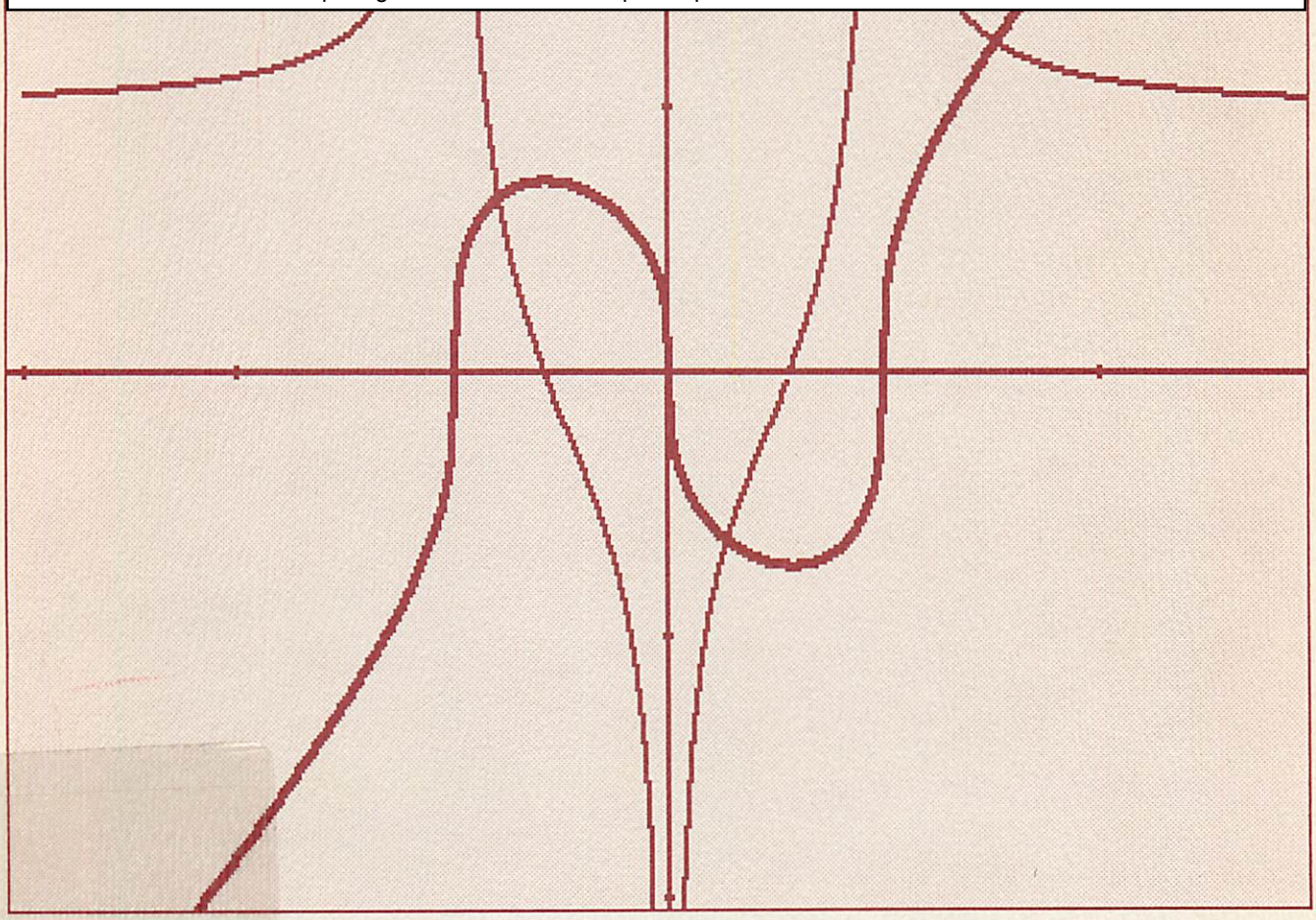
Mathématiques et programmation

Analyse de besoins et inventaire de ressources au collégial



Jacqueline T. Giard

Collège de Sherbrooke
Mathématiques

MATHEMATIQUES ET PROGRAMMATION

Analyse de besoins
et inventaire de ressources au collégial

Juin 1988

Jacqueline T. Giard
Dépt. de mathématiques

Rapport final d'un projet réalisé au Collège de Sherbrooke avec l'aide d'une subvention de la Direction générale de l'enseignement collégial, dans le cadre du Programme d'aide à la recherche et à l'apprentissage (PAREA).

On peut obtenir des copies de ce rapport
en s'adressant directement à l'auteure.
(Voir dernière page)

Les graphes sont ceux de la fonction $f(x) = \sqrt[3]{x^3 - x}$
et de sa dérivée. Ils ont été tracés au moyen du traceur
Logo reproduit à l'Appendice B.

Répondant local du projet: Guy Denis
Révision des textes: Gaétan Roy
Réalisation de la page couverture: Guy Deshaies
Dépôt légal: Bibliothèque nationale du Québec
3e trimestre 1988

ISBN 2-920916-10-6

REMERCIEMENTS

Cette recherche a bénéficié de l'expertise d'un grand nombre de personnes. On me permettra de mentionner particulièrement,

M. Gilles St-Pierre, Directeur du Programme d'aide à la recherche et à l'apprentissage de la D.G.E.C, pour le soutien constant apporté à cet effort de recherche entrepris il y a quatre ans;

Dr Gary M. Boyd, de l'Université Concordia, dont les conseils judicieux ont influencé le cours de cette recherche;

la Direction et le Service d'expérimentation du Collège de Sherbrooke, dont l'attitude et la collaboration soutenues ont contribué au cours des ans à hausser le statut de la recherche en milieu collégial;

Marie-Jane Haguel, du Dépt. de mathématiques, dont la collaboration et les conseils ont été sollicités à plusieurs reprises au cours de cette recherche;

tous mes collègues du Dépt. de mathématiques, dont Lambert Lapointe et Oliver Hayes, qui ont enrichi cette recherche du fruit de leur expérience dans l'utilisation de l'ordinateur à des fins d'enseignement et d'apprentissage;

tous les étudiants dont les travaux et les réactions ont constitué d'excellentes sources d'information pertinente;

ainsi que Renée-Marthe Giard, sans l'aide technique de laquelle la publication de ce rapport aurait tardé encore davantage...

A ces personnes donc, et à tous ceux et celles qui, par leur participation active, leur assistance, leur soutien, leur encouragement ou même simplement par leur intérêt, ont collaboré à la réalisation de ce projet et contribué à son succès, un merci sincère et chaleureux.

SOMMAIRE

Ce projet constitue le troisième volet d'une recherche portant sur les applications de l'ordinateur dans l'enseignement et l'apprentissage des mathématiques au collégial. Le premier volet, réalisé en 1984-85 avait pour but d'explorer les possibilités offertes dans ce domaine par le langage Logo. Le second volet, réalisé en 1985-86, était centré sur l'étudiant placé dans cette situation d'apprentissage. Le dernier volet étudie la question des langages de programmation utilisés pour cette approche.

L'objectif à long terme de cet ensemble de recherches était de doter le système collégial québécois d'un environnement de programmation répondant adéquatement aux besoins de la situation.

Afin de déterminer les caractéristiques essentielles et souhaitables d'un tel environnement, cette recherche a consisté à mettre en évidence les principaux objectifs de l'éducation mathématique à ce niveau, puis à évaluer la mesure dans laquelle sept logiciels différents favorisent l'atteinte de ces objectifs. Elle conclut à la nécessité d'un environnement de programmation intégrant le calcul numérique et symbolique, le graphisme, l'édition de texte, ainsi qu'un langage supportant plusieurs styles de programmation.

AVERTISSEMENT

Afin d'augmenter la lisibilité de ce texte,
les termes désignant des personnes
ont été utilisés dans leur sens générique
recouvrant les genres masculin et féminin.

TABLE DES MATIERES

INTRODUCTION

"L'expérimentation est un processus
en spirale vers l'inconnu."
Roy D. Pea

Au printemps 1986, une équipe du Département de mathématiques du Collège de Sherbrooke présentait à la DGEC, via son programme PAREA, un projet de recherche portant sur l'étude des caractéristiques d'un langage informatique approprié à l'apprentissage des mathématiques. Ce projet constituait le troisième volet d'une recherche ayant comme objectif la création d'un environnement d'apprentissage dans lequel les ressources de l'ordinateur seraient utilisées de façon optimale.

La première étape avait permis d'explorer les possibilités offertes par la programmation en langage Logo, par le biais de la construction d'activités d'apprentissage et de leur expérimentation en situation de classe. Les résultats avaient confirmé que les étudiants suivant cette démarche atteignaient un niveau de performance satisfaisant en ce qui a trait aux habiletés traditionnelles et qu'ils manifestaient des comportements actifs et orientés vers des apprentissages nouveaux. On avait également pu constater que les étudiants réagissaient positivement à cette approche qu'ils jugeaient innovatrice et intéressante, en dépit d'un niveau de difficulté non négligeable. (Giard, Haguel, 1985)

Une seconde recherche, se référant au modèle des styles d'apprentissage de Kolb (1976), devait montrer que les activités proposées dans le cadre de cette approche faisaient appel à des modes d'apprentissage (conceptualisation abstraite et expérimentation active) peu utilisés par la majorité des sujets. Les conclusions de cette recherche soulignaient également l'importance de l'individualisation de l'enseignement, ce à quoi la micro-informatique semblait ouvrir la porte. (Giard, 1986)

Au cours de ces diverses expérimentations, trois versions différentes de Logo furent utilisées. Chacune présentait des points forts et des points faibles liés à la syntaxe, à la puissance, à la précision, à l'espace-mémoire, à la rapidité, au graphisme, à l'interface usager ou à la documentation, selon le cas. Bien que la recherche, voire la construction, d'un langage universel répondant à tous les besoins soit une idée abandonnée depuis longtemps, il paraissait justifié de chercher l'outil le plus approprié à la situation d'apprentissage, outil qui permettrait à l'étudiant de concentrer son attention sur la solution de problèmes mathématiques plutôt qu'informatiques.

Ce fut là le point de départ du troisième projet qui visait, à long terme, le développement d'un environnement informatique approprié aux besoins de l'enseignement et de l'apprentissage des mathématiques dans le système collégial québécois.

Cette préoccupation au sujet des outils informatiques utilisés en milieu éducatif ne date pas d'aujourd'hui. Déjà en 1984, Sloman déplorait le manque de coordination des efforts visant à déterminer les besoins futurs et à éliminer des écoles des langages aujourd'hui dépassés: "Les mauvaises habitudes enseignées aux rares étudiants qui aiment programmer en BASIC me portent à douter de la pertinence d'introduire la micro-informatique dans les écoles."

De plus en plus, on réclame des outils performants à l'usage des étudiants. "Même en admettant qu'on ne peut pas toujours disposer des outils les plus perfectionnés, écrit Yazdani (1984), il importe que plus de gens se penchent sur cette question, développent et testent des prototypes. Nous ne devrions plus être en train de discuter des mérites des langages actuels. Nous devrions plutôt nous entendre sur les caractéristiques souhaitables dans un langage et un environnement de programmation utilisés à des fins éducatives, et développer de tels langages s'il n'en n'existe aucun qui soit jugé adéquat selon les critères retenus."

Enfin, il s'est développé depuis quelques années un important courant de recherche portant sur les facteurs humains dans la communication avec l'ordinateur. Les résultats démontrent que les caractéristiques liées à l'environnement de programmation sont aussi importantes que celles liées au langage même, et font ressortir l'importance de l'interface cognitive dans l'architecture d'un système informatisé. (Ramsey et al. 1978, Nickerson, 1986)

Le projet initial proposait de faire l'inventaire et l'évaluation des différentes versions du langage Logo dans le but de soumettre une recommandation quant à l'adoption ou l'adaptation d'un logiciel existant, ou bien la création d'un nouveau logiciel. Diverses contingences cependant firent que cet objectif fut modifié dès le début de la recherche. D'une part, les versions disponibles de Logo ne présentaient que des différences mineures, et l'une d'entre elles semblait les surpasser nettement malgré certains points faibles; de plus, d'autres langages étaient utilisés à des fins d'apprentissage des mathématiques, et il paraissait intéressant de comparer leurs caractéristiques et leurs possibilités à celles de Logo; enfin des produits nouvellement apparus sur le marché semblaient aussi mériter l'attention de cette recherche. C'est pourquoi il fut décidé d'élargir la perspective de cette recherche afin d'y inclure non pas l'étude de plusieurs versions d'un même langage, mais plutôt l'étude de plusieurs langages différents. On notera toutefois que les didacticiels et autres logiciels dits éducatifs ne sont pas inclus dans cette recherche.

Les objectifs spécifiques de ce projet consistaient à:

1. déterminer les besoins de la situation de l'enseignement et de l'apprentissage des mathématiques de niveau collégial, en termes d'environnement de programmation,
2. examiner la compatibilité de divers langages avec cette situation, et
3. exprimer des recommandations quant aux caractéristiques qu'il serait souhaitable de trouver dans un logiciel utilisé comme outil d'apprentissage des mathématiques au collégial.

Comme les ressources allouées à la réalisation de ce projet étaient beaucoup plus modestes que prévu, il avait été convenu de présenter les résultats sous forme de tableaux et de sorties sur imprimante. Mais comment présenter des résultats de recherche sans respecter un minimum de conventions?

Voici donc ce rapport qui ne devait pas être. Les sections habituellement consacrées à la problématique et à la méthodologie ayant été omises, le lecteur devra se contenter de la brève mise en situation qui précède. Dans le premier chapitre, on trouvera les résultats de l'analyse de la situation d'apprentissage (ou la réponse à la question "Qu'est-ce qui est enseigné?"). Le chapitre II présente la démarche suivie dans la sélection des logiciels retenus pour

les fins de cette recherche, ainsi que les étapes menant à la construction d'une grille (ou la réponse à la question "Qu'est-ce qu'un logiciel langage?"). Les chapitres III et IV contiennent les résultats de l'évaluation des logiciels (ou la réponse à la question "Comment ces logiciels répondent-ils aux besoins de la situation?"). La dernière partie enfin expose les conclusions et recommandations qui ont paru découler de cette évaluation.

La technologie de l'informatique évolue si rapidement qu'il est à craindre que les résultats d'une recherche en ce domaine ne soient dépassés avant même qu'ils n'aient été publiés. Afin d'éviter une telle embûche, cette recherche a voulu produire des résultats qui soient d'un intérêt plus que passager, en examinant des questions qui ne sont pas liées à des systèmes physiques, et en cherchant des principes applicables à de larges catégories d'équipements et de situations.

Voilà pourquoi le résultat le plus important de cette recherche ne réside peut-être pas dans l'évaluation de certains logiciels langages, mais plutôt dans l'analyse de contenu des cours de mathématiques de niveau post-secondaire, dans la grille d'évaluation des logiciels langages, dans les exemples suggérés pour l'utilisation de ces logiciels à des fins didactiques, et qui sait, dans un paradigme de recherche qui pourra être utilisé par d'autres éducateurs ayant à faire face au choix d'un outil informatique approprié à une situation d'apprentissage donnée.

CHAPITRE I

L'APPRENTISSAGE DES MATHEMATIQUES

Dans un article récent où il traite de programmation dans un environnement d'apprentissage des mathématiques, Grogono (1988) écrit: "Je ne prétends pas avoir trouvé quel langage devrait être utilisé pour enseigner les mathématiques, car il faudrait d'abord que je sache pourquoi on enseigne les mathématiques. Sont-elles un ensemble de trucs à utiliser plus tard? Les enseigne-t-on parce qu'elles sont une composante importante de notre culture, ou parce qu'elles facilitent le développement de la pensée logique et abstraite? Ces questions relèvent de la didactique des mathématiques. A la longue, les logiciels peuvent être ajustés à ses demandes." C'est pourquoi la première étape de cette recherche a consisté à faire l'analyse de la situation de l'enseignement et de l'apprentissage des mathématiques au collégial.

1.1 Orientation vers la résolution de problèmes

Au cours des dernières années, l'éducation mathématique à tous les niveaux s'est retrouvée presque constamment à l'avant-scène des préoccupations du milieu. D'une part, on reconnaît l'importance des mathématiques dans les études et même dans la vie de tous les jours; d'autre part, on s'inquiète des taux de réussite qui sont trop faibles, de l'importance accordée à la mémorisation et à l'application d'algorithmes au détriment de la compréhension et surtout de l'incapacité d'appliquer les connaissances mathématiques à la résolution de problèmes.

L'accent mis sur la résolution de problèmes constitue la dernière tendance en didactique des mathématiques. Théoriquement, on voudrait voir ces problèmes tirés de la vie concrète et appliqués au vécu réel des apprenants. Dans la pratique, l'enseignement se fait la plupart du temps à partir de manuels proposant des exercices et des problèmes des plus classiques. Les recherches sur cette méthode d'apprentissage sont légion et les modèles décrivant les processus impliqués dans la résolution de problèmes abondent.

La science cognitive, qui se situe à l'intersection de plusieurs disciplines dont évidemment la psychologie, propose un modèle très explicite, basé sur le traitement de l'information. Dans ce modèle, la résolution de problèmes est considérée comme un processus complexe ayant pour point de départ la lecture d'un énoncé. Ce dernier est suivi à son tour de la recherche et du recouvrement d'items gardés en mémoire et de la construction de représentations successives du problème jusqu'à une dernière représentation qui rend la solution transparente.

Selon cette interprétation, avant de résoudre un problème et afin d'y parvenir, un individu doit posséder des connaissances déclaratives reliées au problème et conservées en mémoire sous une forme quelconque.

L'existence de ces structures mentales de représentation de connaissances est un des postulats fondamentaux de la science cognitive. Leur présence est facilement mise en évidence par le truchement d'expériences élémentaires de lecture où l'on peut observer qu'un individu extrait plus d'information d'un texte que celui-ci n'en contient réellement. Ceci s'explique par le fait qu'il se réfère en même temps à son expérience personnelle ou à ses connaissances sur le sujet.

Les théories mises de l'avant par divers chercheurs ont proposé des noms différents pour ces structures mentales. Minsky parle de "frames", Miller de "chunks", Schank de "scripts", Papert et Piaget de schèmes, pour ne nommer que ces auteurs. Ces noms réfèrent à différentes interprétations des modes selon lesquels le savoir est codé et enregistré par le cerveau humain. Le but de la présente étude n'est pas de discuter des mécanismes de la représentation des connaissances, mais simplement de reconnaître le principe de leur existence, et surtout leur importance dans le contexte de la solution de problèmes mathématiques.

En second lieu, un individu doit, pour arriver à résoudre des problèmes, posséder aussi des connaissances de nature procédurale qui lui permettront de concevoir et d'exécuter certaines tâches spécifiques. A défaut d'un tel savoir, un individu est absolument incapable de s'engager efficacement dans un comportement actif orienté vers la solution d'un problème. Ainsi, par exemple, un étudiant peut savoir ce qu'est un système linéaire sans pour autant être capable d'en résoudre un seul, même incorrectement. Ce savoir procédural, qui peut être relié aux mathématiques ou aux habiletés générales de la pensée, serait intégré aux structures mentales conservées en mémoire.

Enfin, les étapes décrivant la résolution de problèmes peuvent être rattachées à deux processus ou activités se déroulant simultanément ou presque. Le premier de ces processus est constructif, hautement spécifique et détaillé; il synthétise les résultats de l'examen de la mémoire, la localisation et le rappel de notions préalablement acquises, la construction de représentations et l'établissement de correspondances. Le second est analytique et il implique un retour en arrière, une évaluation de l'atteinte des objectifs, en même temps que les mises au point et ajustements nécessaires et l'examen critique des résultats.

Cette interprétation de la nature et des processus de la résolution de problèmes, tels que vus par la science cognitive, a conduit à postuler que, pour résoudre des problèmes, un étudiant doit posséder (à la manière d'un système expert, ou n'est-ce pas plutôt celui-ci qui prend modèle sur celui-là?)

une BASE DE CONNAISSANCES,

contenant le plus grand nombre possible de concepts,

et des HABILETES

reliées aux mathématiques ou à la pensée en général.

1.2 Analyse des contenus des programmes de mathématiques

La première étape de la recherche consista à analyser les principaux cours de mathématiques de niveau collégial. Cette opération de micro-analyse et de macro-analyse, avait pour but d'identifier et de représenter, selon une technique adaptée de Pask (1976), les concepts et les relations autour desquels ces cours sont construits.

Les cours considérés pour les fins de cette analyse étaient les suivants:

201-101-77: Compléments de mathématiques
201-103-77: Calcul différentiel et intégral I
201-105-77: Algèbre vectorielle et linéaire; géométrie
201-203-77: Calcul différentiel et intégral II
201-307-77: Probabilités et statistiques.

Ces cours constituent le noyau central du programme mathématique suivi par les étudiants qui s'orientent vers les études en science ou en administration à l'université. Des cours supplémentaires couvrant des sujets plus avancés (par exemple le 303 ou le 205) peuvent également être suivis, et le sont effectivement, mais plutôt par un petit nombre d'étudiants académiquement forts. D'autre part, les étudiants du secteur professionnel suivent des cours dont plusieurs portent des numéros d'identification différents, mais dont les contenus recoupent ceux d'un ou de plusieurs des cours énumérés ci-dessus.

Ces cours traitent des principaux sujets enseignés en mathématiques au collégial. De nouveaux programmes, en voie d'élaboration depuis plusieurs années, doivent entrer en application en 1989. Les cours sont déjà connus, et les changements qu'ils apportent seront davantage liés à l'organisation qu'au contenu. C'est pourquoi il est raisonnable de penser que les résultats de la présente recherche ne risquent pas d'être rendus désuets par cette réforme des programmes.

Cette analyse a révélé que l'ensemble des mathématiques de niveau collégial est structuré autour des sept principaux sujets ou thèmes suivants:

1. Nombres
2. Fonctions, relations et variables
3. Différentiation
4. Intégration
5. Espace géométrique
6. Espace algébrique
7. Probabilités et statistiques

Les schémas élaborés au cours de cette analyse conceptuelle apparaissent à l'Appendice A. Le schéma 1 représente le développement de ces sept thèmes en domaines du savoir qui se recoupent suivant qu'ils réfèrent ou non aux mêmes concepts. Il contient également les principaux concepts rattachés à chacun de ces thèmes, ainsi que leurs interrelations.

Les schémas 2 à 8 présentent une analyse détaillée de chacun de ces thèmes. Dans ces graphes, les concepts dont la manipulation requiert une notation spéciale ont été identifiés pour utilisation ultérieure au cours de la recherche. On remarquera également que ces graphes sont constitués d'arcs non dirigés. Ce choix résulte du fait que l'utilisation de flèches risquait d'être interprétée comme indiquant une antécédence conceptuelle ou temporelle d'un concept sur un autre, alors que l'ordre de présentation des concepts, dans un cours, relève souvent plus d'un choix pédagogique que d'une priorité absolue.

1.3 Objectifs de l'éducation mathématique

La discussion préliminaire portant sur la nature des processus impliqués dans la résolution de problèmes, et l'analyse conceptuelle des cours constituant le programme en mathématiques, avaient pour but de mettre en évidence les objectifs de l'éducation mathématique au collégial. Pour les fins de cette recherche, ces objectifs ont été classifiés en trois catégories principales selon qu'ils relèvent des mathématiques, de la cognition ou de la méta-cognition.

1.3.1 Objectifs mathématiques

L'objectif prépondérant de l'éducation mathématique est de favoriser, chez les étudiants, l'acquisition de connaissances et le développement d'habiletés reliées au contenu des programmes. Pour atteindre cet objectif, les étudiants doivent maîtriser un certain nombre d'outils servant à la notation.

Savoir déclaratif ou
Ce que les étudiants doivent savoir

Selon Davis (1984), la préoccupation principale de l'enseignement des mathématiques est d'aider les étudiants à se construire un vaste ensemble de structures mentales qui constitueront des formes de représentation puissantes pour les concepts fondamentaux en mathématiques. Dans la situation qui nous intéresse, les concepts visés sont reliés aux sept thèmes ou domaines identifiés précédemment, soit les nombres, les fonctions et les variables, la différentiation, l'intégration, les espaces géométriques et algébriques, ainsi que les probabilités et les statistiques.

Ces modules d'information complexes et intimement liés entre eux sont à l'origine de toute activité mathématique. Quiconque est dépourvu de connaissances préalables dans un domaine particulier sera également dépourvu de moyens pour résoudre des problèmes reliés à ce domaine. Un étudiant auquel on demande de calculer le volume maximal d'un corps doit être capable de se référer à des structures qui, une fois ramenées dans la mémoire active, vont poser un certain nombre de questions relatives au calcul du volume d'un solide, de sa représentation fonctionnelle, de la dérivation et des valeurs critiques de la dérivée.

Afin d'être fonctionnelles en situation de résolution de problèmes, ces structures de représentation des connaissances doivent être aussi vastes, complètes et exemptes d'erreurs que possible. Plus elles sont riches, plus il est probable que des associations fructueuses seront établies, que des modèles adéquats seront remémorés ou construits en temps réel, et que la solution commencera à être ébauchée rapidement.

Savoir procédural ou
Ce que les étudiants doivent être capables de faire

Parallèlement à la construction des concepts mathématiques, les étudiants doivent développer des habiletés. Quel que soit le sujet auquel ces habiletés se rattachent, leur développement est généralement lié à la pratique d'une ou plusieurs des activités suivantes:

. manipuler des nombres, des fonctions, des structures, de façon à développer un sens des mathématiques et des habiletés de calcul numérique et symbolique.

. construire et appliquer des procédures afin de maîtriser les outils et algorithmes formels des mathématiques.

. percevoir et analyser une situation, construire un modèle, et le mathématiser au moyen d'équations, de graphes, de diagrammes et d'autres outils permettant l'expression de relations structurelles.

. expérimenter des modèles, dans le but de déterminer les implications des hypothèses de modélisation et d'interpréter les résultats de la solution dans le contexte de la question posée par le problème de départ.

