector corresponds to the variable the user is pulling on. Of course, this can never be a legal direction in any coupled equation set and hence will not be preserved under the transformation performed in step 2. This strategy is currently used by our prototype system and is implemented by having up/down arrows for each variable. Other, more complex, options such as allowing the user to specify a local objective function or even an arbitrary vector are possible. Our experience with this has been that users often specify directions that have no feasible projections and hence admit no progress. As a result we use the single-variable option described above.

(2) Transformation to a legal direction: This step involves taking the user specified change vector and transforming it into a legal vector (i.e. one that satisfies $J\Delta X = 0$ for that point). It is a standard result of linear algebra that any vector can be projected into a vector space by using a suitable projection matrix. Hence, an arbitrary vector can be projected into the nullspace of the Jacobian to yield a legal vector. The projection matrix P for this is:

$$P = I - J^T(JJ^T)^{-1}J$$

where I is the identity matrix. In practice, the projection is computed without actually computing $(JJ^T)^{-1}$ explicitly. Instead we can solve for a vector y in $JJ^Ty = Jx_d$ and the projected vector Px_d can then be directly computed as $Px_d = x_d - J^Ty$. See Strang (1988) for more details. An alternative to the method shown above is first to compute the nullspace basis of the Jacobian and then to use a least squares algorithm to construct a vector from the basis that is as close to x_d as possible. We refer to this as the constructive approach. The constructive and the projective approach yield identical results if the Euclidean norm is used in both cases. The approaches are also equivalent in terms of the computation involved. In both methods, if at each step one wishes to compute the projections of several vectors, this can be done with practically no computational overhead. For example, in the projection algorithm it involves only solving the intermediate equation $JJ^Ty = Jx_d$ for many different values of x_d. This is analogous to solving the equation $Ax = b$ for different b's using only the same computation as for a single b and is a standard result of numerical analysis. Another point to note is that the transformation process tends to be somewhat scale dependent. As a result it is important to provide well-scaled equations to the Designer's Spreadsheet. If the equations are badly scaled, variables with smaller magnitudes basically get ignored during constraint maintenance.

(3) Taking a legal step: Once the legal direction ΔX_a is computed the algorithm must take a small step ϵ along that direction. Hence, the new design state X_{n+1} can be computed using the update formula,

$$X_{n+1} = X_n + \epsilon \frac{\Delta X_a}{\|\Delta X_a\|}$$

The constant ϵ is chosen keeping in mind the first order assumptions that we have made. There are two important conditions that must be kept in mind when choosing the step size. These are:

(a) Constraint activity: In a problem with inequality constraints, the same set of inequality constraints should be satisfied both before and after the step.

(b) Error correction: There is a tradeoff between the effort required to take a step with a small ϵ and the effort required to perform error correction. Hence, ϵ must be chosen such that error correction is not noticeable to the user.

(4) Error Correction: Since linear approximations are used to maintain the constraints there is an error due to the higher order terms which accumulates over multiple steps. Hence $C(X) = E$ instead of $C(X) = 0$. When this error exceeds acceptable bounds, the system corrects this error by using an under-constrained Newton-Raphson algorithm. This algorithm computes the most efficient error correction step ΔX^* by minimizing $\|\Delta X\|$ subject to $J\Delta X = -E$. This usually corrects the error within a few iterations and hence is transparent to the user. Occasionally, the method fails due to high second derivatives on the constraint surface and the step size ϵ mentioned above must be reduced. The current implementation uses a simple heuristic to increase and decrease step size as necessary. Of course, this not a guaranteed strategy. If several automatic step size reductions do not result in convergence, the program will pause and request directions from the user.

3 Using the Designer's Spreadsheet

Our current implementation of the program is written in the C programming language and runs under the X window system with the Motif window manager. The code itself is only a few thousand lines and so porting or even rewriting for different environments should be relatively easy.

The user interface of the tool was described briefly in the earlier part of the paper and is shown in Fig. 2. The only unexplained feature is the number shown to the right of the up/down buttons for each variable. These numbers are the magnitudes of the projection of the unit vectors along each variable axis onto the tangent plane. A very low value for a variable indicates that the axis corresponding to that variable is almost normal to the tangent plane. This information is displayed to help users decide which variables may be fruitfully manipulated (e.g. small magnitudes indicate that the variable is tightly constrained and cannot be changed without affecting a lot of other variables).

