Le passé, le présent et l'avenir des logiciels intelligents de programmation mathématique*

(* Cet article paraît simultanément dans le numéro du printemps du bulletin d'INFORMS)

JOHN W. CHINNECK

Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6
Canada
chinneck@sce.carleton.ca
http://www.sce.carleton.ca/faculty/chinneck.html

HARVEY J. GREENBERG

Mathematics Department
University Colorado à Denver
Denver, Colorado 80217-3364
États-Unis
hgreenbe@carbon.cudenver.edu
http://www.cudenver.edu/~hgreenbe/

Introduction

Au cours des dernières années, deux courants majeurs ont profondément bouleversé la pratique de la programmation mathématique : la multiplication de plates-formes de calcul puissantes et peu coûteuses et l'amélioration marquée des algorithmes de résolution. Ces développements permettent dorénavant aux programmeurs mathématiques expérimentés de résoudre des modèles d'une portée et d'une complexité beaucoup plus grandes. Mais en même temps, ils ont amené beaucoup de novices à la programmation mathématique, notamment par le biais des solutionneurs intégrés aux populaires tableurs.

Que l'on ait affaire à un expert ou à un débutant, souvent la difficulté du processus de programmation mathématique ne réside pas dans le calcul numérique d'une solution. Les opérations critiques sont plutôt l'élaboration, l'analyse, la compréhension et la communication des modèles et des instances qui permettront d'établir clairement les implications en matière d'aide à la décision. Il est souhaitable, voire nécessaire, de mettre au point une certaine forme d'assistance automatisée pour surmonter la portée et la complexité des programmes mathématiques modernes. Cette prise de conscience a donné lieu, au cours des deux dernières décennies, à la mise sur pied d'une fondation pour la conception d'un *système intelligent de programmation mathématique* (IMPS) [9].

Sur un plan formel, on définit un IMPS comme un logiciel qui réduit la complexité inhérente au *processus de programmation mathématique*. On le décrit de manière encore plus précise dans la section ci-après : il s'agit d'une série de tâches qu'il faut effectuer pour bâtir, utiliser et maintenir un système d'aide à la décision s'appuyant sur un programme mathématique. Notre définition d'un IMPS est un peu plus large que la définition habituelle de l'intelligence artificielle puisque nous y incluons des outils qui ne font pas nécessairement appel au raisonnement; l'intégration d'outils de visualisation en est un exemple éloquent.

La mise au point d'outils logiciels conçus pour assister le processus de programmation intelligente est un champ de recherche très actif depuis de nombreuses années, surtout ces derniers temps. Une bibliographie récente [13] recense plus de 500 références traitant d'un quelconque aspect d'un IMPS, dont plus de 80 portent spécifiquement sur les logiciels, pour la période allant de 1953 à 1996.

Dans le présent article, nous présenterons un résumé des tâches que peut accomplir un IMPS idéal, à partir des travaux menés au cours des vingt dernières années et nous commenterons aussi brièvement l'état actuel des choses. Nous sommes actuellement à compilation les résultats d'une étude sur les logiciels IMPS existants– si nous n'avons pas encore communiqué avec vous, nous vous invitons à nous fournir la description de tout logiciel qui n'aurait pas été porté à notre connaissance (se reporter à

l'Annexe). Bref, il existe un nombre considérable de logiciels IMPS, mais il y a encore plusieurs aspects du processus de modélisation pour lesquels nous ne disposons pas d'outils appropriés. Voilà donc un domaine de recherche prometteur.

1. Le processus de programmation mathématique

La Figure 1 présente un diagramme simplifié des principaux éléments du processus de programmation mathématique. On extrait un *modèle mathématique* du *domaine du problème* en faisant des approximations appropriées des relations (objectifs et contraintes). Une fois les données recueillies, on peut formuler et résoudre diverses *instances* du modèle, qui varient selon les données spécifiques utilisées. On peut ensuite regrouper les résultats du solutionneur pour les différentes instances sous la forme d'*études de cas*. On compare alors les résultats de l'étude de cas au domaine du problème. Tout résultat imprévu ou erroné, ou toute autre question soulevée par les résultats des études de cas, peut conduire à des modifications du modèle mathématique ou des données; le cycle de modélisation et d'analyse se répète.