L'observation et l'expérience montrent que la quasi-totalité du travail accompli par les étudiants, en relation avec leurs cours de mathématiques, est reliée à l'acquisition et au développement de l'une ou l'autre des quatre habiletés décrites plus haut.

Outils de notation ou
Ce que les étudiants devront utiliser

Afin de s'engager dans l'activité mathématique, c'est-à-dire, afin d'appliquer les concepts mathématiques à des procédures spécifiques, les étudiants doivent maîtriser un certain nombre d'outils de notation.

La première fonction de la notation est d'assister les processus de construction de solution ou de preuve. Il est possible d'utiliser les phrases du langage naturel pour présenter un raisonnement ou expliquer comment une solution a été élaborée. Mais les symboles de la notation mathématique conventionnelle assurent une façon uniforme (et relativement indépendante des idiomes) de traiter le raisonnement mathématique. Ceci permet, du moins aux initiés, de le parcourir d'un coup d'oeil rapide et de le comprendre aisément.

Bien sûr, il serait possible à chacun de mettre au point un système de notation qui lui convienne. Mais cette façon de faire rendrait toute communication difficile sinon impossible. Ce serait donc rendre un bien mauvais service aux étudiants que de ne pas leur permettre de se familiariser avec les outils de notation traditionnels utilisés pour la représentation et le traitement de l'information mathématique.

Enfin, la notation utilisée en mathématiques ne résulte pas que de choix arbitraires. Certaines formes de notation (le quotient différentiel df/dx proposé par Leibniz, comparé aux notations utilisées par Newton ou Lagrange, par exemple) sont connues pour faciliter la compréhension et agir comme heuristiques en suggérant de nouvelles façons de représenter des objets et des données.

Pour toutes ces raisons, les cours de mathématiques doivent assurer aux étudiants la possibilité de maîtriser les outils de notation nécessaires à la représentation des concepts et procédures à apprendre. Ces outils servent à représenter des objets que les étudiants auront à manipuler, des opérations qu'ils auront à exécuter sur ces mêmes objets, ainsi que des graphes qui permettront une visualisation des objets et des opérations. Voici une liste de ces outils de notation, regroupés selon les thèmes du programme et classifiés selon ce mode:

1. Nombres

objets: nombres naturels, entiers, rationnels, irrationnels, réels, complexes.

opérateurs: pour les différentes opérations à exécuter sur ces nombres (addition, soustraction, multiplication, division, exponentiation et extraction de racines, inverses, arithmétique modulo).

graphes: la droite numérique, le plan complexe.

2. Fonctions, relations et variables

objets: correspondances entre les éléments de divers ensembles, représentées la plupart du temps par des équations de la forme $y = f(x)$, des tableaux.

-algébriques: $y = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$

$$y = \frac{f(x)}{g(x)}$$

$$y = \sqrt[n]{f(x)}$$

-non algébriques: $y = a^x$, e^x

$y = \log x$, $\ln x$

$y = \sin x$, $\cos x$, $\tg x$, $\cotg x$, $\sec x$, $\operatorname{cosec} x$

$y = \arcsin x$, $\arccos x$, $\arctg x$, $\operatorname{arccotg} x$ $\operatorname{arcsec} x$, $\operatorname{arccosec} x$

$y = \sinh x, \ldots$ $y = \operatorname{arcsinh} x, \ldots$

$y = \lfloor x \rfloor$

$y = [x]$

$y = n!$

opérateurs: pour les différentes opérations à exécuter sur les fonctions (addition, soustraction, multiplication, division, composition, inversion).

graphes: en coordonnées cartésiennes, polaires, sphériques.

3. Différentiation

objets: fonctions et variables

opérateurs: la limite: $\lim_{x \to a} f(x)$

la dérivée: y', $f'(x)$, $\frac{dy}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$

graphes: fonctions et dérivées successives.

4. Intégration

objets: fonctions et variables, différentielles, suites et séries.

opérateurs: intégrale indéfinie, définie, impropre, sommation:

$$f(x)dx, \qquad \int_a^b f(x)dx, \qquad \lim_{t \to \infty} \int_a^t f(x)dx, \qquad \sum_{i=1}^{n} u_i$$

graphes: fonctions et primitives, aires de Riemann, surfaces et volumes.

5. Espace géométrique

objets: points, droites, courbes, plans, figures et solides dans l'espace à 1, 2 ou 3D
vecteurs géométriques: $\vec{v}$

opérateurs: opérations et transformations.

graphes: représentation des lieux géométriques.

6. Espace algébrique

objets: vecteurs algébriques: $\vec{v} = (a_1, a_2, \ldots, a_n)$
matrices: $A_{ij} = \begin{bmatrix} a_{11}, & a_{12}, \ldots, a_{1n} \\ a_{21}, & a_{22}, \ldots, a_{2n} \\ \cdot & \\ \cdot & \\ \cdot & \\ a_{n1}, & a_{n2}, \ldots, a_{nn} \end{bmatrix}$

opérateurs: addition, soustraction, produit par un scalaire, produit scalaire, vectoriel et mixte, transposée, déterminant, inverse, solution d'équations.

graphes: graphes (dirigés et non dirigés) correspondant à des matrices.

7. Probabilités et statistiques

objets: nombres, ensembles, fonctions et variables, séries statistiques.

opérateurs: analyse combinatoire, théorie des ensembles, sommations, statistiques descriptives, corrélation et régression, analyse de la variance, tests d'hypothèses.

graphes: histogrammes, diagrammes, courbes.

1.3.2 Habiletés cognitives

Le processus de construction de représentations, impliqué dans la résolution de problèmes, requiert plus que des connaissances et des habiletés mathématiques. En fait, il nécessite plusieurs des habiletés cognitives qui sont requises pour l'activité intellectuelle en général. Ces

habiletés étant absolument préalables à la véritable activité mathématique, leur développement fait partie des objectifs de l'éducation mathématique.

Il existe une autre raison pour laquelle l'éducation mathématique doit viser à assurer le développement de ces structures mentales. En effet, force est de constater qu'une bien petite partie des mathématiques scolaires est vraiment applicable aux situations concrètes de la vie quotidienne, et que très peu d'étudiants choisiront une carrière dans laquelle ils auront l'occasion d'appliquer les connaissances acquises aux cours. C'est pourquoi, si l'acquisition de concepts et de procédures constitue l'unique résultat des cours de mathématiques, il est à craindre qu'on n'en trouve plus guère de trace dans l'esprit des étudiants après le passage du temps. Si, au contraire, l'éducation mathématique vise également le développement d'habiletés de pensée plus générales, non seulement les étudiants réussiront-ils mieux à résoudre des problèmes, mais il est permis d'espérer qu'il y aura un transfert de ces capacités à des situations non reliées aux mathématiques.

Quelles sont les structures cognitives reliées aux opérations de recherche, de localisation et de rappel des connaissances mathématiques, à la construction de représentations et à l'établissement de correspondances? Davis (1984) reconnaît les quatre suivantes, et énumère certains comportements qui témoignent de leur présence et qui peuvent servir d'indices quant à leur stade de développement. L'ordre dans lequel elles apparaissent ici n'est en aucune façon absolu; elles ont simplement paru aller ainsi, du moins au plus sophistiqué. De toute façon, il y a probablement ici une dépendance en spirale qu'il vaudrait la peine d'examiner plus en profondeur.

Structure de compréhension des procédures

- suivre des instructions énoncées en langage naturel;
- prédire le résultat ou la conséquence de l'exécution d'une procédure.

Structure combinatoire

- faire l'inventaire des possiblités;
- conduire des recherches systématiques;
- construire des arbres de décision.

Structure de comparaison

- identifier des relations;
- établir l'antériorité;
- déterminer les priorités;
- trouver un ordre;
- construire des hiérarchies.

Structure logique

- établir des correspondances
- faire des inférences, construire des raisonnements.

1.3.3 Habiletés méta-cognitives

Les habiletés méta-cognitives sont celles qui permettent à un individu de réfléchir sur ses propres processus cognitifs. Schoenfeld (1987) écrit que la recherche portant sur la méta-cognition s'est concentrée jusqu'ici sur trois catégories de comportement intellectuel, soit la prise de conscience par un individu, de son activité mentale (self-awareness), l'auto-régulation de cette activité (self-control) et les croyances et intuitions qu'il véhicule.

La prise de conscience se manifeste surtout par l'aptitude d'un individu à décrire les moyens qu'il utilise pour mémoriser l'information, la classifier, la localiser et la ramener au besoin, ainsi que par la justesse de ses estimations quant à ses performances attendues et réelles.

L'auto-régulation recouvre l'ensemble des attitudes critiques qui mènent un individu à examiner, corriger ou modifier son propre raisonnement. Dans la résolution de problème, elles interviennent dès qu'un individu évalue son progrès, explore un arbre de décision, vérifie si les hypothèses qui sous-tendent le modèle qu'il a construit reflètent bien la structure du problème, ou cherche à en valider la logique et la cohérence, questionne et interprète les résultats et les teste en regard de la situation concrète. L'auto-régulation se manifeste encore lorsque l'individu utilise des stratégies pour classifier les problèmes, choisir des approches de solution et construit des heuristiques qu'il utilisera de façon systématique par la suite (v.g. ne jamais utiliser une technique difficile avant d'avoir vérifié si une technique simple ne pourrait faire l'affaire).

Enfin les croyances et intuitions sont le bagage qu'un individu transporte avec lui, relativement (dans le cas qui nous intéresse ici) à la difficulté, à l'utilité des mathématiques ou à ses capacités à réussir dans ce domaine. Qu'elles soient conscientes ou non, ces croyances influencent son comportement et ses performances.

Selon Schoenfeld, les habiletés méta-cognitives sont aussi essentielles à la réussite dans la résolution de problèmes que ne le sont les habiletés mathématiques ou cognitives. Elles débouchent sur des considérations épistémologiques et sont absolument indispensables au succès de qui veut "apprendre à apprendre".

L'enseignement peut influencer le développement de ces habiletés. Schoenfeld suggère l'utilisation du video, les jeux de rôle et la discussion de problèmes en petits et en grands groupes. Pour sa part, Vygotsky (1978) insiste sur l'interaction avec le groupe de pairs: "Toutes les fonctions qui actualisent le développement de l'enfant apparaissent deux fois, écrit-il; d'abord au niveau social, ensuite au niveau individuel; d'abord entre les individus, puis à l'intérieur de chacun. Ceci s'applique aussi bien à l'attention volontaire, qu'à la mémoire logique ou à la formation de concepts. Toutes les fonctions cognitives d'ordre supérieur ont leur origine dans les relations entre individus."

Cette étude, qui s'est ouverte sur une réflexion au sujet de la nature des mathématiques scolaires et de la résolution de problèmes, a mené, par le biais d'une analyse conceptuelle des programmes de mathématiques au niveau collégial, à l'identification de trois grandes catégories d'objectifs dans l'éducation mathématique. Ces objectifs sont de nature mathématique, en ce qu'ils se rattachent aux concepts, aux procédures ou aux outils de notation. Ils sont aussi de nature cognitive et méta-cognitive. La suite de la recherche consista à montrer comment et en quoi l'utilisation de divers outils de programmation peut favoriser l'atteinte de ces objectifs.

CHAPITRE II

LES LANGAGES DE PROGRAMMATION

"En libérant l'esprit de tout travail inutile,
un bon système de notation
amplifie la puissance mentale."

A. N. Whitehead

La seconde étape de la recherche consistait à choisir quelques langages de programmation de types variés, utilisables pour l'apprentissage des mathématiques, puis à construire une grille d'évaluation pertinente. On trouvera, en première partie de ce chapitre, un bref aperçu du développement des langages de programmation; il a semblé, en effet, que cet historique permettrait de mieux comprendre la situation qui prévaut présentement, et ainsi d'anticiper les développements futurs. Il a également permis de sélectionner les logiciels qui furent retenus pour évaluation. La suite du chapitre identifie les caractéristiques des langages de programmation. Les résultats de cette analyse ont conduit à l'élaboration de la grille d'évaluation des langages de programmation qui est reproduite à la fin de ce chapitre.

2.1 Le développement des langages de programmation

Les langages de programmation sont des canaux de communication mis au point par l'homme pour entrer en interaction avec une machine de sa fabrication, l'ordinateur.

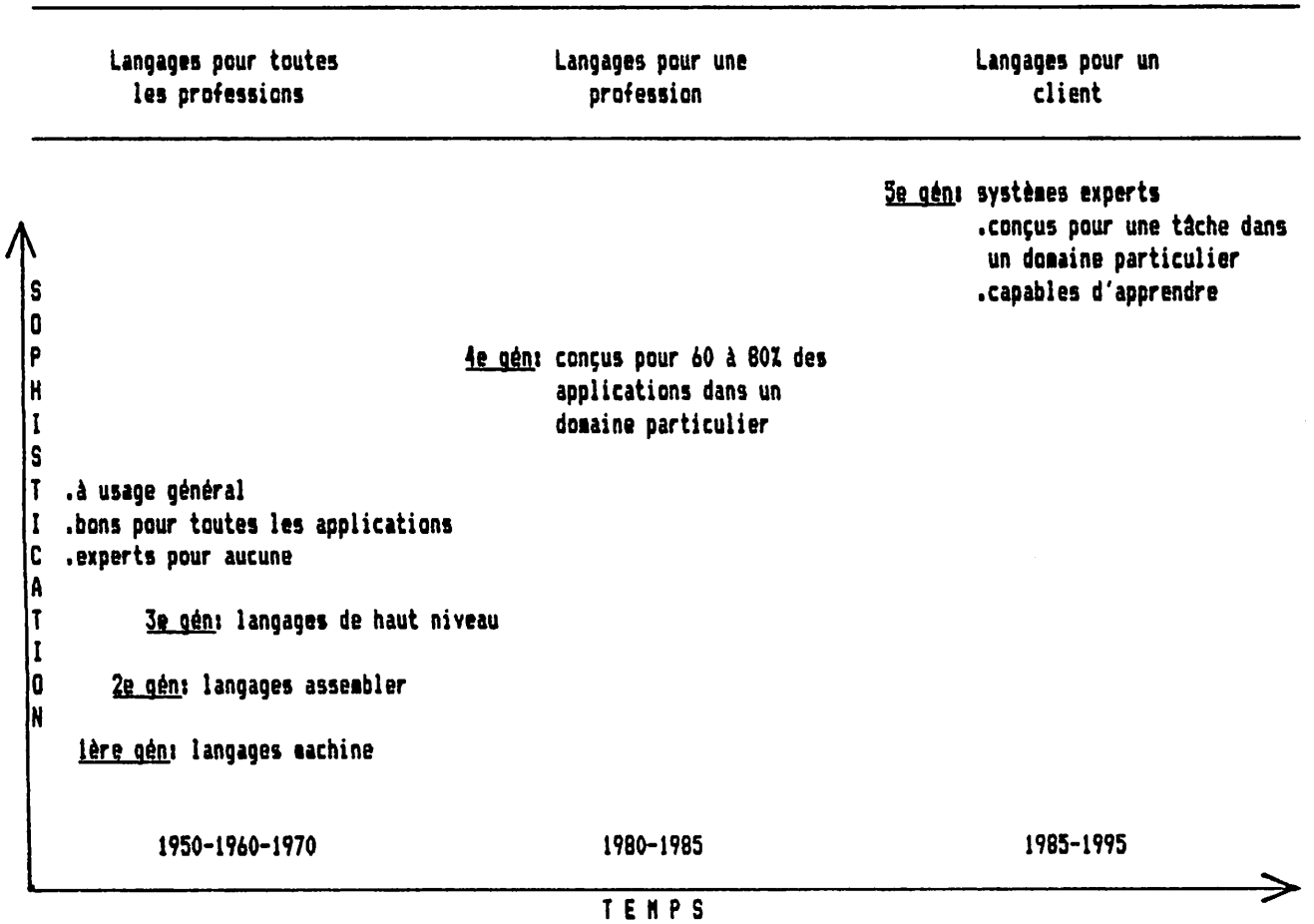
L'évolution des langages de programmation a été soumise à deux types de contraintes. Les premières étaient liées au matériel; l'espace mémoire des premiers ordinateurs était très limité, et le coût élevé des composantes rendait impérieux le besoin de maximiser le rendement de la machine. Les premiers langages de programmation ont donc été conçus par et pour des experts qui devaient trouver les moyens de s'en accommoder et de s'ajuster ainsi aux configurations de la machine. Les secondes contraintes étaient dues davantage au manque de connaissance et de compréhension des structures de la pensée et des modes de communication privilégiés par l'humain.

Le premier obstacle au progrès est en voie de disparaître: le coût des ordinateurs diminue rapidement, en même temps qu'augmente leur puissance. Il est déjà possible de se procurer à bas prix, des micro-ordinateurs dotés d'une unité centrale de traitement 32 bits et dont la capacité de mémoire se chiffre en méga-octets. Les problèmes de temps et de mémoire ne doivent donc plus désormais être invoqués pour justifier l'existence de systèmes non conviviaux.

Pourtant si de tels systèmes continuent d'exister, c'est que le second obstacle demeure toujours présent: malgré les plus récents développements en intelligence artificielle, et malgré les résultats de la recherche sur les facteurs humains dans la communication homme-ordinateur, on ne possède encore que relativement peu de données sur lesquelles fonder la conception de systèmes intégrant les modes de communication naturels à l'homme.

Tout comme l'ordinateur, les langages de programmation ont connu des développements qui s'expriment en termes de générations. Dans un livre récent, Chorafas (1986) représente l'évolution des langages en fonction du temps.

Tableau 1: Evolution des langages de programmation selon les générations



Le Tableau 1 illustre les deux tendances qui prévalent actuellement: la première est de construire des langages spécialisés selon la tâche à accomplir; la seconde est de produire des outils de programmation de plus en plus sophistiqués, c'est-à-dire qui confèrent davantage de puissance formelle à l'utilisateur. Selon Chorafas, ces deux tendances sont irréversibles.

On remarque également, dans ce tableau, que les trois premières générations de langages sont apparues au cours des années 50, que la plupart des langages dits de 3e génération ont été développés entre 1950 et 1980, que l'élaboration des langages de 4e génération coïncide avec l'apparition de la micro-informatique et que l'impact des systèmes experts est encore à venir. Le développement des langages est donc lié de près à celui des ordinateurs; mettre ce lien en évidence aide à comprendre le sens de leur évolution.

La programmation sur ordinateur comprend deux activités: la première consiste à construire des algorithmes ou des procédures permettant de résoudre des problèmes donnés; la seconde consiste à les exprimer dans un code approprié, de façon qu'elles puissent être exécutées par l'ordinateur. Ces activités sont distinctes, mais non indépendantes, et la seconde n'est nécessaire que parce que l'esprit humain et l'ordinateur utilisent des moyens différents pour représenter, sauvegarder et manipuler l'information. Toute communication entre l'un et l'autre exige que s'établisse une correspondance, une traduction en quelque sorte, d'un symbolisme à l'autre. A l'époque des premiers ordinateurs, cette traduction était entièrement assumée par le programmeur qui devait s'exprimer en utilisant carrément le code binaire; avec les ordinateurs de la 5e génération, il est prévu que l'usager pourra s'exprimer en langue naturelle. Entre ces deux extrêmes, le fardeau de la traduction est graduellement transféré de l'homme à l'ordinateur.

On n'a pas encore réussi à écrire un programme de traduction (assembleur, compilateur ou interpréteur) qui puisse accepter comme entrée un programme écrit en langue naturelle, s'il faut entendre par ces termes le langage que nous utilisons quotidiennement. Cependant, un très grand nombre de langages ont été inventés, qui ont plus de lien commun avec le discours quotidien qu'avec le code binaire, ainsi qu'en témoignent les lignes qui suivent

<u>1ère génération: langages machine, ou de bas niveau</u>

Chaque micro-processeur, ou unité centrale de traitement de l'information dans un micro-ordinateur, possède son propre langage, ou jeu d'instructions, intégré à ses circuits. Deux symboles seulement sont utilisés dans ces langages: ce sont les nombres 0 et 1, qui correspondent aux

variables logiques (vrai ou faux), ou encore à l'état d'un circuit (ouvert ou fermé).

Le langage machine est, comme son nom l'indique, le seul langage compris par l'ordinateur. Tout programme écrit dans un autre langage doit donc être traduit en langage machine avant d'être exécuté. Il s'ensuit que l'exécution d'un programme écrit directement en langage machine est beaucoup plus rapide que celle d'un programme écrit dans tout autre langage. L'utilisation du langage machine permet également un accès direct aux circuits de la machine. Voilà pourquoi on a longtemps eu recours à ces langages pour construire des compilateurs ou des systèmes d'exploitation.

Les langages machine apparurent en même temps que les premiers ordinateurs commerciaux, et leur usage a toujours été réservé aux programmeurs professionnels. Le premier pas dans la direction d'un langage plus facile à manier pour l'homme fut de coder les programmes en notation octale, puis hexadécimale. Bien que pouvant paraître de peu d'importance à notre époque, ce changement réduisait considérablement le nombre de caractères devant être manipulés par le programmeur pour transmettre une instruction simple à l'ordinateur. Cependant, quand bien même ils sont écrits sous forme hexadécimale, les nombres sont difficiles à manipuler et à mémoriser, car il n'est pas usuel d'exprimer des idées à l'aide de nombres. Qui plus est, chaque micro-processeur possède son propre langage, car il n'existe pas de langage machine universel malgré les quelques points qu'ils aient en commun.

La programmation en langage machine est restée une tâche ardue et pénible. Tous admettent que l'ordinateur personnel n'aurait pas connu la popularité qu'il a aujourd'hui s'il n'eut existé que ces seuls langages. En fait, comme il n'y aurait que peu ou pas d'utilisateurs non professionnels, le concept même d'ordinateur personnel n'existerait probablement pas.

2e génération: assembleur, ou langage de niveau intermédiaire

La suite logique de ce passage de la notation binaire à la notation octale, puis hexadécimale, fut de remplacer celles-ci par un symbolisme encore plus facile à utiliser par l'homme. En effet, puisqu'un nombre hexadécimal pouvait signifier une instruction à l'ordinateur, pourquoi les lettres de l'alphabet ne pourraient-elles pas en faire autant? Les codes mnémoniques firent donc leur apparition sur la scène informatique, et le programmeur put s'adresser à l'ordinateur en utilisant des abréviations de mots du langage naturel. De là à utiliser des codes semblables pour représenter les différents registres et adresses de

l'ordinateur, il n'y avait qu'un pas. C'est ainsi qu'apparurent les assembleurs, qui ne sont rien d'autre que des programmes capables d'accepter des codes mnémoniques en entrée, de consulter une table de symboles pour interpréter ce code, puis d'effectuer la substitution appropriée en langage machine.

L'énoncé ADD A,10 représentait, sur ses équivalents binaire ou décimal, une amélioration remarquable quant à la facilité de manipulation et surtout de mémorisation. Cependant, il existait encore essentiellement une correspondance biunivoque entre les instructions au micro-processeur et celles du langage assembleur. De façon générale, il fallait encore plusieurs instructions pour faire exécuter une opération assez simple; de plus, le programmeur devait se soucier d'une foule de détails minuscules reliés à l'architecture particulière du micro-processeur. La programmation en langage assembleur était donc encore un processus assez laborieux. De nos jours, on l'utilise surtout pour des raisons d'économie de temps ou d'espace: si un ensemble d'instructions doit être exécuté un grand nombre de fois dans un programme long, il peut être rentable de récrire cette portion du programme en assembleur. Enfin, les programmes écrits en assembleur ne sont pas transférables sur des appareils de types autres que celui pour lequel ils ont été écrits.

L'apparition des langages assembleur jetait quand même les bases de ce qui allait devenir le problème central de la conception de logiciels, soit la production d'outils permettant de construire des programmes faciles à écrire, à simuler, à modifier et à mettre à jour. Au-delà du remplacement des codes numériques par les codes mnémoniques et l'adressage symbolique, on vit apparaître à cette époque,

-l'utilisation des sous-routines qui, en prenant de l'ampleur, devait conduire aux compilateurs;

-une deuxième tentative de concevoir un langage de programmation universel (UNCOL), laquelle devait échouer;

-la différence entre les opérations de compilation et d'interprétation.

3e génération: langages de haut niveau

Le développement significatif qui suivit consista à se défaire du principe de correspondance un-à-un en remplaçant les énoncés à instruction unique par des énoncés complexes. Désormais, pour signifier à l'ordinateur d'ajouter à la valeur de la variable appelée A, la valeur de la variable appelée B, et d'appeler le résultat C, on n'avait qu'à écrire C = A + B. Avec ce symbolisme, le programme de

traduction devait désormais être capable de transformer un énoncé symbolique unique en plusieurs énoncés du langage machine. Ces nouveaux programmes furent appelés compilateurs; ils furent suivis de près par les interpréteurs.

C'est ainsi qu'apparurent, à la fin des années 50, les langages de haut niveau. FORTRAN et COBOL furent parmi les premiers à être implantés sur ordinateur, le premier pour les applications scientifiques et le second pour la gestion. Depuis, quelques milliers d'autres langages ont été développés; ceux dont l'usage est plus répandu sont naturellement mieux connus. On peut citer entre autres, selon la date de leur apparition sur le marché, LISP, BASIC, APL, Logo, Pascal, PROLOG, Forth, Pilot, C, Smalltalk et Modula. Certains sont destinés à des usages spécialisés, mais la plupart peuvent être utilisés à des fins multiples.

Les langages de 3e génération possèdent, à des degrés divers, un certain nombre d'avantages sur les langages des générations précédentes. De façon générale, ces avantages résultent de ce que ces langages utilisent des moyens de communication se rapprochant de ceux privilégiés par l'homme. Ces langages sont plus compacts: un énoncé en langage de 3e génération peut correspondre à plusieurs énoncés en langage de bas niveau. Leur vocabulaire est plus près du langage naturel. Ils sont donc plus faciles à maîtriser et à mémoriser. La plupart admettent la modularisation, c'est-à-dire la décomposition d'un problème complexe en sous-problèmes. Toutes ces qualités permettent la construction de programmes plus faciles à écrire, à simuler, à corriger et à mettre à jour.