We now discuss some examples of how the various functions provided by this tool may be used for design. An important fact to keep in mind during this discussion is that this program is an exploration tool for preliminary design and is not a substitute for detailed optimization performed later in the product development process. Also, the tool does not generate good designs for a designer. It merely makes it much easier for the designer to rapidly try a variety of different options. The overall control and responsibility for the design still lie with the designer.

Some of the basic functions the user is provided with are:

(1) Creating a design model. Ease of creating models is a critical factor in determining user acceptance. An important reason for the success of programs such as Microsoft Excel is that the authors have expended considerable effort in providing various convenient ways of creating models. We feel that the most convenient interface for entering equations is simply to type them in. Researchers such as Piela (1989) have had similar findings. Our current implementation does not support this, requiring the equations to be compiled in as a procedure. However, an

equation editor is fairly simple to implement and we do not view it as a research problem.

(2) Perturbation analysis. The interface of the Designer's Spreadsheet makes it well suited for perturbation analysis, which is an important part of preliminary design. Small changes in variable values can be easily effected while simultaneously controlling the values of the other variables using the lock facility. One of the problems mentioned by earlier researchers (Tidd et al., 1992) is that the variables often change in a non-intuitive manner. This is a direct result of the least-squares strategy used by the system. Our experience was that as the available degrees of freedom came down, the results became more intuitive because the system had fewer options. We also observed that as the number of variables becomes large, it is hard for the designer to keep track of changes the system is making.

(3) Constraint manipulation: The Designer's Spreadsheet provides two important capabilities:

- Fix specific variables.
- Force inequality constraints to be satisfied as equalities.

These capabilities are very useful to designers. For example, if the designer is increasing a particular variable and an inequality constraint becomes active there are three options: (1) allow the violation, (2) enforce the constraint as a strict equality, or (3) enforce the constraint as an equality for now but relax it if motion is towards the feasible side of the constraint.

(4) Snapshot facility. Because the system changes any variable that is not fixed, designers are forced to express constraints explicitly. The disadvantage of this approach is that since the system is path dependent, the final state reached may depend on the order in which variables are fixed. A snapshot facility is available that enables the designer to record interesting points in the design space and to return to them instantly if desired.

(5) Looking for interesting spots in the design space. One of the characteristics of design is that it is not always a systematic process; many design insights are discovered accidentally. For example, there may be regions in the design space where variables that are usually coupled become locally decoupled. The design spreadsheet can easily incorporate a set of "demons" that subject all the points in the space that the designer traverses to a set of tests and alert the designer as appropriate. This "demon" facility is currently not implemented.

4 Preliminary User Testing

To develop a preliminary assessment for the utility of the Designer's Spreadsheet to actual users, we conducted a series of tests with students from a graduate design class at MIT. The tests were designed to compare performance of student designers when using the Designer's Spreadsheet to their performance using a conventional spreadsheet. Due to resource constraints the sample size for this test had to be quite small. Hence, the preliminary results presented in this section must be confirmed by further, more extensive, testing.

In these tests, nine subjects (S1–S9) were tested on two different problems such that each subject used the two tools to solve two different design problems. The experimental design is shown below.

	Problem 1	Problem 2
Conventional spreadsheet	*S1*, *S3*, *S5*, *S7*, *S9*	*S2*, *S4*, *S6*, *S8*
Designer's spreadsheet	*S2*, *S4*, *S6*, *S8*	*S1*, *S3*, *S5*, *S7*, *S9*

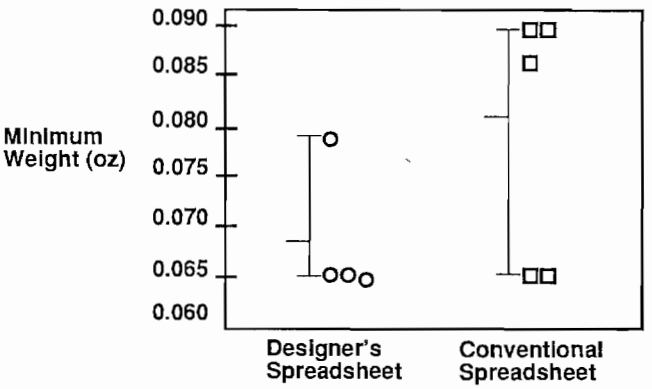


Fig. 3 Results achieved by nine student subjects trying to design a minimum weight spring (Problem 1). The bars indicate the mean and range.