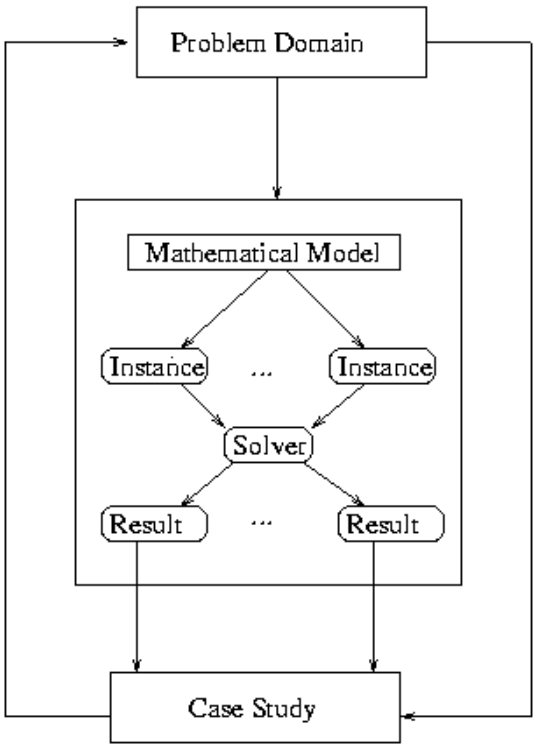

Figure 1. **Vue simplifiée du processus de programmation mathématique**

Le processus présenté à la Figure 1 est idéalisé. Beaucoup de travaux de modélisation ne sont ni exclusivement de type descendant (c'est-à-dire procédant depuis les structures du modèle et progressant jusqu'à l'acquisition des données) ni ascendant (prenant comme point de départ une riche base de données dont on déduit les relations appropriées). Le processus se trouve souvent à mi-chemin entre les deux ou passe de l'un à l'autre.

On peut considérer le processus de programmation mathématique comme un cycle de fonctions appliquées aux éléments de la Figure 1. Vous trouverez ci-après une description des sept fonctions principales du processus qui sont présentes dans tout projet de modélisation. L'aide automatisée peut se révéler utile pour toutes ces fonctions.

Expression du modèle. Il est primordial, surtout dans le cas de modèles vastes et complexes, de disposer d'un moyen pratique d'exprimer le modèle, afin de pouvoir détecter les erreurs et les omissions. Certains langages de modélisation algébriques et interfaces graphiques entrent dans cette catégorie. Il existe deux fonctions connexes : la *documentation*, ou enregistrement d'une description pour faire connaître le modèle aux autres, et la *vérification*, qui consiste à vérifier que ce que vous croyez être dans le modèle résident est bel et bien là.

Représentation du modèle et des instances. On peut représenter le modèle et ses instances de diverses manières : représentations algébriques, schématiques, graphiques et langage naturel. Certains modélisateurs trouveront que certaines vues sont plus naturelles et qu'elles permettent de comprendre plus facilement le modèle ou la solution. D'autres pourront souhaiter faire appel à plusieurs représentations pour arriver à une meilleure compréhension du modèle. Un IMPS doit donc pouvoir rendre compte d'un large éventail de *différences cognitives*. La *présentation des résultats* est une fonction connexe qui peut inclure l'interrogation interactive, la production de fichiers internes pour une analyse ultérieure ou la création de transparents pour rétroprojecteur.

Simplification du modèle. La simplification automatique des modèles visant à améliorer la performance du solutionneur est bien établie, par exemple dans les routines de pré-résolution qu'on retrouvent dans les solutionneurs de programmation linéaire de qualité commerciale. Mais peut-être est-il plus important de faire appel à la simplification pour améliorer la compréhension humaine, par exemple avec la réécriture automatique du code source du langage de modélisation qui permet d'obtenir une expression plus simple du modèle et de trouver un réseau sous-jacent.