D'un point de vue technique, les programmes écrits en langages de haut niveau doivent aussi être traduits en langage machine avant d'être exécutés. Cette traduction est effectuée par un programme compilateur (qui traduit le programme en entier avant de l'exécuter), ou interpréteur (qui traduit et exécute le programme ligne par ligne.) Les programmes compilés sont exécutés plus rapidement que les programmes interprétés, car chaque ligne de code n'est traduite qu'une seule fois quel que soit le nombre de fois où elle sera exécutée. Par contre, les programmes interprétés laissent place à plus de dynamisme dans l'interaction avec l'ordinateur. Enfin, les langages de 3e génération sont conçus pour être utilisés sur une vaste catégorie d'appareils, et les différentes versions d'un même langage ne présentent entre elles que des variations mineures, davantage liées à l'environnement de programmation qu'à la structure même du langage.

Les langages de 3e génération sont les premiers auxquels on a pu s'intéresser en milieu scolaire, à des fins d'apprentissage autres que celui de l'informatique. Dans le contexte de l'éducation mathématique, les systèmes FORTRAN, puis APL et BASIC, implantés sur ordinateur central, ont été utilisés principalement pour la formation des étudiants en sciences et en génie. Depuis l'avènement de la micro-informatique, plusieurs sources rapportent également l'utilisation, la plupart du temps expérimentale, de langages tels BASIC, APL, Logo ou Smalltalk à des fins d'enseignement et d'apprentissage des mathématiques, de l'élémentaire à l'université.

Il faut signaler également l'existence de nombreux langages conçus dans les milieux académiques, à des fins de recherche et de développement, et parfois utilisés localement pour l'enseignement et l'apprentissage de diverses disciplines. (Mentionnons, par exemple, les langages de Turner, SASL, KRC, Miranda, et la série des langages POP, développés dans diverses universités anglaises.) Les soucis pédagogiques ayant prévalu lors de la construction de ces produits en font des outils potentiellement intéressants, mais la plupart ne sont pas distribués commercialement et demeurent peu connus. Au cours des dernières années, certains de ces langages se sont retrouvés sur le marché. C'est le cas notamment de POP-11, implanté depuis peu sur MacIntosh, de Trilogy distribué par une maison de Vancouver, ainsi que de Q'NIAL, développé à l'Université de Kingston, Ont.

On ne peut terminer cette revue des langages de 3e génération sans mentionner les langages orientés objets, qui représentent une forme évoluée des langages procéduraux. Dans ces langages, le concept objet domine celui de fonction. Alors que dans les langages procéduraux, une fonction est appliquée à un objet, une fonction retourne une valeur, ou une procédure est appelée par une autre procédure, en langage objet, un objet reçoit un message, retourne un message ou envoie un message à un autre objet. Dans ces langages, l'idée de modularité est mise en valeur de façon exceptionnelle. On y définit des classes d'objets en décrivant la structure des éléments qui seront associés à ce type d'objets, et en définissant des procédures qui seront appliquées à ces objets. Chaque objet appartient à une classe, ce qui détermine les propriétés de cet objet. Les classes sont organisées en hiérarchie, ce qui fournit également une taxonomie des propriétés.

Smalltalk est le prototype des langages orientés objets, c'est-à-dire celui dans lequel ce paradigme est implanté de la façon la plus complète. Toutefois, on a récemment intégré à plusieurs langages conventionnels (dont C, Logo, Pascal) des dispositions permettant l'approche de programmation orientée objet.

Langages de 4e génération

Les langages de 4e génération désignent les logiciels les plus couramment utilisés aujourd'hui par la vaste majorité des propriétaires d'ordinateurs personnels ou de bureau. Pour reprendre les termes de Chorafas (1986), "Les programmes produits dans le commerce nous tombent dessus comme les feuilles en automne! Logiciels de traitement de texte, de graphisme, de statistiques, de mathématiques financières, de génération de programme, bases de donnée, chiffriers, l'air en est rempli!..." En fait, certains de ces logiciels ne constituent pas des langages de programmation au sens traditionnel du terme, mais on constate que la distinction entre langages de programmation et applications est de moins en moins claire. De plus, selon Chorafas on peut s'attendre à des développements significatifs à tous les six mois.

Comme il a déjà été mentionné, ces logiciels sont conçus pour un bon nombre d'applications reliées à un domaine particulier. Leur but est de fournir un système de programmation rapide, versatile, bien structuré et facile d'accès, capable d'économiser du temps, d'éliminer le recours au codage en langage de bas niveau, de faciliter la correction, la mise à jour et la documentation des programmes. En fait, l'utilisation de ces logiciels à un niveau minimal permet d'obtenir des produits, texte ou base de données, par exemple, sans véritable programmation; l'utilisateur n'a qu'à répondre à quelques questions ou à faire certains choix dans les menus présentés par le logiciel.

Les caractéristiques communes à tous ces logiciels sont d'être extrémement faciles d'accès (la période d'initiation se mesure en termes d'heures), de faire usage d'une syntaxe simplifiée et claire, quoique rigoureuse, d'utiliser au maximum les moyens favorisant la pensée visuelle (what you see is what you get) c'est-à-dire, les fenêtres, les icones et les menus, ainsi que les modes d'entrée autres que le clavier (souris, pointeur, écran digital, clés de fonctions...)

Dans un contexte d'apprentissage des mathématiques, les langages de 4e génération pouvant présenter un certain intérêt sont les logiciels de calcul symbolique et statistique, les chiffriers, les bases de données, les logiciels dits intégrés et les coquilles de systèmes experts.

Les logiciels de calcul statistique permettent, outre les calculs reliés aux statistiques descriptives, l'analyse et les tests d'hypothèses, les calculs de corrélation et de régression, l'analyse de la variance à une ou plusieurs

variables, ainsi que la représentation graphique des résultats sous forme de divers graphes, diagrammes et histogrammes. Il en existe une grande variété dans le commerce; au niveau collégial, ils sont utilisés pour les cours de statistiques.

Les logiciels de calcul symbolique (Computer Algebra Systems) permettent le calcul numérique, l'arithmétique rationnelle ainsi que le calcul symbolique. Ils effectuent entre autres les opérations algébriques élémentaires, la factorisation et la simplification, la résolution d'équations et de systèmes d'équations, les opérations du calcul différentiel et intégral, ainsi que d'autres opérations reliées à l'analyse. Ces logiciels offrent également un langage de programmation permettant à l'utilisateur de programmer ses propres algorithmes en vue de résoudre des problèmes particuliers. Conçus à l'intention de la recherche, ces outils (REDUCE, MACSYMA, SMP) n'étaient à l'origine disponibles que sur ordinateur central. Ils existent maintenant en versions pour micro-ordinateurs (muMATH, sur Apple et IBM, et MAPLE, sur MacIntosh). Ils sont aux mathématiques collégiales ce que la calculatrice est aux mathématiques élémentaires et secondaires; ils ne dispensent pas d'apprendre, mais ils ont changé la façon de faire des mathématiques. On ne peut aujourd'hui se permettre d'ignorer leur existence.

Quant aux logiciels dits intégrés, ils représentent cette nouvelle catégorie de produits permettant à un utilisateur de travailler simultanément (c'est-à-dire sans avoir à changer de logiciel) avec un certain nombre d'applications. Il en existe présentement une grande variété pour le monde des affaires, mais malheureusement une pénurie navrante dans le milieu scolaire. Mentionnons, dans cette catégorie, le logiciel MathCAD, qualifié de bloc-notes mathématique, et un logiciel nommé CAL, intégrant des sous-ensembles de Logo et de Prolog, ainsi qu'un langage-auteur.

Il est certain que les chiffriers, bases de données, coquilles de systèmes-experts et, de façon générale, tous les logiciels permettant de construire des bases de connaissances présentent également plus d'une possibilité d'application dans l'enseignement et l'apprentissage des mathématiques. Cependant, c'est là un domaine dans lequel il existe encore relativement peu d'expertise, et dont l'étude ne pouvait être abordée dans le cadre de cette recherche.

Langages de 5e génération

Les langages de 5e génération sont ceux qui utilisent - peut-être vaudrait-il mieux dire utiliseront - les résultats de la recherche en intelligence artificielle. Selon Nickerson (1986), l'intelligence artificielle est un domaine qui n'a intéressé qu'une petite communauté de chercheurs universitaires pendant une trentaine d'années, mais qui retient l'attention de plus en plus d'investisseurs, depuis que les universités ont commencé à produire de nombreux diplômés compétents en la matière.

On désigne généralement sous le nom global de systèmes experts, ces vastes bases de connaissances dotées d'un moteur d'inférence qui pourront bientôt communiquer l'information, construire des raisonnements et même apprendre.

Il existe actuellement un certain nombre de systèmes experts utilisés encore principalement dans un contexte de recherche. Hayes-Roth (1984) décrit leurs performances de la façon suivante:

. ils résolvent des problèmes très difficiles, aussi bien ou même mieux que des experts humains;

. ils utilisent des heuristiques pour construire des raisonnements, ils peuvent envisager simultanément plusieurs hypothèses, et même fonctionner avec des données erronées et des règles imprécises;

. ils acceptent le langage naturel ou presque.

Dans la même catégorie de langages, on retrouve également les systèmes d'enseignement intelligemment assistés par ordinateur. Ceux-ci sont des programmes destinés au soutien de l'apprentissage, tout comme les logiciels utilisés dans l'enseignement assisté par ordinateur, mais dans lesquels sont appliquées les techniques de représentation sophistiquées de l'intelligence artificielle. Ils sont constitués de composantes distinctes servant respectivement à représenter les connaissances et habiletés à acquérir (module expert), l'état des connaissances de l'utilisateur (modèle de l'apprenant), les stratégies d'enseignement (module tutoriel) et parfois un langage auteur permettant à l'usager de modifier le contenu des modules ou la configuration du système selon le cas. Cette architecture innovatrice, en plus de favoriser une interaction avec l'ordinateur se déroulant en langage près du langage naturel et cherchant à modeler la relation étudiant-professeur, permet aussi, le cas échéant, leur réutilisation dans la construction d'autres programmes.

Au cours des dernières années, on a construit et utilisé, à des fins expérimentales, plusieurs tuteurs intelligents. En ce qui regarde les mathématiques, les premiers et les plus connus sont reliés à la résolution de problèmes (SOPHIE, Brown, Burton et Bell, 1975), au diagnostic des erreurs (BUGGY, Brown, 1978), à la manipulation des expressions arithmétiques (WEST, Burton et Brown, 1979), à la résolution des équations (Quadratic Tutor, Oshea, 1979), aux méthodes de preuve (EXCHECK, Suppes, 1981) et à l'intégration (Symbolic Integration, Kimball, 1982). La tendance actuelle est de concevoir des systèmes permettant de construire de vastes bases de donnees et de les explorer au moyen de la programmation logique.

Selon Kearsley (1987), la contribution la plus importante de ces programmes n'est pas d'avoir réussi une percée dans le domaine de l'enseignement et de l'apprentissage, mais d'avoir permis le développement de systèmes informatiques capables d'intégrer les processus privilégiés dans ce domaine. Pour l'instant, ces systèmes ne tournent que sur les équipements du type LISP Machine, ou SUN Stations; ils n'ont pas encore fait leur apparition dans les salles de classe, sinon à des fins purement expérimentales. Ils sont encore les outils de demain.

2.2 Langages retenus

Les langages de 1ère et de 2e génération constituent donc la borne inférieure du domaine d'exploration de ce projet de recherche, alors que les langages de 5e génération en constituent la borne supérieure.

Restent donc les langages de 3e et 4e génération, parmi lesquels il fallait choisir quelques logiciels pour fins d'évaluation.

Dans la perspective des objectifs de la recherche et des contraintes de la situation d'apprentissage, il paraissait important de tenir compte des caractéristiques suivantes:

. Compatibilité avec les mathématiques

Tous les langages informatiques permettent le calcul numérique (number crunching) mais tous ne permettent pas de le faire de façon aussi naturelle, soit parce qu'ils offrent moins d'opérations et de fonctions pré-définies, soit parce qu'ils ne permettent pas la définition de nouvelles fonctions, ou soit parce qu'ils forcent trop l'usager à se préoccuper de détails relevant de l'informatique. De même, tous les langages supportent au moins un type de structures

de données, mais tous ne permettent pas de représenter et de manipuler aussi facilement les divers objets dont il est question dans les cours de mathématiques de niveau collégial.

. Compatibilité avec l'apprentissage

Dans une situation d'apprentissage, que ce soit des mathématiques ou de l'informatique, l'attention porte davantage sur la construction d'une solution que sur l'utilisation des résultats. Il n'y a pas véritablement de phase d'exécution, au-delà de la mise au point du programme: dès qu'il est résolu, le problème perd de son intérêt et cède la place au problème suivant. C'est pourquoi la rapidité d'exécution assurée par un compilateur présente moins d'intérêt que la facilité de l'interaction que permet un langage interprété. Et bien que cette distinction ne soit plus aussi marquée qu'auparavant, puisqu'il existe des versions interprétées d'un langage qui sont plus rapides pour certaines opérations que d'autres qui sont compilées, un langage utilisé pour l'apprentissage des mathématiques doit être hautement interactif afin de permettre l'exécution d'instructions en mode direct et un échange rapide du contrôle entre l'ordinateur et l'usager pour la mise au point des programmes.

. Facilité d'accès

L'apprentissage visé est celui des mathématiques, et la programmation, un moyen. Le vocabulaire, la syntaxe, le style de programmation, l'interface usager et la documentation doivent être de nature à permettre une première utilisation du langage après une brève période d'initiation.

. Equipement

Enfin, comme l'évaluation devait être faite sur des systèmes de type compatible IBM, ce qui d'autre part semble être une norme assez répandue actuellement dans les collèges et ailleurs, il fallait sélectionner des versions compatibles avec cette catégorie d'équipement.

S'il avait existé un langage répondant de façon évidente et optimale à chacune de ces contraintes, le choix en aurait été facilité. Comme on le verra plus loin, ce n'était pas le cas. Logo, qui était à l'origine de ce projet de recherche et qui est un langage créé spécifiquement pour répondre aux besoins de l'apprentissage, devait évidemment être soumis à l'évaluation. APL et MuMATH furent retenus à titre de langages créés explicitement pour l'apprentissage des mathématiques. Q'Nial fut sélectionné à cause de ses similitudes et de ses différences avec APL, et aussi à cause du nombre d'applications qu'il rend possible.

BASIC, qui est encore (!) le langage le plus fréquemment enseigné au cours d'initiation à l'informatique du secondaire, fut choisi afin de déterminer l'utilisation qu'il serait possible d'en faire dans les cours de mathématiques au collégial. CAL fut sélectionné sur la foi d'une publicité qui en faisait un langage unissant Logo et les mathématiques symboliques et permettant l'exploration de l'algèbre, de la géométrie plane et sphérique, de même que celle de l'algèbre linéaire, du calcul, et de la logique. Enfin, MathCAD fut retenu comme logiciel intégré permettant le traitement de texte mathématique, le calcul numérique et symbolique de même que le graphisme.

Malgré l'intérêt évident que présentent les langages orientés objet, et malgré l'enthousiasme communicatif manifesté par des éducateurs (Arcouet et al., 1987) ayant conduit des recherches sur leur utilisation dans l'enseignement secondaire, aucun de ces langages n'a été retenu pour évaluation parce que leur implantation sur micro-ordinateur est assez récente et qu'il n'existe encore que peu d'expertise au sujet de leur utilisation dans l'éducation mathématique post-secondaire. On remarquera enfin que cette étude n'inclut pas non plus de logiciels de calcul statistique, car ceux-ci sont destinés à un usage spécifique et doivent être comparés entre eux si l'on est intéressé à connaître les outils les plus performants en ce domaine. (Voir Lehman, 1987)

2.3 Vers une comparaison

Une grille d'évaluation doit tenir compte à la fois des caractéristiques du produit à évaluer et de celles de la situation dans laquelle les produits sont utilisés. Comme on le sait, les revues spécialisées publient souvent des évaluations de divers logiciels, langages ou applications. D'autre part, il existe un bon nombre de grilles permettant l'évaluation des didacticiels et autres logiciels destinés à soutenir l'apprentissage en milieu scolaire. C'est à partir de ces sources qu'a été construite la grille d'évaluation de langages susceptibles d'être utilisés pour la résolution de problèmes, dans le contexte de l'apprentissage des mathématiques de niveau collégial.

Identification des caractéristiques des langages

Dans le but de faire connaître aux utilisateurs de l'ordinateur personnel, différents langages susceptibles de convenir à leurs besoins, Taylor (1984) fait l'évaluation de dix langages de 3e génération à partir des critères suivants:

. la représentation des données
. l'énoncé d'assignation
. les expressions arithmétiques
. les expressions logiques
. les entrées et sorties
. les structures de contrôle
. les structures de données
. la manipulation des fichiers
. le graphisme

Dans un livre consacré aux langages de 4e et 5e génération et destiné à permettre aux professionnels de faire la mise à jour de leurs connaissances, Chorafas (1986) propose les critères suivants pour évaluer comment un langage rencontre les besoins de programmation:

1. caractéristiques de la programmation
   . facilité à écrire des programmes
   . simplicité du langage à apprendre
   . flexibilité à répondre à des besoins particuliers
   . lisibilité du code et compréhensibilité des programmes
   . concision des programmes
   . qualité de la documentation

2. aspects reliés à la construction de bases de données
   . manipulation des structures de données
   . abstraction des données
   . gestion des fichiers
   . manipulation des entrées et sorties
   . ouverture à la communication des données
   . caractéristiques des structures de données
   . interface usager

3. considérations générales
   . efficacité du code
   . possibilité de tester les programmes
   . aide au "debugging"
   . décomposabilité
   . respect des normes
   . existence d'utilitaires
   . compatibilité avec compilation centrale
   . précision numérique
   . prévision de la mise à jour rendue nécessaire par l'évolution technologique
   . transférabilité des programmes
   . nombre d'utilisateurs (enseigné dans une université?)

Edwards et Hartwig (1985) proposent des normes à partir desquelles ils ont construit un banc d'essai qui est souvent utilisé à des fins d'évaluation dans la revue Byte. Ces

normes mesurent le temps nécessaire pour exécuter les tâches suivantes:

- enregistrer sur disquette un fichier de 64 Ko
- ramener en mémoire un fichier de 64 Ko
- trouver les nombres premiers < 7 000, au moyen du crible d'Eratosthène
- calculer 10 000 multiplications et 10 000 divisions
- formater des disquettes, copier le contenu de disquettes et de fichiers

Ce banc d'essai était à l'origine destiné à mesurer et comparer les performances de systèmes, ce qui comprend évidemment le micro-processeur et la configuration du système, le système d'opération, les interfaces et le logiciel. Il a été utilisé par ses auteurs pour comparer les divers clones de l'IBM PC. Mais si on fait du système une constante et du logiciel une variable, ce test peut aussi être utilisé pour comparer divers langages (Munro, 1987) ou diverses versions d'un même langage (Dykstra, 1987). Appliqué de cette façon, il mesure alors non seulement la rapidité, mais aussi la puissance de ces logiciels.

Edwards et Hartwig ajoutent qu'en plus du banc d'essai, on doit également tenir compte de la facilité d'utilisation, de la fiabilité, de la compatibilité, des possibilités de mise à jour, du service à la clientèle, ainsi que des commentaires émis par les utilisateurs, les revues, les critiques, les collègues et les vendeurs de bonne réputation.

Munro (1987) suggère qu'un utilisateur choisisse un langage approprié à la tâche et à son propre style de programmation. Il souligne que ce choix doit porter sur un langage (tel que défini par sa syntaxe, c'est-à-dire les objets et les règles) et sur une version de ce langage, c'est-à-dire en tenant compte des outils que ce logiciel met à la disposition de l'utilisateur (éditeur, exécution et correction des programmes, etc...). Selon cet auteur, les caractéristiques à surveiller sont:

- l'interactivité
- la rapidité d'exécution
- la modularité
- l'orientation objet
- la déclaration du type des données
- la transparence
- la lisibilité
- la gestion de la mémoire

Dans une étude de 12 logiciels de statistiques, Lehman

(1987) examine en profondeur les points suivants:

- considérations liées au système: configuration nécessaire, documentation, protection, prix, escompte éducatif, licence d'exploitation
- capacité: nombre et type de variables, données manquantes
- flexibilité et facilité d'utilisation
- caractéristiques de l'éditeur
- interface, entrée/sortie
- transformations: manipulation des données, calculs et opérations, génération de données aléatoires, statistiques descriptives, tests d'hypothèses, analyse multi-variée

L'examen de grilles proposées par divers éducateurs pour l'évaluation de logiciels éducatifs n'a pas permis d'ajouter à cette liste. En effet, ou bien les critères utilisés recoupent ceux qui sont déjà mentionnés ci-dessus, ou bien ils visent l'évaluation des qualités pédagogiques de produits fermés, ce qui ne s'applique pas à la présente situation.

Construction de la grille

Les caractéristiques identifiées plus haut se rattachent au langage lui-même, à l'environnement de programmation ou à des considérations techniques. C'est la structure qui a été donnée à la grille d'évaluation des langages de programmation construite pour les fins de cette recherche.

GRILLE D'EVALUATION

Fiche technique

1. Nom
2. Distributeur
3. Format
4. Equipement requis
5. Documentation
6. Prix

Caractéristiques du langage

7. Représentation des données
   - nombres
   - données non-numériques

8. Structures de données
   - types
   - dimensions

9. Manipulation des données
   - opérations
   - notation
   - priorité
   - puissance
   - précision

10. Structures de contrôle

11. Vocabulaire
   - nombre de mots
   - abréviations
   - congruence avec les termes mathématiques
   - extensibilité
   - importance des habiletés au clavier
   - règles régissant l'utilisation des noms

12. Syntaxe

13. Structure des programmes
   - variables
   - exécution
   - modularité
   - lisibilité et concision
   - transparence
   - orientation objet

Environnement de programmation

14. Gestion des fichiers

15. Gestion de l'espace de travail, de la mémoire

16. Editeur

17. Graphisme

18. Aide à l'usager
   - correction des erreurs
   - pauses
   - système d'aide
   - documentation
   - utilitaires

On remarquera que certaines des caractéristiques suggérées par les auteurs ci-dessus, telles la rapidité, la transférabilité, n'ont pas été retenues parce qu'elles sont moins primordiales dans un contexte d'apprentissage que dans un contexte de construction d'applications ou de conception de logiciel.

CHAPITRE III

ANALYSE

L'évaluation des langages de programmation s'est faite en deux étapes, soit, l'identification et l'analyse des caractéristiques des sept logiciels, au moyen de la grille élaborée précédemment, puis la comparaison et l'évaluation de chacun des langages en regard des objectifs de l'éducation collégiale.

On trouvera dans ce chapitre, les résultats de la première partie de l'évaluation, soit les caractéristiques principales de chaque langage. A ce sujet, quelques remarques sont peut-être de mise.

En premier lieu, il faut se rappeler que cette étude n'a pas été faite dans une perspective de comparaison stricte des performances, ainsi qu'on le fait dans les revues spécialisées en informatique, mais en considérant les possibilités offertes par ces langages lorsqu'ils sont utilisés à des fins didactiques. C'est donc volontairement que certains aspects techniques reliés à la programmation ou aux caractéristiques physiques du système ont été ignorés, et que l'accent a été mis sur les aspects pertinents à l'enseignement et à l'apprentissage des mathématiques.

On remarquera en outre que cette analyse tente de faire ressortir les différences plutôt que les similitudes. Ceci explique pourquoi certains points dont l'étude avait été envisagée au moment de l'élaboration de la grille (v.g. modes d'entrée et de sortie, ou règles de formation des noms) n'ont finalement pas été objets d'évaluation, la recherche ayant montré que tous les langages étudiés offraient des possibilités quasi-identiques à ce sujet.

Enfin, il va de soi que cette analyse ne prétend pas exposer de façon exhaustive toutes les possibilités offertes par chacun des langages en rapport avec les activités mathématiques de niveau collégial, et ceci pour deux raisons évidentes: personne, et certainement pas l'auteure de cette recherche, ne saurait prétendre à l'expertise en programmation dans chacun des langages étudiés; et surtout, il n'y a pas de limite à l'ingéniosité dans la création de nouvelles activités et situations d'apprentissage.

Cette partie de la recherche a voulu tenir compte des applications mises au point par plusieurs professeurs utilisant la programmation sur ordinateur comme soutien à l'apprentissage des mathématiques de niveau collégial, au Collège de Sherbrooke et ailleurs. Elle a également puisé dans les ouvrages de documentation accompagnant les logiciels, et dans la littérature de recherche sur le sujet. Il est souhaitable qu'elle puisse être complétée par les lecteurs de ce rapport: toute contribution à cet égard ne saurait qu'enrichir la problématique à l'étude.

Analyse des caractéristiques des langages

L'évaluation de tous les logiciels a été faite sur un système comprenant un ordinateur de marque Olivetti M-24 (compatible XT), doté d'un micro-processeur 8086 et d'un co-processeur 8087, de 640 Ko de RAM, d'une carte CGA et d'un lecteur de disquettes double, et relié en parallèle à une imprimante Epson FX-85.

On trouvera à l'Appendice B quelques tests effectués dans chacun des langages. Ces pages ne représentent qu'une infime partie des essais effectués au cours de cette recherche. Leur intégration à ce rapport a pour but de donner un exemple concret de l'environnement de programmation offert par chaque langage.

Dans cette étude, les langages sont présentés par ordre alphabétique, en commençant par ceux de 3e génération. Pour chaque langage, on a cherché à utiliser le vocabulaire qui lui est propre. De plus, pour abréger l'énumération des caractéristiques, on a omis de mentionner les items de la grille d'évaluation dans les cas où ils ne s'appliquent pas à un langage particulier (v.g. extensibilité en BASIC), ou quand l'information à ce sujet ne peut être fournie avec certitude (v.g. priorité des opérations en Q'Nial).

On notera enfin que dans chaque cas, c'est une version particulière du langage qui a été soumise à l'étude. Certaines caractéristiques peuvent donc ne concerner que la version étudiée.