Ideally each user would use both tools on the same problem and the results would then be compared. However, since the problems had to be simple enough to be amenable to solution in a short time, we felt that having tried to solve the problem once would have a significant effect on the quality of future solution efforts. Hence, separate problems were used for each tool. We were interested in trying to include some variety in the test problems. However, even with nine participants, testing on more than two problems was impractical at the time.

The experimental protocol used was as follows:

(1) When the subject arrived, the researcher explained the interface of the tool for 15 minutes. This explanation used a simple problem as an example and the subject was encouraged to try out the tool to solve the sample problem.

(2) The subject was presented a sheet containing the problem description. The equations for the problem were already input to the tool that would be used for the test. When the subjects felt they understood the problem, they would begin solving the problem. The total time allowed was 20 minutes, 10 minutes for reading and understanding the problem statement and 10 minutes for solving the problem. If subjects felt that they had reached the best solution they could, they had the option to stop.

(3) Step 2 was then repeated using the second tool and problem.

A questionnaire regarding major and computer skills filled out by each subject at the conclusion of the experiment indicated that there were no obvious biasing factors in the sample population. The tests were run on a locally available computer workstation that all the participants had used, though to varying degrees.

Both the design problems were drawn from actual industrial projects, though the actual numbers used were changed to protect proprietary information. Brief descriptions of the problems used are as follows:

- Problem 1 dealt with the design of a coil spring used to provide a clamping force in an indexing mechanism. A number of constraints on force, compressed length and geometry were specified. The goals in this problem were first to achieve any design that satisfied all the constraints and then to find the feasible design that minimized the weight of the spring.
- Problem 2 was related to a battery-operated fastening tool. The goals of the problem were to find the design that allowed the tool to install the largest number of fasteners and then to find the lightest design that allowed the tool to install at least a specified number of fasteners.

Figures 3 and 4 show some of the results achieved by the subjects. A visual check of Fig. 3 indicates a noticeable difference in the mean performance of the test subjects. To test the likelihood that the differences in design performance that we observed were just due to chance, the small sample "t" test and the Wilcoxon rank-sum test (McClave and Dietrich, 1985) were applied to the data. The number α is the probability that a difference in the mean performance at least as great as the one we observed would arise if the samples were drawn from the same underlying population. The α values for this test are shown below.

Problem no.	t test	Wilcoxon test
1	$\alpha = 0.1$	$\alpha = 0.1$
2	$\alpha = 0.1$	$\alpha = 0.1$

The sample size is small and an α of 0.1 is not a definitive result, but the results suggest that the designer's spreadsheet may enhance designer performance.

One of our concerns with the testing was that the differences in design quality could simply have been due to the fact that different numbers of operations were performed by the test subjects. Even though text subjects were allotted the same time to solve each problem using both tools, we were concerned that the differences in performance arose simply because of the differences in the time required to try a design solution. To see if this was an issue we examined the number of solutions tried by users when achieving their final design.

To account for the differences in the user interface of these tools, an operation was defined as any contiguous sequence of changes made to a variable. For example, changing the value of a variable from 0 to 1 would always be read as one operation, whether the change was actually made in a single step or as 10 (consecutive) steps of 0.1 each using the buttons on the Designer's Spreadsheet.

The distribution of the number of operations performed by the test subjects is shown in Fig. 5. Figure 5 shows that using the above definition of operation there are no dramatic differences in the mean number of operations performed by subjects using different tools. This evidence, combined with previous results on solution quality, suggests that the Designer's Spreadsheet helps users achieve better solutions with no noticeable additional effort.

Another issue regarding these experiments is the effect of user interface alone on the observed results. We were concerned that the positive results observed were not due to the mathematical functionality of the tool itself but merely to the user interface. To explore this hypothesis, we tested user performance on a conventional spreadsheet with two different user interfaces.