Débogage. Il s'agit du processus qui consiste à éliminer les erreurs mécaniques dans une instance et, parfois, dans le modèle lui-même. La correction de modèles qui sont justes d'un point de vue mécanique, mais qui constituent des vues erronées de la réalité, s'effectue à l'aide des outils d'analyse générale, chacun étant pris séparément. Divers types d'erreurs sont courants dans les programmes mathématiques, entraînant parfois une non-réalisabilité, une non-limitabilité et une non-viabilité. On peut déboguer le modèle en procédant à une mise au point des instances du modèle ou en travaillant directement au niveau symbolique.

Gestion des données. Dans bon nombre de processus de programmation mathématique, on recueille et on génère un large volume de données. Des techniques spécifiques aux bases de données sont fréquemment utilisées pour gérer l'information et elles pourraient servir à l'avenir à l'application de technologies connexes comme l'exploration de données. Nos observations sur cette fonction ne sont pas exhaustives; le sujet est vaste et mérite une attention particulière.

Gestion des scénarios. En règle générale, on résout plusieurs instances d'un modèle, chacune

représentant un *scénario*. L'*étude de cas* est une collection de scénarios connexes, et certains cas peuvent se recouper (c'est-à-dire reprendre en partie les mêmes scénarios). La grande quantité d'information produite de cette manière doit être traitée et filtrée afin que l'analyste puisse facilement en tirer des conclusions déterminantes.

Analyse générale. Il existe plusieurs questions générales se rapportant à tout modèle mathématique. Par exemple : Existe-t-il des relations particulières entre les données? Quelles sont les formes fonctionnelles non linéaires? Qu'est-ce qui détermine le prix d'une variable donnée? Les outils d'analyse générale permettent de répondre à de telles questions en offrant les moyens de sonder le modèle de diverses manières. La *validation*, c'est-à-dire l'étape qui consiste à déterminer dans quelle mesure le modèle réussit à représenter la réalité, est une fonction connexe. Par exemple, le modèle de programmation linéaire est-il une représentation adéquate du système, compte tenu des décisions pour lesquelles il a été conçu? Enfin, examinons l'*analyse de redondance*: la contrainte est-elle redondante parce que d'autres contraintes sont plus restrictives ou pour des motifs économiques?

L'environnement entourant le projet de modélisation peut ajouter à la complexité du modèle et ainsi rendre d'autant plus nécessaire l'assistance automatisée. Ainsi, un modèle peut être éclectique parce que les divers éléments du modèle relèvent de personnes différentes (p. ex. le National Energy Modeling System [20]). Il arrive aussi qu'un modèle soit mis en application ou modifié par d'autres personnes que celles qui l'ont élaboré. Dans l'un ou l'autre de ces cas, les assistants intelligents se révèlent indispensables.

2. La contribution des logiciels d'hier, d'aujourd'hui et de demain

Quand on examine les logiciels IMPS créés dans le passé, on constate que le plus grand défi inhérent à la conception de systèmes durables consiste à ne pas se laisser dépasser par les changements technologiques rapides dans le marché des logiciels. Jusqu'à maintenant, notre étude indique que beaucoup de systèmes ne sont plus utilisables parce qu'au moment de leur mise en place, on les avaient associés à un système de développement qui n'existe plus ou qui nécessiterait une maintenance considérable pour arriver à être au niveau de la plate-forme hôte. Les systèmes conçus pour fonctionner avec plusieurs plate-formes réussissent généralement à durer, à condition qu'ils continuent à offrir des fonctionnalités concurrentielles.

Le format et les fonctionnalités des futurs IMPS sont inextricablement liés aux développements informatiques, tant matériels que logiciels. La conception d'interfaces utilisateur toujours plus rapides et plus performantes offre de superbes avenues de recherche; en effet, une meilleure exploitation de cette technologie permettra d'améliorer les IMPS. La croissance phénoménale du Web a déjà un impact sur l'enseignement de la programmation mathématique; par exemple, on a maintenant accès à des codes Java qui permettent un calcul interactif en temps réel. On peut s'attendre à voir dans les années à venir des chercheurs travailler collectivement, par l'entremise du Web, au développement et à la résolution d'immenses modèles. Or, pour faciliter la gestion de ce processus, il faudra créer des assistants intelligents.