APL

APL (A Programming Language) est d'abord et avant tout un système de notation, mis au point pour l'enseignement des mathématiques par K. E. Iverson dans les années 50. Il a servi de point de départ à la conception d'un langage de programmation qui fut implanté d'abord sur ordinateur central, puis sur micro-ordinateur.

APL est connu surtout pour sa compatibilité avec les mathématiques. Il est également connu pour son vocabulaire de symboles particulièrement hermétique. Chaque opération devant être représentée par un symbole unique, il fallut -après avoir épuisé les symboles utilisés traditionnellement en mathématiques- puiser dans l'alphabet grec et même inventer de nouveaux caractères en superposant des caractères du clavier. Il en résulta un langage extrêmement compact en même temps qu'illisible pour les non initiés. Comme la syntaxe d'APL diffère également des règles en usage dans les autres langages, la communauté des programmeurs en est venue à reconnaître d'une part les APL-istes et, d'autre part, le reste du monde.

La structure de données en APL est le tableau. Dans des versions récentes, on a implanté les tableaux emboîtés, ainsi qu'un vocabulaire alternatif faisant usage de mots. Pour les raisons mentionnées plus haut, APL est utilisé principalement pour construire des applications reliées aux mathématiques, à des fins d'enseignement et de recherche.

FICHE TECHNIQUE

1. Nom — APL*PLUS/PC

2. Distribué par — STSC, Inc., a CONTEL Company
2115 East Jefferson St.,
Rockville, Maryland 20852

3. Format — 2 disquettes 5 1/4", 360 Ko.

4. Equipement requis — IBM PC, XT, ou compatible, 256Ko RAM, lecteur de disquettes simple ou double
Moniteur monochrome ou couleur, carte CGA ou Hercules
MS-DOS ou PC-DOS, version 2.0 ou plus
facultatif — Sortie RS-232 ou MODEM intégré.

5. Documentation — 6 guides, publiés par STSC (1983):
APL*PLUS PC SYSTEM; Installation Guide, 46 p.
APL*PLUS PC SYSTEM; User's Guide, 102 p.
APL*PLUS PC SYSTEM; Programmer's manual, 244 p.
APL*PLUS PC SYSTEM; Formating, 116 p.
APL*PLUS PC SYSTEM; File System, 72 p.
APL*PLUS PC SYSTEM; System Functions, Variables and Constants, 358 p.

6. Prix — environ 200.00 $, varie selon la quantité achetée.

CARACTERISTIQUES DU LANGAGE

7. Représentation des objets

Un seul type d'objet: les tableaux emboîtés dont les éléments sont des tableaux emboîtés.
Les atomes constituant ces tableaux sont des données numériques ou alphanumériques.

.données numériques
-Distinction entre nombres entiers et non entiers
-Notation pour nombres non entiers: décimale ou scientifique
-Une seule étendue: $10^{-308} < n < 10^{308}$
-Constante pi ($\circ$); le nombre e s'obtient en faisant (*1).

.données non numériques — Caractère ou suite de caractères.

8. Structures de données

Les tableaux emboîtés qui sont des ensembles de données occupant une position déterminée dans une structure rectangulaire.
Un tableau peut contenir des items de types différents.
Caractéristiques d'un tableau:
-rang: nombre de coordonnées nécessaires pour spécifier ses dimensions.
-dimensions d'un tableau: rang maximum: 63
nombre maximum d'éléments par axe: 32 767
espace mémoire maximum: 64 Ko
Tableaux particuliers: tableau vide
tableau de rang 0: scalaire
tableau de rang 1: vecteur
tableau de rang 2: matrice

9. Manipulation des données

La manipulation des données se fait au moyen de fonctions scalaires ou mixtes, et d'opérateurs.
-Fonctions scalaires et non-scalaires (ou mixtes): s'appliquent à des scalaires ou à des tableaux et retournent des scalaires ou des tableaux.
-Opérateurs: s'appliquent aux fonctions et retournent de nouvelles fonctions.

.fonctions scalaires sur données numériques

-Addition, soustraction, multiplication, division, inverse additif inverse multiplicatif, signe, quotient entier, reste, puissance
-Fonctions maximum, minimum, plus grand entier $\leq n$, plus petit entier $\geq n$, valeur absolue, factorielle, combinaison
sin, cos, tg (angles en radians), arcsin, arccos, arctg, sinh, cosh, tgh, arcsinh, arccosh, arctgh, e^x, a^x, ln x, log x, nombres pseudo-aléatoires.

notation

-Symboles: infixe (dyadique) ou préfixe (monadique)
-Mots: préfixe

évaluation

Les expressions sont exécutées de droite à gauche

précision

-Nombre de chiffres significatifs: maximum 16
-Par défaut: 10

usage de parenthèses

Comme en notation mathématique usuelle. La notation préfixe diminue le nombre de parenthèses nécessaire.

.fonctions scalaires sur données quelconques

-Egalité et comparaison: 6 fonctions
-Logique: et, non et, ou, non ou, non

.fonctions mixtes sur données numériques

Ordre numérique croissant ou décroissant, inversion de matrice, solution d'équation, changement de base

.fonctions mixtes sur données quelconques

-Appartenance, égalité
-Ordre alphabétique ascendant ou descendant
-Structure (description, construction, modification): 15
-Sélection: 3
-Assignation et exécution: 3

.opérateurs sur données numériques

Produit par un scalaire, produit scalaire et matriciel

.opérateurs sur données quelconques — Distribution d'une opération sur les éléments d'un tableau

10. Structures de contrôle
-Branchement conditionnel: si...alors...sinon
CASE
-Branchement inconditionnel: itération
GOTO...LABEL
appel de fonction
-Récursion

11. Vocabulaire
-Deux types de primitives: les fonctions, qui opèrent la manipulation des objets et sont propres à l'interpréteur APL, et les commandes de système, qui gèrent l'espace de travail, les fichiers, et sont des instructions au système APL, propres à la version APL*PLUS.
-Pour les fonctions et opérations, deux vocabulaires: l'un utilisant des symboles et l'autre, des mots.
-Symboles: 65 symboles, dont plusieurs sont utilisés de plus d'une façon, ce qui leur donne des sens différents (monadique ou dyadique, en superposant deux caractères ou en utilisant deux caractères).
-Mots: 205 mots complets ou abréviations
-Commandes de système: 26 commandes

.extensibilité — Possibilité de définir de nouvelles fonctions et opérations ayant le même statut que celles du vocabulaire APL et de les sauvegarder dans un espace de travail.

12. Syntaxe
-Les symboles de citation ' doivent encadrer les données alphanumériques.
-Le symbole ; doit précéder le nom des paramètres d'une fonction
-Les fonctions à un argument sont préfixes, les fonctions à deux arguments sont infixes.
-Les commandes de système doivent être précédées d'une parenthèse.
-Le moins unaire est différent du moins de la soustraction
-Le symbole de l'exponentiation est celui que la plupart des langages utilisent pour la multiplication.

Il y a cinq types d'énoncés:
-exécutables: l'énoncé d'assignation
le branchement
les expressions contenant des fonctions, des opérations, des variables et/ou des données.
-non exécutables: la ligne titre de définition d'une fonction
le commentaire
Aussi, des énoncés composés de plusieurs énoncés simples, séparés par un délimiteur

13. Structure des programmes — La programmation consiste à définir de nouvelles fonctions; un programme est un ensemble structuré de fonctions.
Chaque fonction est une suite d'énoncés comprenant:
-la ligne titre: symbole initial, résultat explicite, nom de la fonction, les arguments de la fonction et les variables, s'il y a lieu.
-le corps de la fonction: lignes numérotées, étiquettes s'il y a lieu, énoncés exécutables et commentaires, s'il y a lieu.
-le symbole final qui permet de protéger, si on le désire, la fonction contre des modifications accidentelles.

.variables
-Les variables créées par assignation de valeur sont globales à l'espace de travail
énoncé d'assignation: nom <- valeur
-Les variables créées en cours de définition d'une fonction sont locales à cette fonction. Cependant elles peuvent être transmises aux fonctions appelées par celle-ci.
-On peut créer une variable dépendante (fonction de une ou plusieurs variables), représentée par son équation algébrique, en utilisant l'énoncé d'assignation pour créer une variable chaîne, ou en définissant une fonction.

.style de programmation
-Programmation modulaire fonctionnelle, restreinte par la possibilité d'utiliser l'énoncé d'assignation pour définir de nouvelles variables à l'intérieur des fonctions.
-Les fonctions peuvent être composées comme en mathématiques.

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers
Types de fichiers:
-fichiers de définition de programmes
-fichiers de données, pouvant être lues en cours d'exécution
-fichier binaire
-espace de travail: toutes les fonctions, variables et programmes présents
-trace: déroulement d'une séance d'interactions avec l'ordinateur
Système puissant pour la gestion des fichiers.

15. Gestion de l'espace de travail
-Allocation dynamique de la mémoire à mesure que des objets sont créés ou détruits.
-Possibilité d'étendre l'espace mémoire
-Le contrôle de l'environnement est effectué au moyen de commandes de système.

16. Editeur
-Editeur de ligne permettant de définir de nouvelles fonctions
-Editeur plein écran pour afficher et corriger les fonctions et variables définies par l'usager (4 pages)
-42 touches contrôlant le mouvement du curseur, l'insertion et la destruction de caractères, la copie et le déroulement de l'écran

17. Graphisme

Le plan graphique contient 1024x1024 points. Des commandes permettent d'afficher à l'écran une partie de ce plan, de sélectionner une fenêtre sur l'écran, d'agrandir le texte ou le dessin, de superposer des images et de masquer des points.
-origine du système d'axes de référence au coin supérieur gauche de l'écran.
-3 modes de résolution: basse, moyenne, haute.
-commandes permettant de dessiner des cercles, ellipses, arcs, diagrammes par secteurs, de colorier des formes, créer des textures à l'aide de caractères graphiques et écrire du texte.
-démo des possibilités graphiques du système dans le fichier d'aide.

18. Aide à l'usager

correction des erreurs
-Distinction entre messages d'erreurs et rapports concernant des erreurs
-Erreurs de syntaxe signalées par un message, un pointeur localise l'erreur
-Possibilité d'insérer des commentaires documentant les programmes

.pauses
Possibilité d'intercaler des pauses dans les programmes pour déceler les erreurs, vérifier la valeur courante des variables, ...

.système d'aide
-Système d'aide donnant de l'information sur le système APL*PLUS, les messages d'erreur, les applications, ...
-Possibilité d'enregistrer des fichiers d'aide adaptés à divers besoins communication d'information, document de référence,...
-Cours interactif sur le langage APL, dans programmes utilitaires

.documentation
Documentation abondante, en anglais seulement.
-APL Is Easy!: initiation au langage et à la programmation APL
-APL*PLUS PC System 1: guide d'installation, guide de l'utilisateur, manuel du programmeur
-APL*PLUS PC System 2: formatage, fichiers, fonctions système, index global
Manuels présentés dans de petits classeurs, bien organisés, faciles à consulter, renseignements complets sur le système mais peu d'exemples et de techniques de programmation.
Feedback demandé à l'usager sur le système, la documentation,...

.utilitaires
-Didacticiel sur APL
-Fichier d'aide
-Démo sur le graphisme
-Programmes utilitaires, applications

BASIC

Le nom BASIC est un acronyme pour Beginner's All-Purpose Symbolic Code. Ce langage fut créé par J. Kemeny et T. Kurtz au College Dartmouth, en 1965. L'intention de ses créateurs était de construire un langage approprié pour l'initiation à la programmation sur ordinateur. A partir de 1976, on développa des versions de BASIC pour micro-ordinateur; certaines étaient suffisamment concises pour être contenues en ROM, ce qui explique pourquoi l'usage de BASIC se répandit si rapidement.

BASIC est un langage dérivé de FORTRAN dont il a hérité plusieurs caractéristiques. Il existe une grande variété de versions BASIC compilées ou interprétées. Les versions compilées offrent l'avantage bien connu de la rapidité, et des implantations nouvelles (ZBASIC, QuickBASIC) permettent la programmation structurée dans un environnement intégré.

Pour les fins de cette recherche, la version GW-BASIC 2.0, distribuée avec les appareils Olivetti a été retenue.

FICHE TECHNIQUE

1. Nom	GW-BASIC, version 2.0
2. Distribué par	Microsoft Corporation
3. Format	1 disquette 5 1/4", 360 Ko.
4. Equipement requis	IBM PC, XT, ou compatible, configuration minimum MS-DOS 2.11 et plus
5. Documentation	MS GW-BASIC interprété; guide de l'utilisateur. Olivetti, (1984), 496 p.
6. Prix	80.00 $ pour GW-BASIC et MS-DOS (prix pour l'éducation)

CARACTERISTIQUES DU LANGAGE

7. Représentation des données

Deux types de données (appelées constantes): les constantes chaînes et les constantes numériques.

.données numériques

Cinq types de constantes numériques:
-entières; étendue: $-32768 < n < 32767$
-à point décimal fixe ou flottant; étendue: $10^{-38} < n < 10^{38}$
-octales et hexadécimales
Le nombre e s'obtient en faisant EXP (1).

.données non numériques

Chaînes de caractères (longueur maximale: 255 caractères)

8. Structures de données

Les tableaux, dont les éléments sont des constantes de même nature (numériques ou chaînes de caractères).
Dimensions: par défaut: 1 dimension, 10 éléments
sur déclaration: jusqu'à 255 dimensions, 32767 éléments par dimension (soumis aux contraintes du système)
L'accès aux éléments se fait de façon directe.

9. Manipulation des données

La manipulation des données se fait au moyen d'instructions, de commandes et de fonctions:
-instructions: généralement utilisées dans un programme
-commandes: généralement utilisées en mode immédiat, pour la gestion des programmes et du système.
-fonctions: exécutent des opérations sur les constantes.

.opérations sur données numériques

-Addition, soustraction, multiplication, division, inverse additif, signe, quotient entier, reste, partie entière, arrondissement, exponentiation, racine carrée
-Fonctions valeur absolue, sin, cos, tg (angles en radians), arctg, e^x, ln x, nombres pseudo-aléatoires

notation
-Infixe pour les symboles: +, -, *, /, ^
-Préfixe pour les mots

priorité
1. Exponentiation
2. Moins unaire
3. Multiplication, division à virgule flottante
4. Division de nombres entiers
5. Arithmétique modulo
6. Addition, soustraction
7. Autres fonctions mathématiques
8. Opérateurs relationnels (=, <>, <, >, <, >)
9. Opérateurs logiques (NOT, AND, OR, XOR, IMP, EQV)

précision
-Nombres décimaux à simple précision: 7
-Nombres décimaux à double précision: 16
-Par défaut: 7

.opérations sur données non numériques
-Logiques: voir ci-haut
-Egalité et comparaison, ordre alphabétique (voir ci-haut)
-Mesure: 1
-Recherche et sélection: 4
-Construction: 4
-Conversion: 9

.opérations sur structures de données
-Déclaration du nom et des dimensions d'un tableau
-Construction de boucles pour appliquer les autres opérations sur les éléments des tableaux.

10. Structures de contrôle
-Branchement conditionnel: (IF...condition...THEN...instruction...ELSE...instruction
(WHILE...condition...instruction...WEND)
-Branchement inconditionnel: (GOTO...no de ligne)
(FOR...NEXT...)
appel de sous-routine (GOSUB)

11. Vocabulaire
-176 mots réservés (27 commandes, 91 instructions, 59 fonctions)
-28 caractères spéciaux (symboles)
Remarques: -plusieurs opérateurs accomplissent la même opération sur des objets de types différents
-plusieurs opérateurs de conversion de type.

12. Syntaxe
Symboles particuliers:
-variables: entières: nom terminé par %
décimales: à simple précision: nom terminé par !
à double précision: #
alphanumériques: nom terminé par $
-chaînes de caractères: entre guillemets
-arguments des fonctions: entre parenthèses
-symbole = sert pour l'assignation de variables et le test d'égalité
Plusieurs types d'énoncés, chacun ayant une ou des syntaxes différentes.

13. structure des programmes — Un programme est une suite d'énoncés numérotés de 0 à 65529, et dont le dernier est END
Un seul énoncé par ligne, ou plusieurs, séparés par :

.variables
-toutes les variables créées par assignation de valeur sont globales.
énoncé d'assignation: nom = valeur
-les programmes n'ont pas de paramètres.
-l'instruction DEF FN permet de créer une fonction de une ou plusieurs variables, représentée par son équation algébrique. Utilisable à l'intérieur de la définition d'un programme seulement.

.style de programmation — Programmation procédurale, structurée mais non modulaire

---

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers — Types de fichiers:
-fichier de programme: contient le programme en mémoire lors de la commande d'enregistrement
-fichier protégé: enregistré dans un format binaire codé.
-fichiers de données à accès direct ou séquentiel; contiennent des données qui peuvent être lues pendant l'exécution d'un programme.
Les programmes qui excèdent la capacité de mémoire peuvent être divisés en segments logiques.

15. Gestion de l'espace de travail
-Concept inexistant: tout le travail en mode direct et la définition des programmes se font dans l'éditeur
-L'usager doit déclarer les dimensions des objets qu'il crée.
-Contrôle de l'affectation mémoire par l'usager

16. Editeur
-Editeur d'écran pleine page
-20 touches contrôlent le mouvement du curseur, l'insertion et la destruction de caractères, la copie et le déroulement de l'écran

17. graphisme
-3 modes d'écran graphique, 16 commandes:
résolution moyenne: 320x200 pixels,
haute résolution: 640x200
très haute résolution: 640x400
-2 systèmes de référence: coordonnées absolues, point (0,0) au coin supérieur gauche de l'écran; coordonnées relatives (position du pixel par rapport au dernier pixel repéré.
-Possibilité d'afficher des points, de tracer des figures, de colorier haute résolution

18. **aide à l'usager**

correction des erreurs	-77 messages, dont 43 aident à la programmation et à la correction des erreurs et 34 donnent de l'information sur le système et ses limites. -Possibilité d'insérer des commentaires pour documenter les programmes
.pauses	Possibilité d'intercaler des pauses dans les programmes pour déceler les erreurs, vérifier la valeur courante des variables, ...
.documentation	Un seul manuel d'utilisation, en français. MS GW-BASIC interprété: guide de l'utilisateur, un manuel bien structuré contenant les renseignements usuels sur la programmation en BASIC, l'éditeur, le calcul, les fichiers, le graphisme, la communication asynchrone, un glossaire des commandes, instructions et fonctions, l'interprétation des messages d'erreur. Manquent: des renseignements sur la configuration du système, des exemples de programmation explicite, un index.

Logo

Logo est un langage développé en 1968 par W. Feurzig, assisté de S. Papert, M. Bloom, R. Grant et C. Solomon, chez Bolt, Beranek & Newman Inc., sous commandite de la National Science Foundation. Des recherches ultérieures furent menées au Laboratoire d'intelligence artificielle ainsi qu'à la Division des études et recherches en éducation du Massachussetts Institute of Technology.

Le nom Logo n'est pas un acronyme. Il a été imaginé par Feurzig, à partir du mot grec "logos" signifiant mot, pensée. L'intention de ses créateurs était de construire un environnement d'apprentissage permettant de présenter l'ordinateur et la programmation à de jeunes enfants.

Logo est issu de LISP, langage créé vers la fin des années 50 par J. McCarthy pour les besoins de l'intelligence artificielle. Il existe une grande variété de logiciels portant le nom de Logo. A une extrémité, on retrouve tous les logiciels vendus sous le nom de Logo, mais qui n'en sont que des sous-ensembles ne contenant pas les caractéristiques fondamentales de ce langage; à l'autre extrémité, des versions avancées tournant sur MacIntosh et exploitant les avantages de cette technologie; on pense plus particulièrement à TLC-Logo, une version dans laquelle sont implantés certains concepts de l'orientation objet, au Object Logo de Coral Software, et aux dernières versions de Logo pour MacIntosh.

Entre ces extrêmes, divers produits se rattachent à deux dialectes principaux: celui conçu par Logo Computer Systems Inc., tournant sur divers appareils dont Apple et IBM et connu sous le nom de Logo LCSI; et celui conçu à MIT, tournant principalement sur les appareils IBM et Commodore, distribué par Krell ou Terrapin et connu sous le nom de Logo MIT.

Les recherches précédant ce projet avaient permis d'expérimenter trois versions différentes de Logo:
-le Logo pour Apple, version française, distribuée par LCSI;
-Logo PC, version française (MIT) tournant sur IBM et distribuée par les Editions Turgeon de Montréal;
-Logo pour IBM, version 1.0, distribuée par LCSI mais non disponible en français.

La dernière s'étant avérée la plus performante des trois dans la situation d'apprentissage des mathématiques, c'est celle-là qui fut retenue pour évaluation dans le cadre de cette recherche.

FICHE TECHNIQUE

1. Nom — IBM PC Logo, version 1.0

2. Distribué par — Logo Computer Systems, Inc.,
via les centres de vente IBM

3. Format — 1 disquette 5 1/4", 360 Ko.

4. Equipement requis — IBM PC, XT, ou compatible, 128Ko RAM, lecteur de disquettes simple ou double
Moniteur monochrome ou couleur, carte CGA
MS-DOS ou PC-DOS, version 2.0 ou plus
facultatif — Adaptateur de contrôle pour jeux (manette, bouton)

5. Documentation — Logo: Programming with Turtle Graphics, IBM (1983)
IBM Personal Computer Logo Reference, IBM (1983), 358 p.

6. Prix — non disponible

CARACTERISTIQUES DU LANGAGE

7. Représentation des données

Deux types d'objets: les mots et les listes. Les nombres sont des mots d'un type particulier. Les éléments indivisibles sont le nombre et le caractère.

.données numériques — Le nombre est un mot dont les caractères sont des chiffres; il est considéré comme une entité indivisible et est évalué à lui-même.
-Pas de distinction entre nombres entiers et non entiers
-Deux notations: à point flottant, avec ou sans point (partie décimale ou non)
notation scientifique
-Une seule étendue: $10^{-9999} < n < 10^{9999}$
-Constante: pi; le nombre e s'obtient en faisant EXP 1.

.données non numériques — Le mot est une suite de caractères précédée de guillemets.

8. Structures de données — Les listes, dont les éléments sont des nombres, des mots ou des listes.
Les listes peuvent contenir des éléments de types différents.
Listes particulières: liste vide
phrase: liste dont les éléments sont des mots
L'accès aux éléments d'une liste se fait de façon séquentielle ou directe.

9. Manipulation des données — La manipulation des données se fait au moyen de procédures appelées commandes ou opérations.
-Commandes: produisent un effet
-Opérations: retournent un objet

.opérations sur données numériques

-Addition, soustraction, multiplication, division, inverse additif, quotient entier, reste, partie entière, arrondissement, puissance, racine carrée
-Fonctions sin, cos, (angles en degrés), arctg, e^x et ln x, nombres pseudo-aléatoires
Pour appliquer ces opérations sur les données numériques contenues dans une liste, il faut définir de nouvelles opérations.

notation

-Infixe ou préfixe pour les symboles
-Préfixe pour les mots

priorité

1. moins unaire
2. multiplication et division
3. addition et soustraction
4. autres opérations mathématiques, incluant celles définies par l'usager
5. opérateurs relationnels < et >
6. égalité (= ou EQUALP)
7. opérateurs logiques (AND, OR, NOT)
8. commandes (PRINT, SHOW, ...)

évaluation

Les expressions sont parcourues de gauche à droite par l'analyseur, et les opérations sont effectuées dès que les entrées appropriées ont été identifiées.

précision

Nombre de chiffres significatifs: entre 5 et 1000
entre 3 et 100 pour certaines primitives v.g. sin
par défaut: 10

usage de parenthèses

Comme en notation mathématique usuelle; l'utilisation des opérations en mode préfixe diminue la quantité de parenthèses nécessaires.
L'utilisation de parenthèses permet aussi de donner plusieurs entrées à une primitives, à condition que l'opération soit associative, v.g. (SUM 2 43 -51)

.opérations sur données quelconques

-logique: AND, OR, NOT
-égalité et comparaison: =, <, >
-appartenance: 1 opération
-mesure: 1 opération
-identification du type d'objet: 4 opérations
-sélection: 5 opérations
-construction: 5 opérations

10. Structures de contrôle

-branchement conditionnel (IF...condition...[action1] (sinon) [action2]
(TEST...condition
IFTRUE...action1
IFFALSE...action2)
-branchement inconditionnel (itération (REPEAT n [action])
(GO...LABEL)
appel de sous-procédures
-récursion

11. Vocabulaire

-224 mots et symboles (189 primitives préfixes, 7 symboles, 15 primitives à point, et 13 mots réservés)
-33 abréviations

.extensibilité — Possibilité de définir des mots ayant le même statut que ceux du vocabulaire de base et de les sauvegarder dans un fichier.

.traduction — Possibilité de traduire le vocabulaire de base dans une autre langue au moyen de la primitive COPYDEF ou par d'autres moyens.

12. Syntaxe

Symboles particuliers:
-le symbole " doit précéder immédiatement les mots, ainsi que tous les noms: de procédures, de variables et de fichiers.
-le symbole : doit précéder immédiatement la valeur d'une variable
-les listes doivent être encloses entre crochets carrés []
-espaces facultatifs entre les nombres et les opérations (sauf moins unaire)

Un seul type d'énoncé: l'exécution de procédures
Nécessité de distinguer entre commande et opération, expression et instruction
-commande: procédure pouvant avoir des paramètres, et produisant un effet explicite
-opération: procédure pouvant avoir des paramètres, et retournant un résultat implicite qui est un objet Logo (nombre, mot ou liste).
-expression: nombre ou opération suivie de ses entrées
-instruction: commande suivie du nombre d'expressions requises comme entrées à cette commande

13. Structure des programmes

Programmer consiste à écrire de nouvelles procédures, qui sont des commandes ou des opérations.

```
Syntaxe:  TO   nom   paramètres
          suite d'instructions
          END
```

Un programme est un ensemble structuré de procédures commandées par une procédure maîtresse.