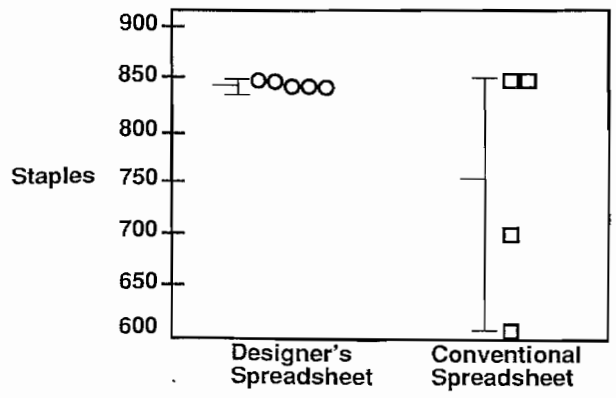


Fig. 4 Results achieved by nine student subjects trying to maximize the number of fasteners fired by a fastening tool (Problem 2)

The control case was the standard keyboard interface where the user changed spreadsheet cells by typing in new values. The other interface was one where, like in the Designer's Spreadsheet, the user changed a value using a pair of up-down buttons.

The experimental design for this test was as follows. Six subjects, S1–S6, were used. These subjects were different than the ones used in the user tests described earlier. Problems 1 and 2 are the same design problems used in the earlier experiments.

	Problem 1	Problem 2
Conventional interface	*S1*, *S3*, *S5*	*S2*, *S4*, *S6*
Modified interface	*S2*, *S4*, *S6*	*S1*, *S3*, *S5*

The data collected from these tests is shown in Fig. 6. The figure shows that on the average, both the quality of the eventual design and the number of operations required to reach a final design are different, depending on the type of interface. A t-test was used to determine if these differences were statistically significant. This test showed that it was likely that differences at least as large as we observe in this test would be observed even if the samples were drawn from the same population. We treat this as preliminary evidence that design performance is not significantly affected by use of either of the interfaces tested in this experiment.

Finally, we performed some informal tests with practicing designers from industry. The users were chosen from local companies in the Cambridge area. All the testing was performed at MIT since differences in hardware and software made it hard to move the test code.

The protocol followed was as follows:

(1) All the subjects had been selected because they were using or had used conventional spreadsheets in their work. Hence, they were introduced directly to the idea of the Designer's Spreadsheet and given a brief description of the tool.
(2) The subjects then worked on the fastening tool design problem described earlier.
(3) The test problem was followed by an open-ended interview in which there was a general discussion about merits and demerits of the tool.
(4) Finally, a survey form was filled out. The results of this survey are summarized in Fig. 7.

The results of the testing with industrial users were also quite encouraging. Most of the subjects found the tool intuitive and easy to use. One specific interesting result was the perception of the Designer's Spreadsheet when compared to existing math programming solvers. All the subjects felt that the functionality provided by the Designer's Spreadsheet would be useful despite the availability of solvers. The primary reason stated for this was that the various intermediate states that the subjects saw when using the Designer's Spreadsheet were perceived as being useful in understanding the design solution better.

5 Conclusion

We believe that we have developed a superior design tool for preliminary engineering design that subsumes the capabilities of conventional spreadsheet programs. A simple and practical theoretical basis for the tool, both numerical and symbolic, has been developed. To verify the usefulness of this tool we have tested the Designer's Spreadsheet on a variety of practical design problems and have run tests with a variety of student and industrial users. These are described in detail in the reference (Ramaswamy, 1993). The findings from these and other tests performed suggest that the Designer's Spreadsheet is an intuitive and easy-to-use tool and provides more support for design problem solving than conventional spreadsheets.