Nous expliquerons maintenant brièvement, à l'aide d'exemples précis, comment les logiciels assistent actuellement les fonctions du processus de programmation mathématique.

Expression et représentation des modèles. Il existe divers modes d'expression des programmes mathématiques complexes, notamment les langages de modélisation algébriques, les langages de programmation logique par contraintes, les interfaces graphiques et les tableurs. Pour un bref survol, consultez [15]

L'expression d'un modèle doit être une description complète comportant suffisamment de détails pertinents pour permettre l'aide à la décision. Les vues incluent toute représentation qui peut fournir une explication utile, même si cette vue élimine ou omet certains détails. L'expression et la représentation du modèle font partie du volet *discours* [12] d'un IMPS (comment l'ordinateur et nous communiquons l'information relative au modèle et à ses instances).

Dans le discours, on peut employer différents styles de langage, par exemple algébrique ou procédural. Les *langages de modélisation en réseau* peuvent utiliser une interface utilisateur graphique ou constituer une variante spécialisée des langages de modélisation algébriques (p. ex. Proflow [5]). Les langages de *programmation logique par contraintes* offrent aussi un autre moyen d'exprimer et de représenter les relations logiques. Certains systèmes peuvent fournir un discours en langage plus naturel pour appuyer l'analyse du modèle.

Parmi les expressions tabulaires, on trouve le *schéma fonctionnel*, qui représente le modèle au moyen d'une syntaxe de cellules particulière [1,15,18]. On utilise aussi les tableaux pour visualiser des données comportant différentes *tranches* et *agrégations*, éléments courants dans la majorité des langages algébriques. On peut combiner certains de ces éléments pour construire des vues du processus [1]. Les *tableurs* sont une forme particulière de discours tabulaire, où les cellules fusionnent logique de modèle et données. Les *graphiques* peuvent se présenter sous une forme iconique ou schématique. Les *matrices* peuvent fournir des vues utiles, par exemple on peut afficher une très grande matrice qui utilise un seul pixel allumé pour toute cellule contenant une entrée différente de zéro ou indiquer les entrées positives par un signe + et les entrées négatives par un signe –. Vous trouverez dans le texte de Greenberg et Murphy [14] une analyse générale des structures et une étude des systèmes dans lesquels on se sert de graphiques et de matrices.

Certaines vues ne sont pas courantes en programmation mathématique, mais proviennent plutôt des concepts de bases de données, tels que les *relations de dépendance* – dont les fonctions et les variables dépendent d'un ensemble donné. Le système MODLER [11] fournit des vues de ce type.

Dans un IMPS idéal, le modélisateur devrait avoir le choix entre diverses modes d'expression du modèle et devrait pouvoir passer aisément d'une vue à l'autre au besoin. La grande majorité des systèmes offrent une seule façon d'exprimer le modèle et. habituellement, aucune autre vue. MIMI [1] est peut-être le système commercial le plus évolué car il donne accès à plusieurs modes d'expression et de représentation du modèle. MProbe [7] offre aussi diverses vues tabulaires, ainsi que des courbes du profil des fonctions non linéaires de nombreuses dimensions et une vue algébrique grâce à un lien à AMPL. MODLER [11] fournit de nombreuses vues, mais requiert une notation algébrique; ANALYZE [10] offre la possibilité de choisir parmi diverses vues tabulaires, graphiques en réseau, matricielles et algébriques.

Il devient maintenant impératif de faire de la recherche fondamentale sur des vues qui permettront de représenter d'autres formes de modèles que les modèles linéaires. L'animation est un domaine de recherche prometteur; voyez à ce sujet le travail de pionnier de Jones [17].