.variables
-Les variables créées par assignation de valeur sont globales à l'espace de travail.
énoncé d'assignation: MAKE "nom valeur
-Les variables créées à l'intérieur d'une procédure sont globales, à moins d'être déclarées explicitement locales.
-les paramètres d'une procédure sont locaux à cette procédure, mais peuvent être transmis aux procédures appelées par celle-ci.
-on peut créer une variable dépendante (fonction de une ou plusieurs variables), représentée par son équation algébrique en utilisant l'énoncé d'assignation ou en définissant une procédure opération.

.style de programmation
-Programmation procédurale, modulaire
-Possibilité de définir des fonctions et de les composer comme en mathématiques.
-Les procédures peuvent être manipulées comme des objets au moyen des primitives TEXT et DEFINE, et être données en entrées à d'autres procédures.

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers

Types de fichiers:
-espace de travail: contient les procédures et les variables définies au cours d'une séance de travail, en mode de définition ou dans l'éditeur.
-trace: contient toute l'interaction qui s'est déroulée pendant une séance de travail.
-fichier de données: contient des données qui peuvent être lues pendant l'exécution d'un programme.
-fichier dessin: contient une image de ce qui était affiché à l'écran (texte ou dessin).
-fichier binaire: pour l'enregistrement de routines écrites en langage binaire.
23 primitives pour créer, ramener, détruire des fichiers (pas de touches de fonctions)
Pour modifier le contenu d'un fichier, il faut le ramener dans l'espace de travail, le modifier puis l'enregistrer à nouveau.

15. Gestion de l'espace de travail

-4 modes d'interaction: exécution directe, définition de procédures, édition et graphique
-256 Ko de RAM utilisés directement si disponibles (164 Ko à l'usager)
-Possibilité d'étendre l'espace-mémoire au moyen de .EXAMINE et .DEPOSIT
-Allocation dynamique de la mémoire à mesure que des objets sont créés ou détruits
-Possibilité de recyclage de la mémoire
-27 primitives gérant l'espace de travail

16. Editeur

-Editeur intégré au système
-L'écran de l'éditeur est conservé jusqu'à effacement
-16 clés contrôlent le mouvement du curseur, l'insertion et la destruction de caractères, la copie et le déroulement de l'écran

17. Graphisme

Trois modes d'écran:
-texte: 25 lignes sur 40 ou 80 colonnes
-graphique: 240 x 320 pixels
-mixte: écran graphique (20 lignes) écran texte (5 lignes)
Micro-monde de la tortue:
-géométrie locale ou cartésienne
-18 primitives gérant la position et l'orientation
-25 primitives gérant l'écran
-18 primitives gérant le dessin et la couleur
-un seul mode de résolution

18. Aide à l'usager

.correction des erreurs — -50 messages dont 22 aident à la programmation et à la correction des erreurs et 28 donnent de l'information sur le système et ses limites.

.pauses — Possibilité d'intercaler des pauses dans les programmes pour déceler les erreurs, vérifier la valeur courante des variables, ...

.documentation — Deux manuels d'utilisation, présentés en cartables, en anglais seulement:
-Logo: Programming with Turtle Graphics. Manuel d'initiation à la programmation en Logo. Nombreux exemples.
-IBM Personal Computer Logo Reference: références plus complètes sur l'installation, la syntaxe, l'éditeur, les fichiers, la manipulation des nombres; glossaire alphabétique avec exemples et explications, utilitaires, assembler, organisation de la mémoire, index.

.utilitaires — Un fichier TOOLS contient 58 procédures pouvant être ramenées en mémoire en même temps que Logo. Les plus utiles:
-valeur absolue
-insertion de commentaires
-structures de contrôle: FOREVER, WHILE
-distribution d'une opération sur une liste
-pour identifier le rang d'un item dans une liste
-primitives graphiques

Q'Nial

Nial est un langage interactif, développé par une équipe internationale sous la direction des professeurs M. Jenkins et T. More, à l'Université Queen's de Kingston, Ont., au début des années 80. Nial est un acronyme pour Nested Interactive Array Language; on signale également la ressemblance de ce mot avec le nom du chef viking Njal, héros légendaire danois dont l'aptitude à résoudre des problèmes était remarquable. Q'Nial est une version interprétée, développée pour l'ordinateur central et les micros 16 bits.

Considérant que le langage APL ne permettait pas un traitement de l'information symbolique assez puissant pour la manipulation des connaissances au sens de l'intelligence artificielle, les concepteurs de Nial travaillèrent à étendre les concepts utilisés en APL à la création d'un langage permettant un degré élevé de sophistication dans la manipulation des objets. A la base de Nial, on retrouve donc la théorie des tableaux rectangulaires emboîtés. Misant sur la puissance des ordinateurs personnels de demain, les concepteurs de ce nouveau système utilisèrent le langage C afin d'en augmenter la généralité et la transférabilité.

Il en résulta un langage qui, pour reprendre les termes de Jenkins (1985), "opère sur les listes comme LISP, sur les tableaux à dimensions multiples comme APL; permet la programmation structurée comme Pascal, effectue les calculs scientifiques et mathématiques comme FORTRAN, et autorise l'usage du vocabulaire anglais comme COBOL".

On le dit utilisé pour la construction de prototypes ainsi que dans les calculs reliés à la recherche en génie et en intelligence artificielle. En éducation, il a fait l'objet de plusieurs expérimentations (Glasgow, Jenkins, et Hendren, 1986), au cours desquelles on a construit des micro-mondes destinés à supporter l'apprentissage de l'algèbre et de la logique. Il n'en fallait pas davantage pour que l'on s'y intéresse dans la situation d'enseignement et d'apprentissage des mathématiques de niveau collégial.

FICHE TECHNIQUE

1. Nom — Q'Nial: Educational Version 3.06, 1985

2. Distribué par — Nial Systems Limited
20 Hatter St.,
Kingston, Ont. K7M 2L5

3. Format — 4 disquettes 5 1/4", 360 Ko.

4. Equipement requis — IBM PC, XT, AT, ou compatible, 512 Ko RAM, lecteur de disquettes simple ou double
Moniteur monochrome ou couleur, adaptateur graphique
MS-DOS ou PC-DOS, version 2.0 ou plus
facultatif — Co-processeur mathématique 8087 ou 80287

5. Documentation — Q'Nial Tutorial, Nial Systems Ltd (1985), 182 p.
Q'Nial Reference Manual, Nial Systems Ltd (1985), 461 p.
Q'Nial User's Guide, Nial Systems Ltd (1986), 322 p.

6. Prix — License éducative: 500.00 $
Documentation: 65.00 $

CARACTERISTIQUES DU LANGAGE

7. Représentation des données

Un seul type d'objet: les tableaux rectangulaires emboîtés dont les éléments sont des tableaux rectangulaires emboîtés.
L'élément atomique de cette structure récursive est le tableau qui a lui-même comme seul élément.
Cet atome peut être un nombre entier, un réel, un complexe, une valeur de vérité, un caractère, un mot, une faute.

.données numériques

Distinction entre nombres entiers et réels.
-entiers: étendue: -32 768 < n < 32767
-réels: notation décimale ou scientifique
étendue: $10^{-308} < n < 10^{308}$
L'entier 3 n'est pas égal au réel 3.0
-complexes: notation: ajb
-constantes: pi et j; le nombre e s'obtient en faisant EXP 1.

.données non numériques

-valeur de vérité: les lettres 1 (Vrai) et 0 (Faux)
-caractère: n'importe quel caractère du clavier; le système ne différencie pas les majuscules des minuscules. ('a, '5, '&)
-mot: suite de caractères ne contenant pas d'espace ou autre délimiteur. (*nombre)
-faute: élément d'information généré par le système ou l'usager et signalant une erreur ou des conditions spéciales; précédé d'un point d'interrogation (?invalid assignment)

8. Structures de données

Les tableaux rectangulaires emboîtés qui sont des ensembles de données occupant une position déterminée dans une structure rectangulaire.
Un tableau peut contenir des items de types différents.

-propriétés d'un tableau: valence: nombre d'axes
forme: nombre d'éléments par axe
compte: nombre d'items au premier niveau d'emboîtement

-dimensions d'un tableau: nombre maximum d'éléments: 32767 (8188 si nombres réels, 4192 si nombres complexes)
espace mémoire maximum: 64 Ko

-tableaux particuliers: tableau vide
singleton: tableau de dimension 0 (scalaire)
liste: tableau de dimension 1 (vecteur ligne)
colonne: tableau de dimension 1 (vecteur colonne)
chaîne de caractères: liste de caractères pouvant contenir des espaces et autres délimiteurs, encadrée par des symboles de citations. ('3 est un entier')
chaîne de bits: suite de 1 et 0 correspondant aux valeurs de vérité Vrai ou Faux (1110010) (vecteur booléen)
solitaire, paire, triplet: listes contenant un seul, deux ou trois items.
table: tableau de dimension 2 (matrice)

Ce langage supporte donc aussi la structure de liste. L'accès aux éléments d'une liste se fait de façon séquentielle ou directe.
Les tableaux peuvent être affichés à l'écran au moyen de diagrammes illustrant leur structure.

9. Manipulation des données

La manipulation des données se fait au moyen des opérations et des transformateurs.

-opérations: s'appliquent sur des tableaux. Elles sont "pervasive" (s'appliquent aux atomes d'un tableau), "pervading" (s'appliquent aux tableaux atomiques des items d'un tableau) ou "binary pervading" (s'appliquent aux tableaux atomiques correspondants de deux tableaux ayant la même structure).

-transformateurs: s'appliquent sur des opérations (v.g. EACH, ITERATE) et distribuent une opération sur les items d'une liste de listes.

.opérations sur données numériques

-Addition, soustraction, multiplication, division, inverse additif, inverse multiplicatif, signe, quotient entier, reste, puissance

-Fonctions maximum, minimum, plus grand entier $< n$, plus petit entier $> n$, valeur absolue, racine carrée, partie réelle, partie imaginaire, conjugué, phase sin, cos, tg (angle en radians), arcsin, arccos, arctg, sinh, cosh, tgh, e^x, ln x, log x, nombres pseudo-aléatoires

notation

Infixe ou préfixe avec deux arguments, sinon préfixe

évaluation

-De gauche à droite pour la notation infixe
-De droite à gauche pour la notation préfixe

précision

Nombre de chiffres significatifs: maximum: 16
par défaut: 6

usage de parenthèses

Comme en notation mathématique usuelle. La notation préfixe diminue le nombre de parenthèses nécessaire.

.opérations sur données quelconques

-Logique: AND, OR, NOT
-Egalité: égalité, inégalité, disjonction
-Appartenance: appartenance, non appartenance, inclusion, égalité
-Comparaison: 8 opérations
-Identification du type: 10 prédicats
-Conversion: 8 opérations
-Structure (description, construction et modification): 26 opérations
-Sélection: 17 opérations
-Insertion: 3 opérations
-Recherche: 3 opérations
-Transformation: 13 opérations

10. Structures de contrôle

-Branchement conditionnel: IF...THEN...ELSE...ENDIF
CASE...FROM...ELSE...ENDCASE

-Branchement inconditionnel: itération: FOR...WITH...DO...ENDFOR
REPEAT...UNTIL...ENDREPEAT
WHILE...DO...ENDWHILE
appel de sous-procédure

-Récursion

11. Vocabulaire

319 symboles et mots ayant le statut d'expression (31), d'opération (218), de transformateur (19), ou de mot réservé (51).

.extensibilité

Possibilité de définir de nouvelles expressions, opérations et transformateurs qui seront enregistrés dans un fichier et pourront être ramenés au besoin.

12. Syntaxe

Symboles particuliers:
-l'accent grave doit précéder les caractères
-les guillemets doivent précéder les mots et les noms de fichier, mais pas les variables (appelées identificateurs)
-le point d'interrogation précède les fautes (messages d'erreurs)
-les symboles de citation doivent encadrer les chaînes de caractères.
-le symbole # doit précéder les commentaires;
-le symbole point-virgule doit séparer les expressions tapées sur une même ligne; il supprime l'évaluation d'une expression.
-le moins de la soustraction, en notation infixe, doit être séparé du second argument par un espace. Impossibilité d'écrire une expression de la forme a - b - c.
La syntaxe des différents construits du langage est illustrée par des schémas dans le Guide de l'usager.

13. Structure des programmes

La programmation consiste à définir de nouvelles expressions, opérations ou transformateurs. Tous retournent explicitement le dernier résultat obtenu.
-expression: énoncé valide décrivant un objet sémantique; aucun paramètre
syntaxe: Nom IS (suite d'expressions)
-opération: expression à un ou plusieurs paramètres, s'appliquant à des tableaux et retournant un tableau.

syntaxe: Nom IS OPERATION liste de paramètres (suite d'expressions)
-transformateur: expression à un ou plusieurs paramètres, s'appliquant sur des opérations et retournant une opération.
syntaxe: même que celle de l'opération

.variables

-Les variables créées par assignation de valeur sont globales à l'espace de travail
énoncé d'assignation: nom GETS valeur
ou: nom := valeur
-Les variables créées en cours de définition d'une expression sont locales à cette expression.
-Les paramètres d'une opération ou d'un transformateur sont locaux à cette expression mais peuvent être transmis aux expressions appelées par celle-ci.
-Il est possible de déclarer 'externes' des variables ou des expressions qui seront définies ultérieurement.
-On peut créer une variable dépendante (fonction de une ou plusieurs variables) représentée par son équation algébrique, en créant une variable chaîne au moyen de l'énoncé d'assignation, ou en définissant une nouvelle opération.

.styles de programmation

Ce langage supporte plusieurs styles de programmation:
-la programmation procédurale, décrite ci-dessus. L'idée de modularité est renforcée par le fait que les variables créées en cours d'exécution de programme sont locales.
-la programmation fonctionnelle à la Backus, qui utilise des listes d'opérations (atlas) et de transformateurs (galaxies) pour définir des fonctions sans variables.
-la programmation logique à la manière de Prolog.
-quelques aspects de la programmation orientée objet.
-la construction de bases de données.

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers

Types de fichiers:
-espace de travail: contient les variables et les expressions définies au cours d'une séance de travail.
-trace: contient toute l'interaction qui s'est déroulée pendant une séance de travail.
-fichier de définition: contient toutes les variables et les expressions définies dans l'éditeur.
-fichier de données
-fichier dessin: contient une image de ce qui est affiché à l'écran
L'accès aux fichiers est séquentiel; on peut ajouter de l'information au début ou à la fin.

15. Gestion de l'espace de travail

-Espace de travail, au départ: 130 Ko
libres: 112 Ko
-Possibilité de modifier l'espace mémoire
-L'espace de travail permet l'accès à un éditeur du choix de l'usager, à un éditeur enregistré en fichier, au système d'opération, à la bibliothèque de programmes.

-Il est également possible d'accéder à l'interpréteur lexical et syntaxique, ainsi qu'à la table des symboles.
-Le contrôle de l'environnement est assuré par des opérations de type SET.
-L'allocation de mémoire est automatique.

16. Editeur

-Aucun éditeur intégré au système; en mode direct, impossibilité de revenir sur les caractères de la ligne courante sans les effacer.
-L'usager peut utiliser un éditeur de son choix, ou celui qui est contenu dans un fichier de la bibliothèque de programmes.

17. Graphisme

2 modes graphiques:
-alphanumérique: 25 lignes sur 80 colonnes
-graphisme par pixels: 200 x 300 pixels
Aussi: possibilité d'utiliser le micro-monde de la tortue proposé dans un des fichiers de la bibliothèque. Ce programme n'utilise pas la syntaxe Logo mais contient les primitives essentielles à la construction de procédures semblables à celles de Logo. Système d'axes cartésiens dont l'origine est au centre de l'écran et le sens de rotation positif en sens contraire des aiguilles d'une montre.

18. Aide à l'usager

.correction des erreurs

Messages d'erreurs regroupés de la façon suivante:
-erreurs de syntaxe repérées par l'analyseur lexical et syntaxique: 47
-erreurs repérées en cours d'exécution: 147
-erreurs en mode graphique: 33
-erreurs externes: 18
Possibilité d'insérer des commentaires permettant de documenter les programmes. L'affichage des définitions utilise une forme canonique et l'indentation aide au dépistage des erreurs.

.pauses

Possibilité d'intercaler des pauses dans les programmes pour déceler les erreurs, vérifier la valeur courante des variables, ...

.aide

-Fichier d'aide non disponible sur IBM PC
-Didacticiel sur la programmation en Q'Nial dans les utilitaires.

.documentation

Trois manuels d'utilisation, reliure cartonnée, en anglais seulement. (La consultation en est difficile car les livres ne restent pas ouverts.)
-Tutorial: les rudiments de la programmation en Q'Nial, expliqués de façon à favoriser l'apprentissage interactif; exercices et réponses, glossaire du vocabulaire informatique, liste des utilitaires, index.
-User's Guide: installation; vocabulaire Q'NIAL; caractéristiques du système; techniques et exemples de programmation; utilitaires; index.
-Reference Manual: explications plus rigoureuses des divers aspects du langage, dictionnaire détaillé des expressions pré-définies en Q'Nial, y compris leur classe, usage, syntaxe, définition et les expressions équivalentes, le cas échéant; index.
Feedback demandé à l'usager sur le système, la documentation, les problèmes rencontrés, les applications construites.
Commentaires occasionnels sur les limites du système Q'Nial.

.<u>utilitaires</u>	-Didacticiel sur Q'Nial -Applications en comptabilité -Calculs statistiques: traitement des données, nombres de Stirling, corrélations, mesures de tendance centrale et de dispersion, tabulation, statistiques non paramétriques, régression, analyse de la variance. -Système auteur permettant de créer des tutoriels sur des sujets variés -Différents utilitaires écrits par les usagers de Q'Nial

CAL

CAL: The Mathematics Discovery Language est un logiciel intégré développé par Bluejay Lispware en 1986, et destiné, comme son nom l'indique, à promouvoir l'apprentissage des mathématiques par la découverte.

CAL inclut plusieurs sous-ensembles contenant respectivement des outils permettant le calcul numérique et symbolique, la manipulation des propositions et le graphisme. Il offre quelques possibilités de programmation à la manière de Logo et de Prolog, ainsi qu'un langage-auteur permettant la construction de nouveaux environnements destinés à supporter des apprentissages particuliers. Comme tel, il mérite le nom d'environnement d'apprentissage, puisqu'il permet un type d'exploration relativement informel et flexible de problèmes ouverts.

La structure générale de CAL est hiérarchique. Elle est expliquée au moyen de la métaphore suivante: on peut considérer CAL comme une maison; dans cette maison, les états sont des pièces; les environnements sont l'ameublement qu'on décide de placer dans ces pièces; enfin, CAL possède un choix de sous-sol: ce sont les deux modules qui modifient légèrement l'environnement de travail. L'utilisation de CAL commence par le choix d'un module, d'un état et d'un ou de plusieurs environnements, ce qui crée un espace où il sera possible d'exécuter certaines tâches. L'usager peut également créer de nouveaux environnements.

En plus de supporter l'apprentissage des mathématiques, CAL permet à l'usager de se familiariser avec des concepts et des tachniques utilisés en intelligence artificielle, notamment les systèmes experts, le "pattern-matching", ou le chaînage arrière et avant. CAL est un logiciel écrit en LISPER LISP, dialecte lui-même écrit en assembler et permettant la manipulation des symboles.

FICHE TECHNIQUE

1. Nom	CAL: The Mathematics Discovery Language (1986)
2. Distribué par	Bluejay Lispware Box 1708 Glendora, California 91740
3. Format	4 disquettes 5 1/4", 360 Ko.
4. Equipement requis	IBM PC ou compatible, moniteur monochrome ou à affichage graphique, 512 Ko RAM, lecteur de disquettes simple ou double, PC-DOS ou MS-DOS, version 2.0 ou plus
5. Documentation	CAL: The Mathematics Discovery Language, Bluejay Lispware (1986), 325 p. CAL: An Introduction for Students, Blujay Lispware, 86 p.
6. Prix	228.00 $U.S.

CARACTERISTIQUES DU LANGAGE

7. Représentation des données

Deux types d'objets: les nombres et les expressions algébriques

.nombres	-Nombres entiers -Nombres décimaux à point fixe ou flottant -Nombres rationnels sous forme de quotient d'entiers étendue des nombres réels: $10^{-255} < n < 10^{255}$ -Nombres complexes notés sous la forme a+bi -Constantes: e (E), pi (PI), i (I)
.expressions algébriques	Expressions, fonctions et relations, équations et systèmes d'équations contenant des variables qui doivent avoir été déclarés au préalable, mais ne doivent pas nécessairement avoir reçu de valeur.

8. Structures de données — Aucune

9. Manipulation des données — La manipulation des données se fait au moyen de commandes et de fonctions. La plupart de ces opérations s'appliquent indifféremment sur les données numériques ou alphanumériques.

.opérations arithmétiques	Addition, soustraction, multiplication, division, inverse additif, exponentiation quotient entier, reste
.autres fonctions	Partie entière, valeur absolue, conversion degrés à radians, sin, cos, plus petit que, conjugué, racine carrée, racine cubique, racine

notation	-Symboles (mode infixe), mots-clés (mode préfixe) -Possibilité de définir une opération infixe symbolisée par # -L'arithmétique rationnelle est invoquée en sélectionnant Algebraic Evaluation dans le menu.
priorité	1. Exponentiation (^) 2. Division (/) 3. Multiplication (*) 4. Opération définie par l'usager (#) 5. Soustraction (-) 6. Addition (+) 7. Plus petit que (<)
évaluation	L'évaluation se fait de gauche à droite (sauf l'exponentiation), en suivant l'ordre de priorité ci-dessus.
précision	Maximum de 255 chiffres significatifs, selon la documentation, mais le système plante bien avant. Par défaut: 1 décimale
.opérations algébriques	Simplification, division synthétique des polynômes, solution d'équations et de systèmes d'équations (non-nécessairement linéaires)
.calcul infinitésimal	Limite, dérivée partielle, dérivée d'ordre n, approximation des racines par méthode de Newton, développement en série de Taylor, sommes de Riemann, solution d'équations différentielles ordinaires d'ordre 1 et 2: approximations numériques seulement, et non résultats exprimés sous forme symbolique.
.calcul vectoriel	Graphes
.logique	Construction de bases de données et étude de relations
10. Structures de contrôle	-Branchement conditionnel (IF...condition...THEN...action...ELSE...) -Dans le sous-ensemble Logo: itération (REPEAT n [action]) récursion
11. Vocabulaire	-Le manuel donne une liste de 24 commandes et de 21 fonctions. -Le langage comporte aussi toutes les commandes sélectionnées à l'aide des menus. la différence entre commandes et fonctions n'est pas claire.
12. Syntaxe	-Le symbole ']' doit terminer toutes les instructions. -Les parenthèses sont largement utilisées dans la définition des procédures Logo (à la manière LISP)
13. Structure des programmes	On ne peut parler de véritable programmation que dans les environnements PLOTTER et LOGIC, ainsi que dans FILER. Dans les deux premiers, la structure des programmes est similaire à celle qui prévaut dans les langages Logo et PROLOG, quoique non identique. Dans FILER, qui est en quelque sorte une coquille pour la construction de minis systèmes experts, les techniques sont particulières à ce genre de programmation.

.variables

-Les variables déclarées sont globales à l'espace de travail
mode de déclaration et d'assignation de valeur:
sélectionner DECLARE
sélectionner DECLARE A VARIABLE
taper le nom de la variable, suivi de]
sélectionner SET
taper la valeur de la variable, suivi de]
Quand on a assigné une valeur à une variable, celle-ci n'est plus considérée comme variable, mais comme constante. Si on veut changer sa valeur, il faut recommencer tout le processus.
-Les paramètres et les variables dans une expression sont déclarés de la même façon.
-Il est possible de créer une variable dépendante (fonction de une ou plusieurs variables), représentée par son équation algébrique en utilisant le procédé décrit ci-dessus, ou en définissant une fonction.
-La variable ANSWER contient le dernier résultat calculé par le système.

.extensibilité

-Possibilité de définir des fonctions
-Possibilité de définir une opération infixe symbolisée par #
-Possibilité d'écrire de nouvelles règles permettant de créer de nouveaux environnements

.style de programmation

-Peu de recours à la véritable programmation, mais l'idée de programmation est présente.
-Aussi, programmation logique dans l'environement Prolog.

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers

Un seul type de fichier, qui permet d'enregistrer des environnements. Chaque environnement occupe 128 KO.

15. Gestion de l'espace de travail

La construction d'un environnement de travail approprié à une situation particulière se fait par la sélection successive d'un module, d'un état et d'un ou de plusieurs environnements, proposés par le système ou créés par l'usager.
-états: CAL: arithmétique décimale et rationnelle sur les nombres et les expressions algébriques
SIMPLIFIER: système expert permettant de simplifier les expressions algébriques; le système procède étape par étape et explique les transformations successives en termes de règles
SOLVER: système expert permettant la résolution d'équations
PLOTTER: le 'tableau noir' de CAL. Permet l'étude et la représentation graphique des fonctions. Contient le sous-ensemble Logo. Voir graphisme.
LOGIC: contient un sous-ensemble de PROLOG (implanté en LISP). Permet de construire une base de données, et en utilisant des règles d'inférence, d'étudier entre autres choses, si des propositions sont vraies ou fausses ou de déduire des propositions.
FILER: permet la modification d'un environnement ou la création d'un nouvel environnement.

-environnements: contiennent des commandes permettant de travailler avec des objets particuliers et de faire des opérations particulières. Ce sont REDU, TRIG, EXPN, COMP, LIMS, SYST, TAYL, NEWT, LISS, SYNT, GEOM, CALC, TRAJ, CIRC
Certains environnement sont utilisables dans un état seulement, d'autres dans plusieurs.
-modules: deux modules avec des propriétés différentes quant à la manipulation des environnements.
Une fois dans l'environnement approprié, les opérations sont aussi sélectionnées par voie de menu. Particularités: 1. Si on choisit une opération, par exemple 'évaluation décimale', on peut évaluer une expression, puis on est automatiquement ramené au niveau supérieur, et pour évaluer une deuxième expression, il faut passer à nouveau par le menu. Cette structure ralentit les interactions considérablement.
2. Pas de sortie sur imprimante autrement que par les touches Shift Prtsc.