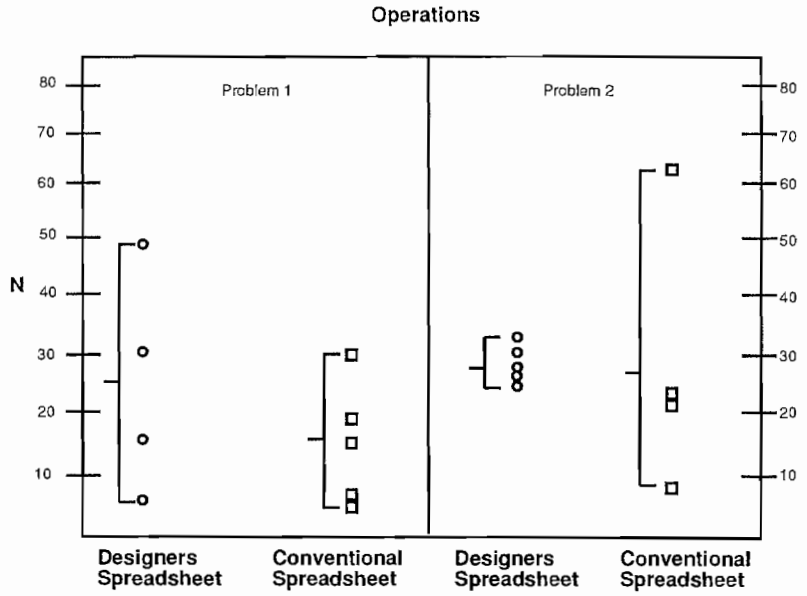


Fig. 5 Number of operations performed while solving Problems 1 and 2

To conclude we would like to reiterate some important points regarding this approach:

(1) Basic contribution. This approach does not claim to solve any of the classical problems of mathematical programming. The basic mathematics underlying both the symbolic and numeric approaches is not claimed to be new. The contribution of this work is the synthesis of existing knowledge to provide a system that the user can interact with in real-time. We believe that the Designer's Spreadsheet provides a designer with increased control and freedom to explore the design space in ways that were simply not possible with a conventional spreadsheet.

(2) Inherent limitations of the tool. A basic limitation of the approach is that it is path-dependent. When the system is performing differential constraint maintenance, it is only possible to follow continuous paths in the design space. In certain situations, such as when a symbolic manipulation approach is successful, this limitation can be overcome but for the most part it applies. As a result, this tool is billed as an exploration tool and not as an optimization tool. Optimization, which involves following paths, backtracking and so on, is still best left to the automated programs.

(3) User interfaces. The user interface often overwhelms the basic power of the tool in terms of determining its broad acceptance. The current spreadsheet user-interface, for example, has evolved over several years of experience and is still changed with every new release of software. This is not only because of new computer technology available today but also due to an increased understanding and appreciation of the way users use the program.

(4) Benefits over spreadsheets. We believe that simply by taking away the inherently directional nature of the spreadsheet, we have created a computer tool more suitable for use by designers. However, that improvement was achieved for certain special cases by Serrano several years ago. Our approach is, we believe, a more general way of providing an infinitely permutable spreadsheet. We have examined existing research on direct manipulation interfaces and pulled out the bare minimum that is necessary to implement the Designer's Spreadsheet. As a result of this shift in thinking, we are able to save a lot of computation and hence are not subject to serious performance problems.

(5) Implementation issues. The current implementation of the system is in the *C* programming language and uses the Motif toolkit for the GUI. All the examples described in this paper including the user tests were run on a Sparcstation 2. As the vast majority of practicing engineers currently use IBM-compatible PCs, we believe that it is necessary to reimplement the system on

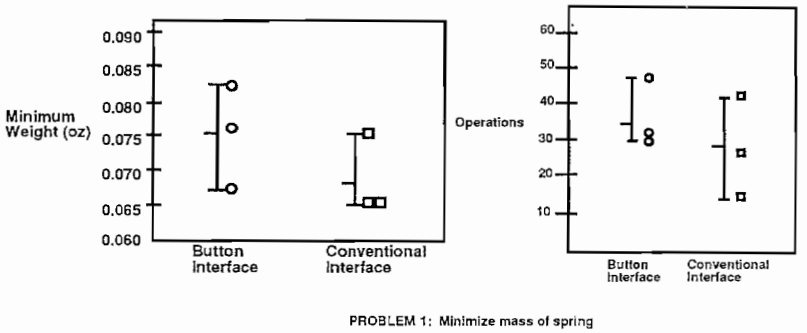


Fig. 6 Results achieved by designers using different interfaces to a conventional spreadsheet