Simplification des modèles. La majorité des solutionneurs commerciaux et certains langages de modélisation incluent une fonctionnalité de pré-résolution, mais ils sont conçus comme des " boîtes noires " visant à accélérer l'optimisation, non comme des outils d'analyse. Il existe peu d'outils qui offrent des simplifications du modèle en vue de faciliter la compréhension humaine. MProbe [7] permet d'examiner les fonctions non linéaires de nombreuses dimensions pour déterminer si on peut les simplifier (par exemple, en les linéarisant). ANALYZE [10] permet d'explorer le modèle pour effectuer des simplifications en programmation linéaire ou en programmation linéaire en numération mixte. GAMSCHK [19] lit les entrées en langage GAMS et fait un rapport sur des façons de simplifier le modèle. On peut aussi utiliser l'analyse de redondance au moyen d'un outil d'analyse générale comme ANALYZE, en dépit du fait que très peu de praticiens comprennent comment et pourquoi cela fonctionne.

Débogage des modèles. Les logiciels traditionnels permettent la détection d'états non optimaux, mais non d'erreurs sous-jacentes. Les logiciels modernes aident à en trouver la cause et proposent des solutions de réparation et de prévention.

Le débogage associé à la *non-réalisabilité* existe depuis longtemps déjà. Des méthodes de resserrement successif des bornes sont incluses dans les routines de pré-résolution en PL depuis de nombreuses années et elles permettent parfois de détecter la non-réalisabilité et de fournir des renseignements utiles grâce à la revue de la séquence des réductions (voir, à titre d'exemple, la commande REDUCE dans ANALYZE). Cependant, au cours des dernières années, la plupart des principaux solutionneurs commerciaux de PL ont opté pour l'isolation d'un sous-système non réalisable irréductible (*Irreducible Infeasible Subsystem* (IIS)) [4,6,0]. Le IIS correspond souvent à une petite fraction des contraintes du modèle et il a la propriété d'être non réalisable, alors que tout sous-ensemble propre est réalisable; ainsi, chaque contrainte du IIS contribue à cette non-réalisabilité. Trouver un IIS permet d'orienter le diagnostic. Une récente étude de Chinneck [6] traite des derniers développements concernant l'analyse de la non-réalisabilité pour toutes les formes de programmes mathématiques.

En théorie, on peut procéder au débogage dans un cas de *non-limitabilité* en appliquant la méthode de débogage utilisée pour la non-réalisabilité (du moins en PL et avec certaines restrictions en PNL et en programmation en numération mixte), mais certains exemples démontrent que cette méthode n'est pas efficace. La non-limitabilité peut être imputable à divers facteurs, par exemple un cycle négatif dans un modèle en réseau. Il est parfois préférable de rechercher ces causes directement, plutôt que de s'en remettre à une technique conçue pour une non-réalisabilité primale.

La non-viabilité est une propriété d'un modèle en réseau dans lequel au moins une variable de flot est forcée à zéro par la structure du modèle en réseau (non par les bornes supérieures des flots); cela peut se produire dans toutes les formes de modèle en réseau, y compris les *réseaux de traitement* [2]. Étant donné qu'il est improbable que le modélisateur conçoive un arc qui ne peut pas supporter le flot, une telle erreur mécanique devrait être signalée avant la résolution du modèle. Chinneck [2,3] a mis au point des méthodes pour identifier et isoler la non-viabilité et ces méthodes ont été intégrées à des prototypes de logiciels réalisés dans le cadre universitaire. Il est intéressant de noter que tout programme linéaire général peut être considéré comme un réseau de processus en raison de la structure d'entrée/sortie de ses opérations [14].

Une généralisation de la non-viabilité s'applique à toute variable qui est forcée à une valeur fixe par les contraintes du modèle. Compte tenu de cette interprétation, il reste maintenant à développer des méthodes et des logiciels pour toutes les formes de programmes mathématiques qui ne sont pas en réseau.

Analyse générale. Les logiciels d'analyse générale doivent proposer un ensemble d'outils pour explorer le modèle de diverses manières afin de répondre aux questions et de fournir des explications. Examinons deux outils créés par les auteurs de cet article.