16. Editeur

-Editeur de ligne
-Option permettant d'imprimer les expressions rationnelles sur deux lignes

17. Graphisme

-Sélection d'une fenêtre rectangulaire sur un plan cartésien; origine et graduation du système d'axes variables; le sens de rotation positif est le sens usuel en mathématiques
-Le mode graphique permet de tracer les graphes de fonctions, courbes, famille de droites, enveloppes de courbes, d'explorer la géométrie des transformations, d'étudier les effets des variations paramétriques, d'illustrer les processus d'approximations par (Newton, Taylor, Riemann)
-Géométrie de tortue: 7 commandes

18. Aide à l'usager

.correction des erreurs

-Ligne de messages intégrée au menu
-Quelques messages signalant les erreurs

.documentation

Deux manuels, à l'intention du professeur et de l'étudiant, reliés en spirale, anglais seulement:
-CAL: The Mathematics Discovery Language: initiation à l'utilisation de CAL sous forme de didacticiel interactif: discussion expliquant les caractéristiques de CAL; suggestions de projets. Pas d'aperçu global de la structure générale de CAL. Pas de liste complète du vocabulaire, ni des messages. Index très incomplet.
-An Introduction for Students: reproduction d'une partie du premier manuel.

.utilitaires

Deux environnements supplémentaires présentés sous formes d'appendices dans le volume CAL: fonctions transcendantes et géométrie sphérique.

MathCAD

MathCAD est un logiciel de conception très récente, permettant le calcul, la représentation graphique et l'édition de documents. Il intègre donc certaines des qualités que l'on retrouvait jusqu'à présent soit dans les logiciels du type MuMATH, TK Solver ou Eureka, soit dans les traitements de texte à vocation mathématique. On qualifie parfois ce type de logiciels de 'bloc-note électronique', puisqu'il permet à la fois la résolution de problèmes et l'élaboration de documents.

MathCAD ne peut être classé parmi les langages de programmation, puisqu'il ne permet pas la définition de procédures par l'usager. S'il a quand même paru intéressant de l'inclure dans cette étude, c'est parce qu'il est l'outil le plus récent dans cette catégorie, et qu'il exploite au maximum les capacités techniques de l'ordinateur personnel. Pour des raisons évidentes, la grille permettant d'identifier les caractéristiques des langages de programmation n'a pas été utilisée intégralement.

L'objectif de l'évaluation était d'identifier les caractéristiques de ce logiciel, afin de déterminer l'intérêt qu'il pourrait y avoir à le mettre à la disposition des étudiants pour la construction et la rédaction de solutions à des problèmes complexes, et bien sûr, à la disposition des professeurs pour la préparation de divers documents didactiques, notes de cours ou examens.

La version étudiée présente quelques améliorations par rapport à la version 1.1. Ce logiciel est programmé en langage C de Microsoft et en assembler.

FICHE TECHNIQUE

1. Nom	MathCAD, version 2.0
2. Distribué par	MathSoft Inc., One Kendall Square, Cambridge, MA 02139
3. Format	2 disquettes 5 1/4", 360 Ko. ou 1 disquette 3 1/2", 1.2 Mo
4. Equipement requis	IBM PC, XT, AT, PS/2, ou compatible, lecteur simple ou double, 512 Ko RAM, adaptateur monochrome CGA, EGA ou Hercules PC-DOS ou MS-DOS, version 2.0 ou plus
recommandé	Co-processeur 8087 ou 80x87
5. Documentation	MathCAD Reference Manual, MathSoft Inc., 290 p. MathCAD Quick Reference Manual, MathSoft, Inc., 12 p.
6. Prix	349.00 $US 50.00 $US pour la mise à jour

CARACTERISTIQUES

7. Représentation des données

Trois types d'objets: les nombres, les expressions algébriques et les tableaux

.nombres

-Nombres réels
notation: entiers ou à point décimal flottant, double précision
étendue: $10^{-307} < n < 10^{307}$
-Nombres complexes
notation: a+bi ou a+bj
Les nombres réels sont considérés comme un sous-ensemble des complexes et sont affichés automatiquement sous la forme a+bi quand le rapport b/a tend vers une certaine limite de tolérance établie par l'usager.
-Constantes: e=2.71828, pi=3.14159, oo=10^{307}, Z=.01, i ou j

.expressions algébriques

Expressions, fonctions, équations, systèmes d'équations et d'inéquations, contenant des nombres et des variables ayant reçu une valeur au préalable.

8. Structures de données

Une structure de données: les tableaux, dont les éléments sont des expressions numériques ou algébriques; ils servent à représenter les vecteurs, les matrices et les données statistiques.
Dimensions: 64 Ko, environ 8000 éléments

9. Manipulation des données

La manipulation des expressions numériques et algébriques ainsi que des structures de données se fait au moyen d'opérations, de fonctions et de commandes.

.opérations arithmétiques

Addition, soustraction, multiplication, division, inverse additif, exponentiation, reste, signe

.autres fonctions	Valeur absolue, racine carrée, racine, plus grand entier < n, plus petit entier >n, 3 fonctions trigonométriques et leurs inverses, 3 fonctions hyperboliques et leurs inverses, e^x, ln x, log x, partie réelle, partie imaginaire, argument, factorielle
notation	Symboles (mode infixe), mots-clés (mode préfixe)
évaluation	Comme dans un texte ordinaire, i.e. de gauche à droite et de haut en bas
précision	Déterminée par l'usager -maximum: 15 décimales -par défaut: 3
.algèbre linéaire	-La plupart des opérations arithmétiques et des fonctions mathématiques à un argument peuvent être distribués sur les éléments d'un vecteur ou d'une matrice au moyen de la commande VEC. -Opérations usuelles d'algèbre linéaire -Vecteurs: longueur, dernier élément, maximum, minimum -Matrices: dimensions, somme des éléments d'une diagonale, matrice identité -Racines d'une équation, solution d'un système d'équations, solution optimale
.calcul infinitésimal	-Sommes et produits partiels, dérivée en un point, intégrale définie (simple et double) -Fonctions d'interpolation linéaire et cubique, transformations de Fourier et leurs inverses, fonctions de Bessel
.calculs statistiques	Corrélation, droite de régression, moyenne, écart-type, variance, fonction gamma, erreur, distribution normale cumulative, distribution de fréquence
10. Structures de contrôle	Branchement conditionnel: IF (condition, vrai, faux) UNTIL (condition, valeur)
11. Vocabulaire	-61 fonctions et opérations (décrites ci-dessus) -25 commandes de système, de calcul, d'édition et de texte
12. Syntaxe	-Les touches du clavier sont utilisées pour les symboles d'opérations. Les expressions qui apparaissent à l'écran ou à l'imprimante sont identiques aux expressions mathématiques usuelles. -Les autres fonctions sont représentées par leurs abréviations anglaises usuelles. -Les commandes correspondent à des mots ou des clés de fonctions.
13. Programmation	MathCAD ne permet pas la définition de fonctions ou de programmes par l'usager.
.variables	-Les variables utilisées dans une expression doivent avoir reçu une valeur numérique au préalable et faire partie de l'espace physique balayé par l'analyseur avant d'arriver à l'expression à évaluer (en haut et à gauche de l'expression). -Enoncé d'assignation: nom := expression -Seule exception: les variables globales, déclarées par nom = expression.

ENVIRONNEMENT

14. Gestion des fichiers

Types de fichiers:
-fichiers de données enregistrés en ASCII.
-fichiers de documents (espace de travail)

15. Gestion de l'espace de travail

MathCAD permet trois types d'interaction: le mode calcul, le mode édition de texte et le mode graphique. A cette fin, l'écran peut être divisé en régions rectangulaires de dimensions variables contenant des expressions, du texte ou des graphes.
-Mode calcul: automatique ou manuel, local ou global
-Mode texte: plusieurs commandes d'édition permettant la recherche, la destruction, l'insertion, la copie, le déplacement et la manipulation de parties de texte.
-Mode graphique: voir graphisme
Mémoire disponible: 324 Ko
La gestion de la mémoire est automatique. L'usager peut vérifier l'espace mémoire disponible; il a également accès au DOS.

16. Editeur

-Les commandes d'édition sont similaires à celles d'un traitement de texte.
-On peut aussi manipuler globalement des blocs de texte ou d'équations au moyen des commandes déplacer, découper, coller.
-Alphabet grec

17. Graphisme

L'usager détermine la taille de la région, les intervalles sur les axes et la graduation.
Possibilité de tracer des points, droites, courbes, fonctions en escalier, histogrammes, et de tracer plusieurs graphes sur le même système d'axes.

18. Aide à l'usager

.correction des erreurs

-Ligne de messages intégrée au menu
-Erreurs de syntaxe identifiées et pointées à l'écran

.documentation

MathCAD Reference Manual: relié en spirale, anglais seulement
-explications sur l'entrée des données et les facilités d'édition du système
-capacités de calcul, y compris la description des méthodes numériques utilisées
Quick Reference manual: pour les vérifications rapides des caractéristiques de MathCAD

.système d'aide

-Fichier d'aide "on-line"
-Fichier didacticiel permettant de se familiariser avec l'environnement
-Disquette de démonstration des divers aspects de MathCAD
-Ligne téléphonique sans frais pour support à l'usager
-Bulletin tri-annuel

MuMATH

MuMATH est un système interactif capable d'effectuer des opérations de calcul numérique et symbolique sur des expressions contenant des variables auxquelles on n'a pas assigné de valeur numérique. Ce langage, développé vers la fin des années 1970, constitue une version pour micro-ordinateur, de systèmes plus puissants tournant sur ordinateur central.

La première version de muMATH apparut en 1979. Elle était le résultat des efforts de A. Rich et D. Stoutmayer visant à améliorer la rapidité et la puissance de calcul numérique d'un interpréteur LISP. La version 83 présente plusieurs améliorations par rapport à la première, particulièrement en ce qui a trait à la puissance des opérations reliées au calcul infinitésimal.

La structure de muMATH est organisée en une hiérarchie de fichiers dont chacun offre des possibilités de calcul se rattachant à un domaine particulier. Pour constituer un espace de travail, il suffit de ramener en mémoire les fichiers nécessaires à un ensemble d'opérations, ainsi que les fichiers préalables dans la hiérarchie. Cet espace de travail peut alors être enregistré globalement dans un fichier pour utilisation ultérieure.

MuSIMP est le langage de programmation dans lequel muMATH a été écrit. Il est accessible à l'utilisateur de muMATH qui veut étendre les capacités de cet outil en définissant de nouvelles fonctions qui permettront d'obtenir des résultats non disponibles dans ce logiciel. Selon les termes de ses concepteurs, muSIMP offre à l'utilisateur la puissance d'un langage applicatif comme LISP en même temps qu'une syntaxe plus naturelle.

FICHE TECHNIQUE

1. Nom — Microsoft muMATH Symbolic Mathematics Package
MuSIMP-83 version 4.06

2. Distribué par — The Soft Warehouse, P.O. Box 11174, Honolulu, Hawaii 96828, U.S.A.
au Canada: Microsoft Canada Inc., 6300 Northwest Drive, Mississauga, Ont., L4V 1J7

3. Format — 4 disquettes 5 1/4", 360 Ko.

4. Equipement requis — IBM PC ou compatible, moniteur monochrome ou à affichage graphique, 128 Ko RAM, lecteur de disquettes simple ou double, PC-DOS ou MS-DOS, version 2.0 ou plus

5. Documentation — MuMATH-83 Reference Manual, The Soft Warehouse, (1983), 280 p.

6. Prix — 325.00 $

CARACTERISTIQUES DU LANGAGE

7. Représentation des données

Quatre types d'objets: les nombres, les expressions algébriques, les tableaux et les arbres binaires.

.nombres
-Nombres entiers
-Nombre décimaux, à point fixe ou flottant
-Nombres rationnels, sous forme de quotient d'entiers
-Nombres irrationnels, représentés sous la forme exponentielle non évaluée étendue des nombres réels limitée par la capacité du système.
-Nombres complexes, notés sous la forme a+b#I
-Constantes: e (#E), pi (#PI) et i (#I)

.expressions algébriques — Expressions, fonctions et relations, équations et systèmes d'équations

8. Structures de données
-MuMATH: les tableaux emboîtés (non nécessairement rectangulaires), dont les éléments sont des données numériques ou algébriques; ils servent à représenter les vecteurs et les matrices.
Les tableaux peuvent contenir des éléments de types différents.
-MuSIMP: les arbres binaires, dont les éléments (noeuds) sont des objets muSIMP.

9. Manipulation des données — La manipulation des données se fait au moyen d'opérations et de fonctions. La plupart s'appliquent indifféremment sur les données numériques, algébriques et les tableaux, mais quelques-unes s'appliquent sur un type d'expression particulier.

.opérations arithmétiques — Addition, soustraction, multiplication, division, inverse additif, exponentiation, quotient entier, reste, PPCM, PGCD

.autres fonctions — Valeur absolue, numérateur, dénominateur, minimum, les 24 fonctions trigonométriques, trig. inverses, hyperboliques et hyp. inverses, e^x, ln x, $\log_{10}$ x, $\log_e$ x, factorielle

notation
- Infixe pour les opérateurs arithmétiques (+, -, *, /, ^)
  Le symbole de multiplication * est facultatif, sauf devant une expression entre parenthèses.
- Préfixe pour les autres fonctions utilisant des mots; l'argument ou les arguments doivent être entre parenthèses, séparés par des virgules.
- L'arithmétique rationnelle est invoquée en mettant à FALSE la variable de contrôle POINT.

priorité
1. Symbole de citation (')
2. Factorielle (!)
3. Exponentiation (^)
4. Moins unaire (-)
5. Multiplication et division (*, /)
6. Addition et soustraction: (+, -)

évaluation — L'évaluation se fait de gauche à droite (sauf l'exponentiation), en suivant l'ordre de priorité ci-dessus.

précision — Fixée par l'usager; limite déterminée par les contraintes du système.
Par défaut: arithmétique rationnelle

.opérations algébriques — Développement, dénominateur commun, factorisation, simplification, développement en fractions partielles, rationalisation, conjugué, PGCD, quotient, reste, substitution, solution des équations, sélection de parties des équations

.algèbre linéaire
- La plupart des opérations arithmétiques et des fonctions mathématiques à un argument se distribuent sur les éléments d'un vecteur ou d'une matrice.
- Produit scalaire, vectoriel, mixte, produit par un scalaire, matrice identité, transposée, produit matriciel, puissance, division, inverse, déterminant, résolution de systèmes linéaires

.calcul infinitésimal
- Limite, dérivée partielle, dérivée d'ordre n, séries de Taylor, intégrale indéfinie et définie, sommes partielles, produits partiels,
- Solution des équations ordinaires d'ordre 1 et d'ordre supérieur.

.calcul vectoriel — Quelques opérations du calcul vectoriel

.autres manipulations
- Logique: 3 opérations
- Comparaison et appartenance: 6 fonctions et opérations
- Prédicats: 12 fonctions
- Sélection: 17 fonctions
- Construction: 5 fonctions
- Modification: 4 fonctions
- Manipulation des chaînes: 7 fonctions

10. Structures de contrôle -Branchement conditionnel (WHEN...EXIT)
-Branchement inconditionnel: itération (LOOP...ENDLOOP)
(BLOCK...ENDBLOCK)
appel de sous-routine (SUBROUTINE...ENDSUB)
récursion

11. Vocabulaire -MuMATH: 88 opérations et fonctions
19 variables de contrôle
-MuSIMP: 138 opérations, fonctions et variables de contrôle dont certaines sont les mêmes que pour MuMATH.

12. Syntaxe Symboles particuliers:
-le symbole ' doit précéder les chaînes de caractères.
-les arguments des opérations préfixes doivent être entre parenthèses, séparées par des virgules.
-le symbole de multiplication * est facultatif sauf devant une expression entre parenthèses.
-le symbole = teste l'égalité de deux expressions.
-le symbole == sépare les deux membres d'une équation.
-le symbole # précède les nombres e, pi et i.
-le symbole ; doit terminer les expressions.
-le symbole % permet d'insérer des commentaires.
-le symbole $ supprime l'affichage des résultats.
-espaces facultatifs entre les nombres et les opérations (sauf moins unaire)
-les composantes d'un vecteur ligne doivent être entre crochets carrés [] et séparés par des virgules.
-les composantes d'un vecteur colonne doivent être entre accolades {} et séparés par des virgules.
-les matrices sont notées comme un ensemble de vecteurs lignes ou colonnes.

Un seul type d'énoncé: l'expression terminée par le symbole ;

13. Structure des programmes -MuMATH: utilisé en mode direct
-MuSIMP: programmation procédurale structurée de style similaire à Pascal.

.variables -Les variables créées par assignation de valeur sont globales à l'espace de travail.
énoncé d'assignation: nom: valeur;
ou: ASSIGN (nom, valeur);
-On peut créer une variable dépendante (fonction de une ou plusieurs variables), représentée par son équation algébrique en utilisant l'énoncé d'assignation ou en définissant une fonction MuSIMP.
-Le système comprend aussi des variables de contrôle globale auxquelles des valeurs par défaut sont assignées; elles peuvent être changées pour d'autres valeurs choisies dans un domaine spécifié pour chaque variable.
-La variable @ contient le dernier résultat calculé par le système.

.transparence -L'allocation de mémoire est automatique.

ENVIRONNEMENT DE PROGRAMMATION

14. Gestion des fichiers

Types de fichiers:
-MuMATH-83 est contenu dans des fichiers sources écrits en MuSIMP. Chacun des 22 fichiers offre des capacités de calcul particulières. L'organisation des fichiers est hiérarchique, la présence de certains fichiers dans l'espace de travail étant préalable à d'autres.
-Les fichiers systèmes contiennent les programmes, les variables et les fichiers sources présents dans l'espace de travail au moment de l'enregistrement du fichier. Le système offre 3 fichiers de ce type, reliés à l'algèbre, au calcul et à l'algèbre linéaire. Ces fichiers ne peuvent être imprimés.
-Il est possible de créer des espaces de travail sous forme de fichiers systèmes.

15. Gestion de l'espace de travail

-320 Ko de RAM utilisés si disponibles
-L'allocation et le recyclage de la mémoire sont automatiques.

16. Editeur

-Accès à l'éditeur de ligne du système d'opération.
-Editeur pleine page pour la programmation en MuSIMP; option permettant d'imprimer les expressions rationnelles sur deux lignes.

17. Graphisme

La version pour IBM n'offre pas d'option graphique.

18. Aide à l'usager

.correction des erreurs

8 messages signalent différents types d'erreurs.

.pauses

-Possibilités d'insérer des pauses aidant la correction des programmes.
-Système élaboré d'aide au dépistage et à la correction des erreurs.

.documentation

Un manuel d'utilisation présenté en cartable, anglais seulement. Deux parties:
-MuMATH: guide d'installation, initiation à l'utilisation de muMATH et muSIMP; organisation des fichiers; contenu des fichiers sources et exemples d'utilisation.
-MuSIMP: structures de données, gestion de la mémoire, messages; vocabulaire muSIMP; programmation en muSIMP; langage machine; ouvrages de référence; glossaire et listes des vocabulaires muMATH et muSIMP.

.utilitaires

-Editeur
-Dépistage et correction d'erreurs

.système d'aide

-Fichiers offrant 4 leçons sur muMATH et 8 sur muSIMP
-Soutien téléphonique

CHAPITRE IV

COMPARAISON ET EVALUATION

"L'univers des solutions possibles
peut être simple ou complexe.
La ligne de démarcation est celle qui sépare
ce que nous savons de ce que nous ignorons."
D. N. Chorafas

Ce chapitre présente les résultats de la comparaison et de l'évaluation des langages faites à partir des caractéristiques identifiées au chapitre précédent. Ces résultats sont regroupés selon un ordre quelque peu différent de celui de la grille: ils font ainsi mieux ressortir, du moins faut-il l'espérer, les liens entre mathématiques et programmation.

4.1 Comparaison et évaluation des langages

4.1.1 Les langages de programmation comme systèmes de notation

Représentation des objets

Nombres

Seul MuMATH permet de représenter tous les ensembles de nombres inclus dans les réels, y compris les rationnels sous forme de quotient d'entiers ou sous forme décimale au choix, et les irrationnels sous forme exponentielle non évaluée. Son intervalle de représentation est le plus large, et il permet également la représentation des nombres complexes.

MathCAD et CAL représentent les irrationnels par leur approximation décimale. MathCAD affiche les rationnels sous forme de quotient d'entiers, mais utilise leur approximation décimale dans les calculs; CAL représente et traite les rationnels sous leur forme exacte, mais son intervalle de représentation est restreint. Logo ne fait pas de distinction entre les divers types de nombres réels, et son intervalle de représentation est aussi très large.

Q'Nial permet la représentation des nombres complexes mais ne reconnaît pas l'égalité des formes entière et décimale d'un même nombre. APL et BASIC ne permettent pas la représentation des complexes, mais font la conversion automatique de nombre entier à décimal. Enfin, BASIC offre l'étendue la plus restreinte de tous.

L'affichage est unidimensionnel dans tous les langages, sauf dans MathCAD et dans CAL (optionnel).

Expressions symboliques

MuMATH, MathCAD et CAL permettent la représentation d'expressions algébriques, de fonctions et de relations, d'équations, de systèmes d'équations et d'inéquations dans le cas de MathCAD. Cependant, seul MuMATH n'oblige pas la déclaration préalable des variables. Avec CAL, elles doivent être déclarées avant d'être utilisées dans une expression, et avec MathCAD, elles doivent avoir reçu une valeur numérique.

APL, Logo et Q'Nial ne permettent que la représentation de fonctions et de relations d'une ou de plusieurs variables, sous forme de variable globale ou de procédure. En Q'Nial, les variables doivent avoir reçu une valeur, ou avoir été déclarées externes au préalable.

BASIC n'offre que l'énoncé DEF FN, utilisable en mode de programmation seulement.

Structures de données

Avec sa structure de tableaux rectangulaires emboîtés à accès direct, et ses listes à accès direct ou séquentiel, Q'Nial offre le plus de versatilité dans la représentation des données groupées, car il permet d'émuler un grand nombre d'autres structures dont les arbres, les réseaux et les fiches. De plus, les diagrammes utilisés en option pour représenter les structures de données à l'écran sont intéressants du point de vue visuel et offrent un bon support à la représentation mentale des objets appartenant à l'espace à n dimensions. MuMATH-MuSIMP offre aussi des structures puissantes pour la représentation des données groupées.

APL supporte la structure de tableaux emboîtés, BASIC celle de tableaux dont les dimensions doivent être déclarées, et Logo, la structure de liste. Ces trois logiciels sont classés au même rang car, pour représenter les vecteurs et les matrices, la structure de tableau est plus fonctionnelle. La structure de liste est par contre plus versatile en ce qu'elle permet d'en émuler plusieurs autres et aussi en ce qu'elle s'accommode mieux d'un espace mémoire fragmenté.

MathCAD ne permet que la représentation des nombres en tableaux et CAL ne supporte aucune structure de données.

Représentation des opérations

Tous les langages, sauf APL, utilisent les mêmes conventions (symboles et notation infixe) pour la représentation des opérations arithmétiques. APL, Logo et Q'Nial offrent l'alternative des primitives préfixes, ce qui diminue le nombre de parenthèses nécessaires dans les expressions complexes. Les autres fonctions mathématiques usuelles sont préfixes et utilisent à peu près les mêmes mots ou abréviations.

MuMATH, MathCAD et CAL contiennent d'autres opérateurs tels limite, dérivée, intégrale, sommation. Les caractères générés à l'écran par MathCAD sont identiques à ceux qu'utilise la notation mathématique traditionnelle, mais l'association avec les touches du clavier n'est pas évidente.

Graphisme

CAL et MathCAD permettent la représentation graphique de fonctions et de relations sur des systèmes d'axes cartésiens dont la taille et la graduation sont déterminées par l'usager. En fait le mode graphique est probablement l'environnement le plus intéressant de CAL du point de vue apprentissage, car il permet l'étude des transformations, la représentation des solutions aux équations différentielles, la solution numérique et graphique de divers types d'équations, l'étude de l'influence des variations des paramètres et le tracé des fonctions approximant l'intégrale. MathCAD permet également de tracer l'histogramme correspondant à un ensemble de données statistiques.

CAL et Q'Nial contiennent des sous-ensembles dotés de certaines des caractéristiques de la géométrie de la tortue sous forme d'un environnement dans le cas de CAL, et d'un fichier défini par l'usager dans le cas de Q'Nial. Ni l'un ni l'autre n'est très performant, et c'est en Logo que ce concept est le mieux présenté. Le mode graphique de Logo permet à la fois la représentation en géométrie locale et cartésienne. On regrette toutefois que le sens de rotation positif ne soit pas le même qu'en mathématiques, et que les angles soient mesurés en degrés. Enfin, cette version de Logo ne permet que le graphisme en résolution moyenne.

APL et BASIC offrent des possibilités de représentation graphique semblables, sur un système d'axes dont l'origine coïncide avec le coin supérieur gauche de l'écran.

Le Tableau 2 résume les résultats présentés ci-dessus relativement à l'évaluation des langages de programmation comme systèmes de notation. Si on ajoutait une pondération numérique aux cotes d'évaluation, on constaterait que

MathCAD se classe premier comme système de notation. En effet, c'est bien ce logiciel qui offre le meilleur soutien à l'apprentissage et à la maîtrise du système de notation utilisé en mathématiques. Cependant, les choses ne sont pas aussi simples, car MathCAD est suivi de près par MuMATH, Q'Nial et Logo, lesquels sont dotés de structures plus puissantes que celles de MathCAD pour la représentation des données groupées, tout en ne faisant pas strictement usage de la notation utilisée en mathématiques et en n'offrant pas d'opérations de calcul différentiel et intégral. Un détail agaçant dans MuMATH: l'usager doit se préoccuper constamment des majuscules et des minuscules. C'est pourquoi il est impossible de considérer l'un quelconque de ces langages supérieur à tous les autres en tant que système de notation.