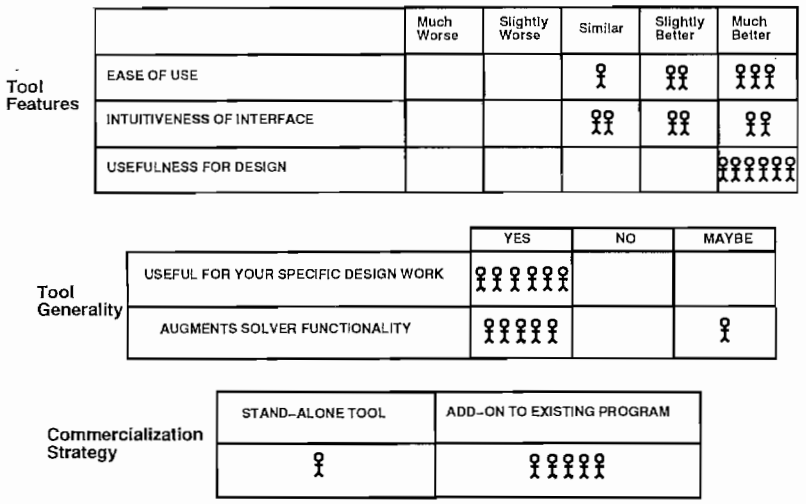


Fig. 7 Ratings for designer's spreadsheet vs conventional spreadsheet from six industrial subjects

these machines. Other desirable attributes of the next implementation of the system include,

- Ability to view and manipulate the equation network in graphical form.
- An object-oriented implementation on both the graphics and the underlying mathematical engine.
- Symbolic manipulation capability that enables us to parse typed equation input and also to avoid finite differencing to the extent possible.
- Addition of some of the advanced user-interface features such as graphing, table generation and so on.
- Integration at the higher end with a more powerful optimization program such as GAMS.
- Support for discrete variables.

Finally, we acknowledge that this work focuses on a very specific problem: how to improve the facility with which designers manipulate parametric models of modest size. Several broader issues associated with these problems have not been addressed in the current work.

6 Acknowledgments

We are grateful to Daniel Whitney, Steven Graves and Warren Seering for their comments on this research. We thank the anonymous reviewers for their helpful suggestions. Funding for this research was provided by the MIT Leaders for Manufacturing Program, the MIT New Products Program, the Defense Advanced Research Projects Agency, and the National Science Foundation.

References

Dobson, W. G., and Wolff, A. K., 1984, *Engineering Problem Solving With Spreadsheet Programs,* American Society for Metals.

Gleicher, M., and Witkin, A., "Differential Manipulation," *Graphics Interface '91,* June 1991.

Gonda, M., Fertig, K. W., and Teeter, R. J., "Aerospace Conceptual Design Using An Intelligent Design and Analysis Environment: Design Sheet," *AIAA 1992 Aircraft Design Systems Meeting,* 1992.

Kral, I. H., *The Excel Spreadsheet for Engineers and Scientists,* Prentice Hall, 1992.

Lai, K-L., Malone, T., and Yu, K-C., "Object Lens: A Spreadsheet for Cooperative Work," *ACM Transactions on Office Information Systems,* 1988.

Light, R. A., "Symbolic Dimensioning in Computer-Aided Design," Master's thesis, M.I.T., 1980.

McClave, J. T., and Dietrich, F. H., *Statistics,* Dellen Publishing Company, 1985.

Milley, M. K., "An Interactive Computer Aid for Design Parameter and Constraint Tradeoff Analysis." Master's thesis, Massachusetts Institute of Technology, 1972.

Papalambros, P. Y., and Wilde, D. J., *Principles of Optimal Design: Modeling and Computation,* Cambridge University Press, Cambridge, England, 1988.

Piela, P. C., "ASCEND: An Object Oriented Computer Environment for Modeling and Analysis," PhD thesis, Carnegie Mellon University, 1989.

Ramaswamy, R., Ulrich, K. T., Kishi, N., and Tomikashi, M., "Solving Parametric Design Problems Requiring Configuration Choices," ASME JOURNAL OF MECHANICAL DESIGN, Vol. 115, No. 1, 1993.

Ramaswamy, R., "Computer Tools for Preliminary Parametric Design," PhD Thesis, Department of Mechanical Engineering, MIT, June 1993.

Serrano, D., "Constraint Management in Conceptual Design," PhD thesis, Massachusetts Institute of Technology, October 1987.

Slocum, A. H., *Precision Machine Design,* Prentice Hall, 1992.

Strang, G., *Linear Algebra and Its Applications,* Harcourt Brace Jovanovich, 1988.

Tidd, W. F., Rinderle, J. R., and Witkin, A., "Design Refinement via Interactive Manipulation of Design Parameters and Behaviors," *Proceedings of the ASME Design Theory and Methodology Conference,* 1992.

Witkin, A., Gleicher, M., and Welch, W., "Interactive Dynamics," Computer Graphics, Vol. 24, No. 2, pp. 11–21, March 1990, *Proceedings 1990 Symposium on Interactive 3D Graphics.*