ANALYZE fournit une assistance informatisée pour analyser les programmes mathématiques linéaires et leurs solutions. Il comporte trois niveaux d'utilisation. Le premier niveau permet une interrogation interactive pratique et, combiné à MODLER, il offre une syntaxe suffisamment riche pour permettre la description de rapports désirés par l'usager [8]. Le second niveau inclut des procédures qui facilitent l'analyse de diverses façons. Il permet de répondre à des questions courantes, telles que *Et si …?, Pourquoi …?* et *Pourquoi ne pas …?*, par un accès facile à l'information concernant la solution. En outre, on peut résoudre efficacement une analyse diagnostique, par exemple le débogage d'une non-réalisabilité, à l'aide de routines internes ou de liens à de l'information externe, tel qu'un IIS obtenu au moyen de MINOS(IIS) [4]. Le troisième niveau est un environnement fondé sur l'intelligence artificielle qui inclut des traductions d'objets en anglais et un système de déduction à base de règles.

MProbe fournit des estimations de la forme d'une fonction non linéaire (p. ex. convexe, concave ou les deux), de l'étendue de la forme, de la fourchette de valeurs de la fonction, de la pente, la courbe d'une fonction entre deux points arbitraires dans un espace à n dimensions, la navigation du modèle et des estimations de l'efficacité des contraintes. Il propose aussi des estimations de la forme d'un domaine contraint (convexe ou non) et permet d'établir la probabilité que l'on puisse trouver un optimum global (en fonction de la forme et du sens de la fonction objectif et de la forme du domaine contraint). On peut avoir recours au tri personnalisé pour isoler les classes de contraintes susceptibles de poser des difficultés pour la programmation linéaire à numération mixte.

Gestion des données. De nombreux langages de modélisation permettent une connexion directe du modèle mathématique avec des formats courants ou propriétaires de bases de données. L'accès à de grandes bases de données est depuis toujours un aspect important des systèmes de programmation mathématique. L'interfaçage avec des tableurs est un développement plus récent qui s'est révélé tout aussi important, quoique destiné à d'autres utilisateurs.

Gestion des scénarios. Depuis des générations, cette tâche était effectuée par les gestionnaires de modèles qui utilisaient des tables de contrôle d'une manière qui rendait le processus de gestion très difficile. MathPro [18] est un bon exemple d'un système conçu spécifiquement pour faciliter la gestion des scénarios. L'analyse inter-scénarios est actuellement une tâche pour laquelle il existe peu d'outils informatiques. Mais comment utiliser avantageusement l'information provenant d'un ensemble de scénarios quand on résout (ou qu'on précise) un nouveau scénario? Le IMPS doit fournir des outils pour permettre une telle analyse. Cet outil pourrait être aussi simple qu'un lien vers un autre module, comme un progiciel statistique.

3. Conclusions

Le processus de programmation mathématique peut être assez complexe et il en train de le devenir encore davantage avec l'évolution du matériel informatique et des logiciels et l'amélioration des algorithmes. L'époque actuelle rappelle les premiers jours de la programmation informatique générale, quand le programmeur disposait de très peu d'outils et qu'il devait utiliser un simple éditeur de texte pour créer des programmes et se plonger dans des analyses-mémoire pour corriger les erreurs. Aujourd'hui le programmeur dispose d'une foule d'assistants : les éditeurs de codes qui mettent en surbrillance les mots réservés, qui signalent les erreurs au moment où elles sont tapées, sautent aux définitions des variables et indiquent les hiérarchies d'appel, la documentation en ligne,

les débogueurs, les gestionnaires de profils, etc. La programmation mathématique commence à peine à créer ses propres outils spécialisés pour assister l'utilisateur dans le processus visant à créer, à explorer et à maintenir des modèles utiles. En fait, le principal obstacle à l'efficacité de la programmation mathématique n'est pas la résolution des instances des modèles, mais notre capacité à venir à bout de sa complexité inhérente. Les systèmes intelligents de programmation mathématique visent à surmonter cet obstacle.

ANNEXE: PARTICIPEZ À NOTRE ÉTUDE SUR LES LOGICIELS

Si vous connaissez des logiciels qui présentent certains aspects d'un système IMPS, veuillez communiquer avec l'un des auteurs et fournir les renseignements suivants :

- Nom du logiciel.
- Nom et adresse des développeurs ou personnes-ressources (y compris l'adresse de courrier électronique).
- Sources d'information traitant du logiciel : articles publiés, rapports techniques, livres, sites Web, etc.
- Commentaires sur la manière dont le logiciel contribue à chacune des sept fonctions du processus de programmation mathématique.