Tableau 2: Langages et notation

	APL	BASIC	Logo	Q'Nial	CAL	MathCAD	MuMATH
Nombres	B	C	B	C	B	B	A
Expressions symboliques	C	D	C	C	B	B	A
Structures de données	C	C	C	A	-	D	B
Opérations	B	C	B	B	B	A	B
Graphisme	C	C	B	C	B	B	-

A: excellent, B: très bon, C: bon, D: pauvre, E: très pauvre; -: non disponible

4.1.2 Les langages de programmation et le développement des connaissances procédurales

Calcul numérique

MuMATH, MathCAD, APL et Q'Nial offrent le plus grand nombre d'opérations et de fonctions mathématiques, soit environ le double de ce qu'offrent les autres langages, et ces opérations s'appliquent également sur les données groupées en structures lorsque la chose a du sens. Avec MuMATH, la priorité des opérations est bien définie, la précision quasi-illimitée et les calculs sont exacts. MathCAD, APL et Q'Nial utilisent les approximations décimales avec un maximum de 16 chiffres significatifs. Problème important en Q'Nial: à cause de la dichotomie entre nombres entiers et décimaux, les calculs avortent souvent à cause du dépassement de la capacité de calcul sur les entiers, et le système bloque quand le résultat dépasse l'étendue des réels.

Logo, CAL et BASIC offrent moins d'opérations pré-définies. Parmi les trois, Logo offre le meilleur environnement de calcul numérique, du point de vue précision et exactitude et surtout parce que l'usager n'a pas à se soucier du type de nombre réel qu'il manipule. En BASIC, les calculs sont effectués très rapidement. CAL n'est pas à la hauteur de ses promesses en ce qui a trait au calcul numérique: le système bloque bien avant la limite des 255 chiffres significatifs dont parle la documentation.

Calcul symbolique

Seul MuMATH permet les manipulations de type algébrique d'expressions contenant des variables non déclarées. CAL permet d'appliquer quelques règles menant à la simplification d'expressions et à la résolution des équations -mais on doit d'abord déclarer les variables- ainsi que des opérations de calcul telles la dérivée en un point ou l'intégration numérique par les sommes de Riemann. MathCAD ne traite que les expressions contenant des variables ayant reçu une valeur numérique.

Dans les autres langages, il est possible de définir certaines opérations effectuant de telles transformations (v.g. les opérations élémentaires sur les fonctions), mais la chose est ardue et n'est pas nécessairement à la portée des novices en programmation; de plus, elle est relativement sans intérêt pour l'apprentissage des mathématiques, car elle requiert plus d'habileté en programmation qu'autre chose. En BASIC, cette possibilité n'existe pas.

Maîtrise des algorithmes

Les diverses possibilités offertes par les langages déclaratifs augmentent la puissance de l'étudiant dans le domaine du calcul symbolique. Elles sont donc de nature à alléger sa tâche dans des situations où le traitement d'expressions symboliques n'est qu'une des étapes d'un problème complexe. Elles ne peuvent guère l'aider par contre à maîtriser les algorithmes sur lesquels ces manipulations sont fondées car, en général, ces langages n'expliquent pas comment ces résultats sont obtenus, et la documentation qui les accompagne n'est guère plus explicite.

Ainsi, par exemple, quand un étudiant demande à l'ordinateur les racines d'une équation, il obtient bien sûr une réponse qu'il pourra utiliser à différentes fins, mais il n'apprend pas à choisir et à appliquer les différents algorithmes qui permettent d'obtenir ce résultat, pas plus qu'il ne peut comparer leur efficacité, leur vitesse de convergence ou d'autres propriétés. La construction d'algorithmes et leur implantation sur ordinateur force l'étudiant à scruter la nature de ces processus, à établir une correspondance entre les opérations et les objets, bref

à comprendre leur mécanisme.

C'est pourquoi les langages procéduraux, c'est-à-dire ceux qui prescrivent à l'ordinateur la marche à suivre pour obtenir un résultat donné, et surtout les langages qui supportent la programmation fonctionnelle, semblent plus appropriés à l'atteinte de cet objectif.

Un autre domaine relativement inexploré, en rapport avec les mathématiques et l'ordinateur, est celui de la construction de preuves. Le gros des mathématiques collégiales consiste à apprendre des procédures permettant de trouver des réponses numériques ou symboliques à des problèmes. Cependant, il existe des résultats importants, souvent présentés sous forme de théorèmes, qui ne font que prescrire les conditions selon lesquelles des solutions peuvent être obtenues, sans pour autant expliquer comment on peut les trouver. C'est le cas notamment du théorème fondamental de l'algèbre, du théorème de la valeur moyenne, ou des théorèmes sur les limites. Ces théorèmes d'existence, de même que leurs preuves, ont une généralité et une élégance qui s'accommodent bien du langage mathématique formel, mais il est également possible de les manipuler au moyen de langages informatiques. Les langages procéduraux permettent de construire des algorithmes finis et non ambigus, qui prennent fin seulement lorsqu'ils ont produit le résultat désiré, ou lorsqu'ils peuvent déclarer qu'un tel résultat n'existe pas. Si l'on peut prouver qu'un tel algorithme existe, alors on a réussi à démontrer simultanément l'existence de cet objet et la façon de le construire. Les langages supportant la programmation logique permettent, pour leur part, de s'attaquer à la construction de preuves et à la recherche d'objets satisfaisant les conditions de ces théorèmes. Selon Maurer (1983), les preuves algorithmiques devraient constituer un thème fondamental des mathématiques de niveau collégial, maintenant que l'ordinateur fait partie de l'environnement quotidien et qu'il est devenu sans intérêt de consacrer à l'apprentissage des routines du calcul numérique et symbolique tout le temps qu'on y consacrait traditionnellement.

APL, Logo, Q'Nial et, dans une moindre mesure, MuSIMP, permettent cette démarche d'apprentissage centrée sur les algorithmes, en tant que langages procéduraux supportant la définition de fonctions. Q'Nial supporte en plus la programmation logique. BASIC ne supporte que la programmation procédurale; CAL favorise, dans une certaine mesure, la familiarisation avec le cheminement de preuve, et il ne saurait être question de construction d'algorithmes avec MathCAD.

Construction de modèles

La résolution de problèmes implique nécessairement la construction de représentations ou de modèles. Dans cette perspective, la modularité et l'extensibilité sont deux qualités souhaitables dans un langage. La modularité permet l'approche par la décomposition d'un problème en sous-problèmes, et l'extensibilité permet d'ajouter de nouvelles fonctions à l'environnement, d'émuler des structures de données additionnelles et même de générer de nouvelles structures de contrôle. De plus, si l'interpréteur fait partie de l'environnement d'exécution des programmes, il est possible de construire des programmes à l'intérieur desquels de nouvelles instructions seront créées en cours d'exécution, selon que certaines conditions sont satisfaites ou non. De façon générale, APL, Logo et Q'Nial sont des langages extensibles et modulaires qui satisfont également cette dernière condition.

Enfin, l'orientation objet et les possibilités de graphisme sont aussi des caractéristiques fort utiles dans un environnement de résolution de problèmes.

Expérimentation, exécution, exploration

L'exploration est une activité orientée vers la découverte, vers la possession de ce qui était jusque-là inconnu. Elle implique donc la résolution de problèmes ouverts, c'est-à-dire de problèmes peu structurés, souvent définis de façon incomplète, et pour lesquels il n'existe pas d'algorithme connu.

Si l'ordinateur doit être utilisé dans ce contexte, il importe que le langage utilisé permette deux approches de programmation: l'approche descendante selon laquelle la solution est planifiée en termes de procédures et de sous-procédures, où aucune procédure n'est écrite avant que le programme n'ait été entièrement spécifié; et l'approche ascendante selon laquelle le programmeur peut commencer à écrire des procédures exécutant certaines tâches, même s'il n'a qu'une vague idée de ce que sera le programme final.

Il importe également que le langage utilisé soit hautement interactif, afin que l'ordinateur et l'usager puissent collaborer activement dans l'acte d'exploration.

Le Tableau 3 résume les résultats exposés ci-dessus. Il montre que APL, Logo et Q'Nial sont des outils plus propices au développement du savoir procédural, ou du savoir-faire mathématique. Cependant ils ne sont pas équivalents à tous points de vue, car ils offrent des possibilités et des environnements très différents. MuMATH-MuSIMP vient légèrement derrière, car les idées de modularité et

d'extensibilité y sont moins présentes et le graphisme inexistant. Quant aux autres langages, ils offrent moins de puissance à cet égard.

Tableau 3: Langages et procédures

	APL	BASIC	Logo	Q'Nial	CAL	MathCAD	MuMATH
Calcul numérique	B	C	C	D	D	B	A
Calcul symbolique	D	E	D	D	B	C	A
Maîtrise des algorithmes	A	B	A	A	D	E	C
Construction de modèles	A	C	A	A	D	D	D
Exploration	B	D	A	B	D	E	D

A: excellent; B:très bon; C: bon; D: pauvre; E: très pauvre; -: non disponible

4.1.3 Langages de programmation et construction des concepts

Il va de soi que tous les langages de programmation peuvent favoriser le développement de structures de représentation des concepts mathématiques, puisqu'il est possible de construire des activités reliées à l'apprentissage de ces concepts qui utilisent chacun de ces langages, et qu'à peu près n'importe quelle expérience peut ajouter à la somme des connaissances, si l'activité est planifiée de façon adéquate.

Ainsi, par exemple, il est possible d'explorer l'intégration numérique en faisant construire par les étudiants des programmes en Logo ou en BASIC dans lesquels sont implantés divers algorithmes, même si les opérations du calcul infinitésimal ne sont pas disponibles dans ces langages. De la même façon, on peut définir en APL les opérations nécessaires au calcul statistique ou ensembliste.

Cependant, il va aussi de soi que la chose se fait plus facilement avec certains langages qu'avec d'autres, compte tenu des structures de données, des structures de contrôle et des fonctions intégrées dans le langage. C'est pourquoi il a été jugé pertinent d'indiquer ici dans quel domaine chacun des langages semble offrir le plus de possibilités.

APL

APL est surtout identifié à l'algèbre linéaire. Dans ce domaine, il offre toutes les opérations nécessaires, sauf le déterminant. Mais comme la plupart des langages, APL offre aussi un bon environnement de calcul numérique. Le style de

programmation qu'il supporte -en APL, toute procédure est une fonction- permet de représenter les fonctions et les variables d'une façon très naturelle et d'explorer divers domaines: nombres premiers, limite, etc. Il permet enfin assez facilement de définir les opérations nécessaires au calcul statistique et ensembliste et à la logique. On notera cependant qu'APL n'offre que peu de variété dans les structures de contrôle, et peu de versatilité dans les styles de programmation.

La facilité d'accès à ce langage est matière d'opinion. La mémorisation des symboles étranges qui lui servent de vocabulaire (le vocabulaire de mots-clés ne semble pas très utilisé) et la familiarisation avec sa syntaxe et ses règles d'évaluation semblent, à première vue, susceptibles de poser des difficultés, qui sont possiblement compensées d'autre part par la compacité du langage. Ce langage est évidemment aussi celui qui nécessite le moins la maîtrise de l'anglais, bien que les commandes et l'environnement de programmation fasse évidemment usage de cette langue.

Si l'on en juge par les publications à ce sujet, APL est un des langages choisis dans les collèges et universités où les activités sur ordinateur sont intégrées aux cours de mathématiques. Comme il est utilisé très souvent en mode direct, son utilisation ne requiert pas beaucoup de connaissances en programmation.

BASIC

BASIC n'offre en avantages que sa simplicité et son environnement de calcul numérique. En fait, ce langage supporte assez mal la comparaison avec les autres langages évalués ici, et ce, pour des raisons faciles à comprendre.

BASIC, comme Pascal, est issu de FORTRAN, l'un des premiers langages de haut niveau développés dans les années 50. Ces langages reflètent les caractéristiques et les déficiences des ordinateurs de cette époque où le programmeur devait se soucier d'une foule de détails informatiques souvent reliés à l'allocation de la mémoire et à la rapidité d'exécution. Même si leurs carences les plus flagrantes ont été corrigées dans des versions récentes, il n'en demeure pas moins qu'ils sont dépourvus des aspects de modularité, d'extensibilité et de transparence dont sont dotés les langages plus modernes.

De plus, selon Harvey (1985), ces langages ont été conçus avant que les bases mathématiques de la programmation n'aient été véritablement comprises. Ils sont donc moins compatibles avec le langage mathématique que ne le sont les langages modernes basés sur un modèle mathématique (la manipulation des matrices dans le cas d'APL, la composition

des fonctions dans le cas de Logo, et le calcul des prédicats dans celui de Prolog).

Considérant la richesse des apprentissages que permettent les langages plus récents, il ne saurait être justifiable de limiter aux possibilités de BASIC un environnement d'apprentissage des mathématiques, et il est à souhaiter que le cours d'informatique dispensé au secondaire serve à initier les étudiants qui le suivent à des outils plus performants, qui leur conféreront plus de puissance formelle, et qu'ils pourront réutiliser avec profit par la suite.

Logo

Logo offre un bon environnement pour le calcul et les approximations numériques, l'exploration des nombres, des fonctions et des variables, par l'intermédiaire de la manipulation des procédures et de la représentation graphique.

Les possibilités graphiques offertes par Logo constituent peut-être l'aspect le plus attrayant et certainement le mieux connu de ce langage. Utilisé pour l'introduction à la programmation, le graphisme permet la concrétisation d'idées très puissantes en programmation. Intégré à l'apprentissage des mathématiques, il permet la visualisation de phénomènes liés à l'idée de variation.

Logo est aussi le seul langage qui permette l'étude de notions géométriques. Son micro-monde de la tortue est assez connu pour qu'il soit superflu d'en parler ici, mais il existe d'autres micro-mondes destinés à supporter des apprentissages particuliers (v.g. Motions, développé par Thompson pour l'apprentissage des transformations isométriques du plan, Kearsley, 1987).

La facilité avec laquelle de tels micro-mondes peuvent être développés est une des caractéristiques les plus attrayantes de Logo dans le contexte scolaire. Elle ne doit toutefois pas nous faire oublier la facilité d'accès à ce langage dont l'environnement de programmation est extrêmement convivial. Ce langage est d'ailleurs le seul où il soit facile de traduire les primitives et le contenu des messages, ce qui permet de créer un environnement d'apprentissage respectant la langue vernaculaire des apprenants.

Q'Nial

Q'Nial est un langage présentant beaucoup de similitudes avec APL, mais qui a l'ambition d'être plus puissant, et qui l'est effectivement.

Q'Nial possède au départ les primitives nécessaires pour le travail sur les nombres, les fonctions et les variables, l'algèbre linéaire (sauf l'inversion des matrices), la théorie des ensembles, la logique, ainsi que des utilitaires permettant le graphisme et le calcul statistique. Il permet aussi la construction de bases de données où il sera possible d'étudier les relations existant entre les éléments, de définir des règles, d'établir des inférences, des démonstrations et des réfutations. Parmi les langages étudiés ici, Q'Nial est le seul langage à supporter une aussi grande variété dans les styles de programmation.

Mais toute cette puissance a un prix. Le langage ne possède ni éditeur de texte ni éditeur de ligne convenables; son mode graphique n'est pas accessible aussi facilement qu'en Logo, par exemple, et même s'il est défini avec rigueur, la construction des énoncés et la manipulation de la syntaxe présenteraient des difficultés non négligeables dans un contexte d'apprentissage autre que celui de l'informatique.

Pour toutes ces raisons, il ne serait pas réaliste de vouloir intégrer l'initiation à ce langage aux cours de mathématiques, particulièrement en milieu francophone.

Q'Nial est un langage qui possède, dit-on, un bon réseau d'utilisateurs dans le milieu académique. Il a fait l'objet de recherches et plusieurs projets d'études graduées y ont apporté des contributions intéressantes, tant sous la forme de micro-mondes que de publications. Ses promoteurs manifestent également une grande ouverture d'esprit face à l'utilisation de ce logiciel à des fins expérimentales.

CAL

CAL est un logiciel qui promet beaucoup plus qu'il ne donne. Ses possibilités de calcul numérique et symbolique sont réduites, et ses environnements reliés à l'algèbre (SIMPLIFIER et SOLVER), qui sont des états pour employer le vocabulaire CAL, permettent finalement peu de manipulations. Les opérations reliées au calcul infinitésimal se réduisent à peu de choses près à des approximations. L'utilisation des variables n'est pas conforme à celle qu'on en fait en mathématiques: dès qu'une variable a reçu une valeur, elle n'est plus considérée comme variable mais comme constante, et pour lui assigner une nouvelle valeur, il faut la déclarer à nouveau.

Mais la plus grande source de frustrations pour l'utilisateur de ce langage ne peut provenir que de l'environnement contraignant qu'il propose. La sélection des états, des environnements et des opérations se fait par

voie de menu hiérarchique. Or chaque opération, que le résultat soit celui que l'on désire ou non, que l'on ait fait une faute ou non, ramène le contrôle au niveau supérieur, de sorte que si l'usager veut évaluer cinq expressions, c'est cinq fois qu'il devra sélectionner EVALUATION, puis DECIMAL EVALUATION (sans compter le nombre d'erreurs). De plus, l'environnement ne permet pas la sortie sur imprimante autrement que par les touches Shift et PRTSC.

Enfin, la documentation est de piètre qualité. Elle consiste principalement en une série de projets (dont quelques-uns sont intéressants) et le déroulement intégral de l'interaction ordinateur-usager y occupe pratiquement tout l'espace. Quelques pages seulement donnent des renseignements fragmentaires sur le système. Nulle part trouve-t-on de liste complète des commandes et fonctions, des messages, etc... L'index lui-même n'est pas suffisamment détaillé.

CAL permet la familiarisation avec la notion de système expert, en ce sens qu'il permet la définition et l'utilisation de règles d'inférence. Il permet aussi une certaine forme d'exploration des lieux géométriques, mais ne possède à peu près aucune des qualités qui font d'un langage un outil polyvalent et souple, favorisant l'exploration et l'apprentissage plutôt que de le contraindre dans un cadre rigide.

MathCAD

MathCAD est un outil qui confère à l'usager une bonne puissance de calcul numérique dans tous les domaines identifiés, sauf en géométrie. Il permet également le graphisme et l'édition de texte. Son système de notation est conforme à celui des mathématiques.

Bien que ce logiciel n'ait pas été expérimenté avec une clientèle étudiante, il semble qu'il ne devrait pas exiger une période d'initiation très longue. Les choix se font par voie de commande ou de menu et plusieurs opérations et commandes correspondent à une touche ou clé de fonction. La documentation est bien organisée, de consultation facile, et contient des renseignements sur les algorithmes utilisés dans les méthodes numériques.

Il serait certainement avantageux, comme tel, de le mettre à la disposition des étudiants pour la solution de problèmes et la rédaction de devoirs, de même qu'à celle des professeurs pour la préparation de documents. Il présente des qualités que MuMATH ne possède pas (statistiques, graphisme), mais ne saurait remplacer ce dernier pour ce qui est du calcul symbolique.

MathCAD n'est cependant pas un langage en ce sens qu'il ne permet pas la définition de procédures par l'usager. Pour cette raison, sa puissance comme outil d'apprentissage est assez restreinte.

MuMATH

La supériorité de MuMATH sur les autres langages est incontestable pour ce qui est du calcul numérique et symbolique, des opérations du calcul infinitésimal et de l'algèbre linéaire.

MuSIMP permet également la définition de fonctions par l'usager, mais dans un environnement et avec un style de programmation aujourd'hui dépassés. L'éditeur de ligne de MuMATH est non fonctionnel, et l'usager doit se préoccuper des majuscules et des minuscules.

La structure hiérarchique de MuMATH est claire et bien expliquée dans le document d'accompagnement qui contient d'ailleurs tous les renseignements utiles sur le système.

MuMATH est utilisé le plus souvent en mode direct, pour obtenir des résultats de calcul numérique ou symbolique. On suggère également de l'utiliser pour vérifier les réponses à des exercices résolus sans l'aide de l'ordinateur, mais il n'est pas supérieur en cela au bon vieux solutionnaire. En apprentissage, il est beaucoup plus important d'identifier et de diagnostiquer les causes d'erreur que de vérifier l'exactitude des réponses. A cette fin, MuMATH ne peut être d'aucune aide, puisqu'il ne donne pas de résultats intermédiaires, ne justifie pas son cheminement, mais fournit seulement une réponse finale.

Par contre, MuMATH peut être utilisé avantageusement pour la solution de problèmes complexes comportant plusieurs étapes, et pour l'exploration d'exemples multiples et difficiles, qu'il serait trop onéreux d'aborder autrement.

Mentionnons, en terminant, que la puissance de MuMATH en ce qui a trait au calcul symbolique est difficile à évaluer. Un projet intéressant, réalisé dans le cours 203 (Calcul II) a consisté à faire exécuter à des groupes d'étudiants l'ensemble des intégrales indéfinies, définies et impropres contenues dans leur manuel de classe. Cet exercice leur a donné l'occasion de comparer et d'évaluer les résultats donnés dans le manuel, ceux fournis par MuMATH et ceux qu'ils obtenaient en résolvant eux-mêmes les exercices. Il a permis d'explorer les formes algébriques et de développer une attitude critique éclairée à l'égard des possibilités de l'ordinateur en calcul numérique et symbolique. (Voir Appendice B).

Le Tableau 4 résume les possibilités offertes par chacun des langages pour l'exploration des concepts appartenant aux différents domaines abordés dans les cours de mathématiques de niveau collégial. L'absence de mention indique que ce langage ne se prête pas particulièrement bien à l'exploration d'un concept donné. Ceci n'exclut toutefois pas la construction d'activités spécifiques reliées à l'apprentissage de ce concept, v.g. un dériveur en Logo (Giard, 1986) ou les approches numériques d'intégration dans pratiquement n'importe quel langage (voir Appendice B).

Tableau 4: Langages et concepts

	APL	BASIC	Logo	Q'Nial	CAL	MathCAD	MuMATH
Nombres	L	L	L	L	L	L	L
Fonctions et variables	D	P	D	D	P	P	L
Différentiation	-	-	-	-	P	P	L
Intégration	-	-	-	-	P	P	L
Espace géométrique	-	-	P	U	L	-	-
Espace algébrique	L	-	-	L	-	L	L
Probabilités et statistiques	D	-	-	U	-	L	-

L: opérations pré-définies dans le langage; U: opérations définies dans programmes utilitaires; P: partiellement disponibles dans le langage; D: opérations pouvant être définies par l'usager.

4.1.4 Les langages de programmation et le développement cognitif et méta-cognitif

La programmation est un processus isomorphe à la résolution de problèmes: les deux requièrent planification, codage, dépistage des erreurs, expérimentation et documentation.

Nickerson (1986) attribue à chacune de ces activités des pourcentages approximatifs du temps qu'un programmeur professionnel y consacre en moyenne:

a) planification: 25-50%
b) codage: 20-25%
c) dépistage des erreurs: 25-50%
d) expérimentation: 10-15%
e) documentation: souvent escamotée

Chacune de ces étapes requiert une activité mentale considérable. La planification exige l'organisation et la structuration de la pensée, l'établissement de priorités et d'ordres. Le codage requiert la connaissance du vocabulaire et de la syntaxe du langage, ainsi que la maîtrise de

certains procédés de programmation. La précision dans l'expression est obligatoire. Le dépistage des erreurs est le doute systématisé. Si le programme ne tourne pas ou ne produit pas les effets escomptés, il faut poser des questions, générer et tester des hypothèses: s'agit-il d'erreurs de syntaxe? de sémantique? de logique? L'ensemble du processus n'est possible que si la planification a été faite avec soin. Quand le programme tourne, il faut encore adopter une attitude critique face aux résultats. Sont-ils vraisemblables? exacts? précis? Dans quelles conditions sont-ils vrais? La solution peut-elle être invalidée? généralisée? Dans quels cas? La solution est-elle optimale? Enfin, la documentation des programmes est une étape capitale dans le contexte de l'apprentissage. C'est par ce moyen que l'usager rend son raisonnement et son interprétation des résultats explicites aux autres, justifie ses choix d'algorithmes, explique ce que le programme fait ou ne fait pas et dans quelle mesure il pourrait être utilisé pour résoudre d'autres problèmes.

Au cours des dernières années, l'existence de liens entre la programmation et le développement des habiletés de la pensée a fait l'objet de maintes assertions. Papert (1980) affirme que la programmation en Logo aide au développement de puissantes habiletés intellectuelles. Bork (1981) voit la programmation sur ordinateur comme un véhicule puissant pouvant aider au développement de la pensée analytique applicable à de larges catégories de problèmes. Nickerson (1986) considère la programmation comme une activité basée de façon intensive et extensive sur les connaissances.

Bien qu'il semble logique de supposer qu'une activité qui requiert de façon impérieuse l'utilisation d'habiletés cognitives et méta-cognitives puisse en favoriser le développement, il n'existe pas actuellement de preuve scientifique d'une telle relation. Cependant, des résultats de recherche expérimentale méthodologiquement valide et allant au-delà de l'introspective anecdotique ou personnelle commencent à émerger. Mayer et al. (1986) rapportent une étude selon laquelle l'apprentissage de la programmation peut avoir des effets positifs sur le développement des habiletés de la pensée qui sont directement reliées au langage appris. Nickerson abonde dans le même sens et émet l'opinion qu'on doit enseigner ces habiletés de façon explicite, plutôt que de laisser leur développement au gré du hasard.

Mais pour que ces développements puissent se réaliser, il faut que le langage de programmation soit suffisamment puissant pour permettre à l'usager de se concentrer sur la tâche, le problème à résoudre, plutôt que d'avoir à se préoccuper des contraintes du langage, ce qui est plutôt de nature à engendrer frustration et sentiments d'impuissance.

A ce sujet, il est intéressant de noter l'idée amenée par Hancock (1988) que l'apprentissage de la programmation serait facilité si l'apprenant pouvait se créer un modèle mental des programmes. Selon cet auteur, les langages fonctionnels et logiques correspondraient à des modèles explicites, l'équation et le prédicat respectivement, alors qu'il n'existerait pas de modèle explicite d'un programme en BASIC. Pour en construire un, l'usager doit se transformer en ordinateur pour simuler le programme pas à pas, ce qui peut difficilement être considéré comme un progrès du point de vue du développement cognitif et méta-cognitif.