(La bibliographie se trouve à la page 26)

References

1. Chesapeake Decision Sciences, *MIMI User Manual*, New Providence, NJ, 1988.
2. J.W. Chinneck, Formulating Processing Networks: Viability Theory, *Naval Research Logistics* 37 (1990), 245-261.
3. J.W. Chinneck, Viability Analysis: A Formulation Aid for All Classes of Network Models*, Naval Research Logistics* 39 (1992), 531-543.
4. J.W. Chinneck, MINOS(IIS): Infeasibility Analysis Using MINOS*, Computers & Operations Research* 21:1 (1994), 1-9. Software available at http://www.sce.carleton.ca/faculty/chinneck/minosiis.html.
5. J.W. Chinneck, Processing Network Models of Energy/Environment Systems, *Computers and Industrial Engineering* 28:1 (1995), 179-189.
6. J.W. Chinneck, Chapter 14: Feasibility and Viability, in *Advances in Sensitivity Analysis and Parametric Programming*, T. Gal and H.J. Greenberg (eds.), International Series in Operations Research and Management Science, vol. 6, Kluwer Academic Publishers, Boston, MA, 1997.
7. J.W. Chinneck, *Analyzing Mathematical Programs Using MProbe*, Technical Report SCE-98-03, Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1998. Software available at http://www.sce.carleton.ca/faculty/chinneck/MProbe.html.
8. H.J. Greenberg, Syntax-Directed Report Writing in Linear Programming*, European Journal of Operations Research* 72:2 (1994), 300-311.
9. H.J. Greenberg, An Industrial Consortium to Sponsor the Development of an Intelligent Mathematical Programming System, *Interfaces* 20:6 (1991), 88-93.
10. H.J. Greenberg, A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE, Kluwer Academic Press, Boston, MA, 1993. DOS and linux versions are available at http://www.cudenver.edu/~hgreenbe/imps/software.html.
11. H.J. Greenberg*, Modeling by Object-Driven Linear Elemental Relations: A User's Guide for MODLER*, Kluwer Academic Press, Boston, MA, 1993. DOS and linux versions are available at http://www.cudenver.edu/~hgreenbe/imps/software.html.
12. H.J. Greenberg, Intelligent User Interfaces for Mathematical Programming, Proceedings of the Shell Conference: *Logistics: Where Ends have to Meet*, C. Van Rijgn (ed.), Pergamon Press, 1989, 198-223.
13. H.J. Greenberg, A Bibliography for the Development of an Intelligent Mathematical Programming System, *Annals of Operations Research* 65 (1996), 55-90. A 1997 version is on the World Wide Web, http://www.cudenver.edu/~hgreenbe/imps/impsbib/impsbib.html.

14. H.J. Greenberg and F.H. Murphy, Views of Mathematical Programming Models and Their Instances, *Decision Support Systems* 13:1 (1995), 3-34.
15. H.J. Greenberg and F.H. Murphy, A Comparison of Mathematical Programming Modeling Systems, *Annals of Operations Research* 38 (1992), 177-238.
16. H.J. Greenberg and F.H. Murphy, Approaches to Diagnosing Infeasibility for Linear Programs, *ORSA Journal on Computing* 3:3 (1991), 253-261.
17. C.V. Jones, *Visualization in Optimization*, Kluwer Academic Press, Boston, MA, 1995.
18. MathPro, Inc., *MathPro Usage Guide: Introduction and Reference*, Washington, DC, 1989.
19. B.A. McCarl, *GAMSCHK User Documentation*, Department of Agricultural Economics, Texas A&M University, College Station, TX, 1998. Software available at http://agrinet.tamu.edu/mccarl/.
20. *National Energy Modeling System Integrating Module Documentation Report*, DOE/EIA-MO57(95), Energy Information Administration, Washington, DC, 1995. This, and related documents, are available at http://www.eia.doe.gov/bookshelf/docs.html.