Il serait futile de classifier les langages de programmation selon leur aptitude à favoriser le développement des habiletés cognitives et méta-cognitives, car tous les langages offrent des possibilités à ce point de vue, et surtout parce que tout dépend de l'usage qu'on en fait en situation didactique. En d'autres termes, tout dépend des activités d'apprentissage qui sont proposées aux étudiants.

4.2 Caractéristiques souhaitables

Deux observations se dégagent de façon assez évidente de l'étude qui précède: tous les logiciels soumis à l'étude présentent des caractéristiques qui les rendent utilisables dans une certaine mesure, à l'intérieur des cours de mathématiques, mais aucun ne réunit les qualités qui en feraient l'outil non pas idéal, mais meilleur que les autres. En fait, il est assez évident que seul un environnement intégré réunissant certaines caractéristiques pourrait répondre adéquatement aux besoins de la situation. En terminant ce chapitre, il a paru opportun de résumer brièvement les caractéristiques qui, au terme de cette recherche, apparaissent essentielles ou souhaitables dans un environnement de programmation utilisé pour l'enseignement et l'apprentissage des mathématiques de niveau collégial. Il va de soi que ces caractéristiques ne constituent qu'une première esquisse de ce que pourrait être un tel environnement, et que des recherches beaucoup plus approfondies et menées par une équipe pouvant compter sur des compétences variées seraient préalables à tout développement devant faire suite au présent projet.

4.2.1 Représentation et manipulation des objets

Le langage doit permettre de représenter, sous une forme respectant les propriétés relatives à leur définition et à leur structure, tous les objets traditionnellement étudiés dans les cours de mathématiques, en particulier les nombres, expressions numériques et algébriques,

fonctions et relations, variables, équations et inéquations, systèmes, propositions, ensembles, vecteurs et matrices. A cette fin, il doit supporter les structures de données permettant de représenter adéquatement les données complexes, par exemple, les structures de tableau et de liste.

Le langage doit contenir également le plus grand nombre possible d'opérations et de fonctions pré-définies permettant le calcul numérique et symbolique, ainsi que les manipulations opérant la transformation des structures de données. La notation des opérations mathématiques doit être le plus conforme possible à la notation traditionnelle et les résultats doivent être faciles à interpréter et à utiliser. A cette fin l'affichage en deux dimensions (pretty print) augmente la lisibilité des expressions en utilisant plusieurs lignes au besoin pour représenter numérateurs et dénominateurs, exposants et indices. La priorité des opérations doit être définie de façon à minimiser le nombre de parenthèses nécessaires, et la précision, suffisante. Le langage devrait reconnaître l'égalité des formes différentes d'un même nombre. La règle d'évaluation doit être définie rigoureusement et suivie de façon consistante.

4.2.2 Graphisme

La représentation graphique est indispensable à l'apprentissage des mathématiques: quel que soit l'espace considéré, la représentation des objets, des relations qui les unissent et des transformations qu'on leur applique aide la visualisation et la compréhension des phénomènes impliqués.

A cette fin, il est utile que soit intégré à l'environnement un mode graphique supportant la haute résolution, un système d'axes pour la représentation cartésienne, ainsi que les primitives nécessaires pour représenter par une image à l'écran, les valeurs générées par les programmes écrits dans ce langage.

4.2.3 Editeur

Le système doit permettre également l'accès facile à un éditeur pouvant remplir une double fonction. Pour la mise au point des programmes, l'éditeur doit connaître les construits du langage, par exemple les expressions conditionnelles, et les afficher de façon à illustrer leur structure, leur syntaxe et leur place dans la procédure (pretty print). Il doit aider au dépistage des erreurs en émettant des messages explicites, par exemple en affichant la ligne titre et la dernière ligne d'une procédure mentionnée dans un programme sans avoir été définie au

préalable. La possibilité de définir, encours d'exécution de programme, des procédures et des variables ayant une intérêt et une existence temporaire constitueraient également un atout de taille. Enfin, l'éditeur doit minimiser le nombre de caractères à taper pour les fonctions d'édition, ainsi que pour entrer et sortir de ce mode.

Pour l'édition de textes, l'éditeur doit assurer une qualité permettant la présentation de documents tels quels (solutions, devoirs, examens...), éliminant ainsi la nécessité d'avoir recours à d'autres logiciels pour rédiger ces documents de façon convenable.

4.2.4 Vocabulaire

Les principes qui prévalent actuellement dans le choix du vocabulaire de base d'un langage de programmation favorisent l'emploi de mots conservant leur sens du langage naturel. Un tel langage est nécessairement plus facile à apprendre, à mémoriser et à utiliser, et il en résulte des programmes plus lisibles, donc plus faciles à simuler et à corriger.

Cependant, si la tendance actuelle est de choisir des mots faciles à apprendre et à mémoriser, des mots suggestifs et relativement courts, il semble que leur sélection soit assez complexe. Selon Nickerson (1986), plusieurs chercheurs ont mis en évidence le peu de correspondances entre les noms utilisés par différents sujets pour désigner les mêmes opérations. Il y aurait même très peu de consistance entre les choix faits par un même sujet en des occasions différentes. La raison en serait probablement le peu de précision dans le sens des mots utilisés couramment, de sorte que les concepteurs du langage doivent tenir compte de la précision du terme autant que de la facilité de l'apprentissage par association.

Quant aux abréviations, même s'il existe un consensus quant à leur nécessité dans le cas de langages faisant usage de mots du langage naturel, il semble y avoir peu d'unanimité sur la façon dont elles doivent être déterminées. Il n'est absolument pas certain que le procédé de construction doive respecter la règle de formation des abréviations de la langue de référence, puisque celle-ci n'est pas nécessairement la plus naturelle ni celle qui a le plus de puissance mnémonique. Selon Moses et al. (1981), la performance dans la construction des abréviations est améliorée lorsque: 1. la règle d'abréviation consiste à tronquer les mots plutôt qu'à les contracter: 2. les abréviations sont de longueur fixe plutôt que dépendantes de la longueur du mot abrégé; 3. les utilisateurs connaissent la règle d'abréviation plutôt que de mémoriser les abréviations; 4. les déviations à la règle sont signalées.

Il est impossible de dire si ces résultats s'appliqueraient également à un langage faisant usage du français.

4.2.5 Syntaxe

Nickerson (1986) rapporte les résultats d'une recherche menée en 1980 sur l'importance de la syntaxe dans un langage descriptif, plus précisément un éditeur de texte. Selon cet auteur, la recherche démontra hors de tout doute que toutes les catégories d'utilisateurs (de niveau débutant, intermédiaire ou avancé) avaient eu des performances supérieures dans toutes les catégories d'épreuves en utilisant un éditeur de texte dont la syntaxe était similaire à celle de l'anglais. Bien que ces résultats ne prouvent pas que les mêmes différences subsisteraient après un usage prolongé, ni que les mêmes différences seraient constatées avec d'autres logiciels, il est assez normal de supposer que la proximité de la syntaxe d'un langage à apprendre avec celle d'un langage familier facilite la maîtrise du premier, et que les mêmes résultats pourraient être observés dans une situation impliquant le français.

Dans un contexte d'apprentissage, les points à surveiller sont les suivants:

-la syntaxe des expressions mathématiques doit être congruente avec la syntaxe traditionnelle en mathématiques;

-la syntaxe doit permettre des noms de variables et de procédures assez longs pour être significatifs et augmenter la lisibilité des programmes.

-la syntaxe doit être assez redondante pour augmenter la lisibilité des programmes, diminuer les risques d'erreurs, et en faciliter la correction.

-les erreurs de syntaxe doivent être repérées par l'analyseur syntaxique et des messages aidants doivent en résulter.

-la syntaxe utilisée pour la construction et la décomposition des structures de données doit permettre une représentation graphique, c'est-à-dire une visualisation de leur structure plutôt qu'une représentation de type algébrique.

-la syntaxe doit être extensible et permettre à l'usager de définir de nouvelles règles pour de nouvelles applications.

4.2.6 Styles de programmation

Afin de permettre la construction de tous les objets et de toutes les fonctions nécessaires aux besoins, le langage doit supporter plusieurs styles de programmation. La notion de style de programmation est une notion qui s'est précisée considérablement au cours des dernières années, à mesure que de nouveaux langages ont mis divers paradigmes en évidence. La programmation procédurale consiste à spécifier, étape par étape, le cheminement que l'ordinateur doit suivre pour construire les objets et produire les résultats désirés; la programmation fonctionnelle consiste à définir des fonctions qui retournent des expressions dont la valeur dépend uniquement de celles de qui sont données en entrée à ses variables, et qui sont dépourvues de toute autre propriété sémantique; la programmation logique permet l'étude des relations et la construction de preuves et de réfutations; la programmation orientée objet permet la construction d'objets, de classes d'objets et de hiérarchies de classes dotées de propriétés spécifiques; enfin, la construction de bases de données permet l'étude et l'exploration de vastes ensembles de connaissances. Tous ces styles permettent donc la réalisation d'activités différentes. Bien qu'il soit impossible (et probablement sans intérêt) qu'un seul langage intègre tous ces paradigmes, la versatilité des styles serait un atout de plus dans un logiciel utilisé à des fins d'apprentissage.

4.2.7 Gestion de l'espace de travail et des fichiers

Le système doit offrir une variété de fichiers, dont évidemment les fichiers permettant d'enregistrer les programmes, les données, l'espace de travail et le déroulement d'une séance interactive. La gestion des fichiers doit être le plus simple possible.

Il est souhaitable que la gestion de l'espace de travail exige de l'usager un minimum de connaissances nécessaires à l'allocation de la mémoire.

Le flot de l'information circulant entre l'ordinateur et l'usager doit être organisé de façon à permettre des entrées et sorties acceptables non seulement pour l'ordinateur mais aussi pour l'usager. Le langage doit permettre plusieurs modes d'entrée des données et autoriser l'accès à toutes les commandes du système d'exploitation. Un environnement du type offert par WINDOWS semble posséder ces caractéristiques.

4.2.8 Aide à l'usager

messages: les systèmes actuels émettent plusieurs genres

de messages qui sont de nature à aider l'usager qui en a besoin. Dans un premier temps, on pense aux messages classiques, dits "messages d'erreurs" parce qu'ils signalent à l'usager une erreur qui a été repérée par le système et qui empêche l'exécution du programme. Plus ces messages sont clairs, explicites et spécifiques, plus ils donnent de précision quant à la nature de l'erreur, plus l'identification et la correction de ces erreurs sera facile. Dans un contexte d'apprentissage, il est utile que la teneur des messages puisse être modifiée au besoin par le professeur.

Nickerson (1986) énumère d'autres types de messages que les concepteurs de systèmes aiment maintenant à inclure dans leurs logiciels. Ce sont, entre autres:

-la confirmation de commande: elle demande à l'usager de confirmer une commande dont l'exécution pourrait avoir des effets désastreux si elle était commandée par erreur ou en un temps inopportun (destruction d'un fichier ou recherches très longues, par exemple);

-l'annulation de commande (Undo, ou Forget it): elle retarde l'exécution d'une commande pour donner à l'usager la possibilité de changer d'idée, ou d'annuler une commande en cours d'exécution;

-la commande "assez": elle permet d'interrompre l'exécution d'une commande parce que les résultats partiels sont jugés satisfaisants;

-le message qui indique qu'une commande contenant une faute de syntaxe a été acceptée par le système, et qui signale l'erreur à l'usager de même que la commande corrigée;

-un signal rappelant à l'usager si le contrôle lui appartient ou s'il revient à l'ordinateur;

pauses: les pauses en cours d'exécution constituent des moyens efficaces dans la correction et la mise au point des programmes; elles devraient pouvoir être intercalées dans les programmes ou commandées du clavier en cours d'exécution. Elles permettent à l'usager d'avoir accès aux valeurs des variables, et d'avancer pas à pas dans l'exécution du programme.

La nature de ce qui se passe pendant l'interruption ne devrait pas être fixée de façon trop rigide. Idéalement, il faudrait que toute la puissance du langage, y compris celle de l'analyseur, soit disponible à l'usager pendant les pauses. Ceci lui permet de redéfinir la procédure erronée, de faire exécuter une ou plusieurs autres procédures, d'examiner des structures de données complètes, puis de reprendre l'exécution du programme au point suivant

l'interruption.

système d'aide: il devrait y avoir un bon système d'aide intégré. L'usager devrait pouvoir demander la définition d'une procédure du système, obtenir des rappels de syntaxe, de concepts ou de procédures. Idéalement, l'aide devrait être ajustée aux divers niveaux d'expertise, dispensant des informations complètes ou des rappels succincts selon les besoins.

Le système d'aide peut être organisé en une suite d'écrans de dépannage, ou géré par voie de commandes ou de menu. L'expérience démontre toutefois que ce procédé n'est pas toujours efficace, surtout si la décomposition en arbre ne correspond pas à la structure intellectuelle de la majorité des usagers. De toute façon, un bon système d'aide doit avoir plusieurs portes d'accès et de sorties. L'efficacité maximum semble être obtenue au moyen d'une interface qui dispense l'information appropriée sur la base d'un dialogue où l'usager peut préciser ses besoins.

Dans le même ordre d'idées, et à mesure que seront disponibles des systèmes `comprenant' les programmes, on devrait pouvoir disposer de moniteurs auxquels des techniques d'aide de dépistage et de correction des erreurs ont été incorporées. Ces moniteurs pourraient avertir l'usager d'inconsistances potentielles lorsque des modifications sont envisagées. Enfin, il pourrait être possible à l'usager de spécifier certaines conditions que le moniteur se chargerait de vérifier.

bibliothèque: finalement, le langage devrait contenir, surtout à l'intention des débutants, une bibliothèque de procédures que ceux-ci pourraient utiliser comme sous-procédures dans leurs programmes, ce qui leur permettrait d'atteindre plus tôt le stage où ils pourraient écrire des programmes performants. Tous les usagers devraient avoir accès à cette bibliothèque, de sorte que si un nouvel item y est ajouté, il puisse aussitôt être partagé par les autres. Parmi ces programmes, il serait souhaitable de trouver un module permettant une initiation interactive au langage.

4.2.9 Aide à l'enseignement

Toutes les dispositions spécifiées au paragraphe précédent fournissent évidemment un support à l'enseignement. Un dernier point qu'il importe de mentionner ici est la possibilité de concevoir des espaces de travail, ou micro-mondes, destinés à supporter des apprentissages particuliers.

Au cours des dernières années, le terme "micro-monde", popularisé par Papert (1980), a été utilisé par plusieurs

auteurs pour désigner des réalités assez différentes les unes des autres. Thompson propose le terme "micro-monde mathématique" pour désigner un système comprenant des objets, des relations définies entre ces objets et des opérations qui transforment ces objets et ces relations. L'isomorphisme avec un système mathématique est évident: ces micro-mondes, qui ne contiennent au départ que des termes et des propositions élémentaires, ont un potentiel qui ne demande qu'à être étendu par l'activité des étudiants.

Il va de soi que l'environnement de programmation doit pouvoir être implanté sur réseau, afin de permettre à l'usager un accès facile aux modules dont il a besoin à un moment donné. Il est également souhaitable que le logiciel de réseau permette l'échange d'information entre les différents postes de travail, afin de permettre la collaboration dans la résolution de problèmes.

4.2.10 Documentation

L'existence de programmes utilitaires proposant une initiation interactive à l'utilisation du logiciel, et même l'existence de fichiers et de systèmes d'aide intégrés, ne diminuent pas l'importance d'une documentation de qualité. Il peut paraître superflu d'aborder ce point ici, mais il faut avoir utilisé un logiciel qui en est dépourvu pour estimer ce point à sa juste valeur. En fait, cette lacune se rencontre assez souvent avec les logiciels conçus pour les besoins de l'éducation, mais comme ils sont généralement simples à utiliser, cette lacune passe souvent inaperçue. Par contre, il est très difficile de se servir d'un langage de programmation qui n'est pas accompagné d'un bon manuel de référence. Comme ces documents sont dispendieux à produire et à mettre à jour, il arrive souvent que les producteurs de logiciels choisissent d'épargner sur cet item, ce qui explique qu'il n'existe à peu près pas de documentation en langue française pour ce type de logiciel.

Pour utilisation dans un contexte d'apprentissage, il est donc important que la documentation soit abondante et bien organisée, qu'elle soit présentée de façon à en faciliter la consultation en position de travail à l'ordinateur, et qu'elle soit pourvue de glossaires, de vocabulaires et d'index qui permettent de trouver rapidement les renseignements désirés. Enfin, elle devrait être redondante et organisée en hiérarchie, c'est-à-dire présenter des explications à plusieurs niveaux allant de la vérification d'un détail de syntaxe à la recherche d'explications détaillées sur la structure du langage ou sur l'allocation de la mémoire.

En somme, elle devrait être conçue pour l'étudiant dans un contexte d'apprentissage des mathématiques.

APPENDICE A

La première partie de cette recherche a consisté à préciser les objectifs visés en éducation mathématique au collégial, à partir des contenus des principaux cours qui y sont offerts. Elle a mis en évidence que ces objectifs relèvent des mathématiques elles-mêmes: construction de concepts, acquisition de procédures, maîtrise d'un langage, ainsi que du développement d'habiletés générales reliées à la cognition et à la méta-cognition.

Dans la seconde partie, l'attention a été mise sur les langages de programmation. Un bref historique de leur développement ainsi qu'un tour d'horizon rapide des logiciels existants ont permis de déterminer quelques caractéristiques à partir desquelles ces langages pouvaient être comparés, et d'en sélectionner quelques-uns pour fins d'évaluation.

La dernière partie de la recherche a porté, comme il se devait, sur l'évaluation de ces langages considérés en tant que systèmes formels de notation et en tant qu'outils pouvant favoriser l'atteinte des objectifs visés en éducation mathématique. Il en est ressorti que les langages présentent entre eux des différences importantes, et que certains langages permettent mieux, c'est-à-dire d'une façon plus naturelle, l'échange d'information à caractère mathématique entre l'ordinateur et l'apprenant.

La conclusion de cette recherche est à l'effet que, pour répondre adéquatement aux besoins de la situation, un environnement devrait intégrer les possibilités offertes par les logiciels de calcul numérique et symbolique, un langage doté de caractéristiques précises et supportant plusieurs styles de programmation, un environnement convivial comprenant entre autres un mode graphique et un éditeur, le tout accompagné d'une documentation orientée vers l'utilisation de ce logiciel à des fins d'enseignement et d'apprentissage des mathématiques.

Cette conclusion est-elle justifiée?

L'informatique met aujourd'hui à la disposition de l'éducation mathématique bon nombre d'outils appartenant à la catégorie des logiciels éducatifs ou à celle des langages de programmation. Elle nous promet pour demain les systèmes d'enseignement intelligemment assistés par ordinateur.

Les logiciels éducatifs conçus jusqu'ici sont généralement destinés à supporter l'apprentissage d'une notion ou le développement d'une habileté particulière. Les plus limités se bornent à la présentation de contenu ou d'exercices, les meilleurs permettent la génération d'hypothèses et leur validation sur un grand nombre de cas. En général ils sont faciles d'accès et d'utilisation, et certains mettent à profit les caractéristiques propres à l'ordinateur, telles le graphisme ou le feed-back immédiat. Mais comme ils ne permettent qu'un type d'interaction, ils ne sont souvent utilisés que pendant peu de temps à l'intérieur d'un cours donné, si bien que la maîtrise du logiciel ne représente pas un acquis dans le cheminement d'apprentissage. On dit aussi que ces logiciels, qui reproduisent le modèle de l'enseignement dispensé selon la séquence exposé-exemples-exercices, permettent souvent bien peu d'autonomie et de créativité de la part de l'usager. En ce sens, ils rappellent l'époque précédant l'apparition de la micro-informatique, époque où il existait peu de langages suffisamment faciles d'accès pour permettre leur utilisation dans un contexte d'apprentissage, et peu d'enseignants assez familiers avec l'ordinateur pour ne serait-ce qu'envisager une telle approche, d'où la nécessité de concevoir des outils permettant d'introduire l'ordinateur dans les salles de classe.

Pour ce qui est des langages de programmation, la recherche a montré qu'il est possible de mettre à profit les caractéristiques particulières à chacun pour favoriser l'atteinte de divers objectifs de l'éducation mathématique. Les langages de 3e génération sont d'un usage plus général, et permettent de concevoir des activités d'apprentissage dans pratiquement tous les domaines des mathématiques. Ceux de 4e génération sont plus faciles d'accès mais leur puissance est limitée à des domaines plus restreints. Aucun de ces langages n'offre toutes les capacités qu'il serait souhaitable de trouver dans un environnement de programmation destiné à supporter l'enseignement et l'apprentissage des mathématiques. Pour ce qui est du calcul numérique et de l'exploration des fonctions et des variables, Logo et APL font l'affaire. Le calcul infinitésimal requiert les capacités de MuMATH. En algèbre linéaire, on aurait raison de préférer APL. Pour la puissance de manipulation des objets et des structures de données, de même que pour la versatilité dans les styles de programmation et la variété des applications, il faudrait

disposer de la puissance de Q'Nial. Enfin pour ce qui est de la rédaction de documents, exercices, devoirs et examens, MathCAD est actuellement l'outil le plus sophistiqué qui soit. Il en résulte que pour couvrir l'ensemble des mathématiques de niveau collégial, sans parler de celles du secondaire et de l'université, il faudrait demander à l'étudiant de maîtriser plusieurs langages dont l'usage est limité à un nombre restreint d'applications.

Pourquoi dès lors ne pas attendre l'apparition des tuteurs intelligents?

Un symposium récent (juin 88) a réuni à Montréal un échantillon impressionnant de la communauté des chercheurs en intelligence artificielle appliquée à l'éducation. On y a présenté et défendu plusieurs systèmes actuellement au stade de l'expérimentation, et dont on dit qu'ils assureront une interaction apprenant-ordinateur modelant de près la relation traditionnelle étudiant-professeur. Tout en se défendant bien de vouloir remplacer l'enseignant, leurs concepteurs espèrent ainsi suppléer à certaines lacunes de l'enseignement traditionnel, particulièrement en ce qui a trait à la définition et au séquencement des objectifs, à la planification de l'enseignement, au suivi individuel, au diagnostic des erreurs et à la somme des connaissances requises pour dispenser un enseignement de haut niveau.

Il est encore trop tôt pour prédire ce que seront réellement ces systèmes ou ce qu'on pourra en faire dans l'enseignement. Il est certain que leur puissance sera de beaucoup supérieure à celle des logiciels éducatifs actuels, et qu'il y aura des utilisateurs pour ce genre d'outils. Cependant, on ne peut nier qu'ils imposent des demandes considérables aux systèmes informatiques, en termes de mémoire et de rapidité d'exécution, et ce dans le but de modeler des fonctions remplies actuellement, quoique à des degrés de performance inégaux, par des acteurs humains, via le dialogue étudiant-professeur. On peut choisir d'affecter la puissance des machines à d'autres usages, et c'est l'argument sur lequel la conclusion de cette recherche est fondée.

Cette conclusion découle d'une conception dynamique de l'apprentissage qui considère ce dernier comme un processus actif de construction de concepts et de développement d'habiletés, processus qui a son point de départ dans l'activité de l'apprenant lui-même. Un environnement ouvert, favorisant de multiples formes d'activité, selon la nature des apprentissages visés, offre un meilleur terrain pour l'implantation de cette approche d'apprentissage qu'un environnement dans lequel la nature des interactions est, dans une certaine mesure, déterminée à l'avance. L'éducation mathématique de niveau collégial a besoin d'outils puissants et performants qui permettent à l'étudiant de se libérer du

labeur non authentique qui accompagne souvent la solution de problèmes scolaires, de concevoir et de réaliser des projets significatifs, de construire ses connaissances par l'exploration et la découverte, de développer son expertise par la pratique, tout en concentrant son attention sur les mathématiques et non sur l'informatique. Dans la conjoncture actuelle, un grand nombre de professeurs, soucieux de trouver l'équilibre souhaitable entre les habiletés traditionnelles et la maîtrise des habiletés que rendent possible les nouvelles technologies, cherchent de tels outils. Le besoin est présent ainsi qu'en témoigne l'intérêt manifesté pour ces travaux de recherche au cours des cinq dernières années, au Québec, au Canada et à l'étranger.

La maîtrise d'un tel outil demande-t-elle plus de temps et d'effort? Qu'à cela ne tienne! l'utilité à long terme peut justifier cet investissemment. D'ailleurs cette objection tombera (ou vaudrait-il mieux utiliser ici un conditionnel?) d'elle-même quand on cessera d'envisager la formation de l'étudiant comme une boîte à compartiments dans laquelle chaque discipline tente de s'approprier le plus d'espace possible, et qu'on consentira à y inclure des apprentissages pouvant desservir plusieurs disciplines.

En admettant donc qu'un environnement de programmation tel que spécifié plus haut réponde à des besoins réels, on peut encore se demander comment il se fait qu'un tel outil n'existe pas déjà.

En réponse à cette question, on pourrait invoquer des raisons économiques, tel le manque d'équipement et de ressources, des raisons académiques, tel le manque de préparation du personnel enseignant et le manque de matériel didactique; mais en toute honnêteté, il faut reconnaître que l'éducation mathématique a pris jusqu'ici bien peu d'initiatives pour faire connaître ses besoins à l'industrie informatique.

Il est d'ores et déjà certain que la conception d'un tel projet de développement est une entreprise ambitieuse. Sa réalisation exigerait une approche systémique permettant de prendre en considération tous les aspects de la situation, et la constitution d'une équipe réunissant les compétences d'informaticiens, d'ingénieurs en conception de systèmes, de spécialistes de l'apprentissage et de l'éducation mathématique, de praticiens de l'enseignement à plusieurs niveaux, voire de linguistes. De plus, ces individus devraient parler suffisamment le même langage pour être capables de partager les préoccupations des autres membres de l'équipe.

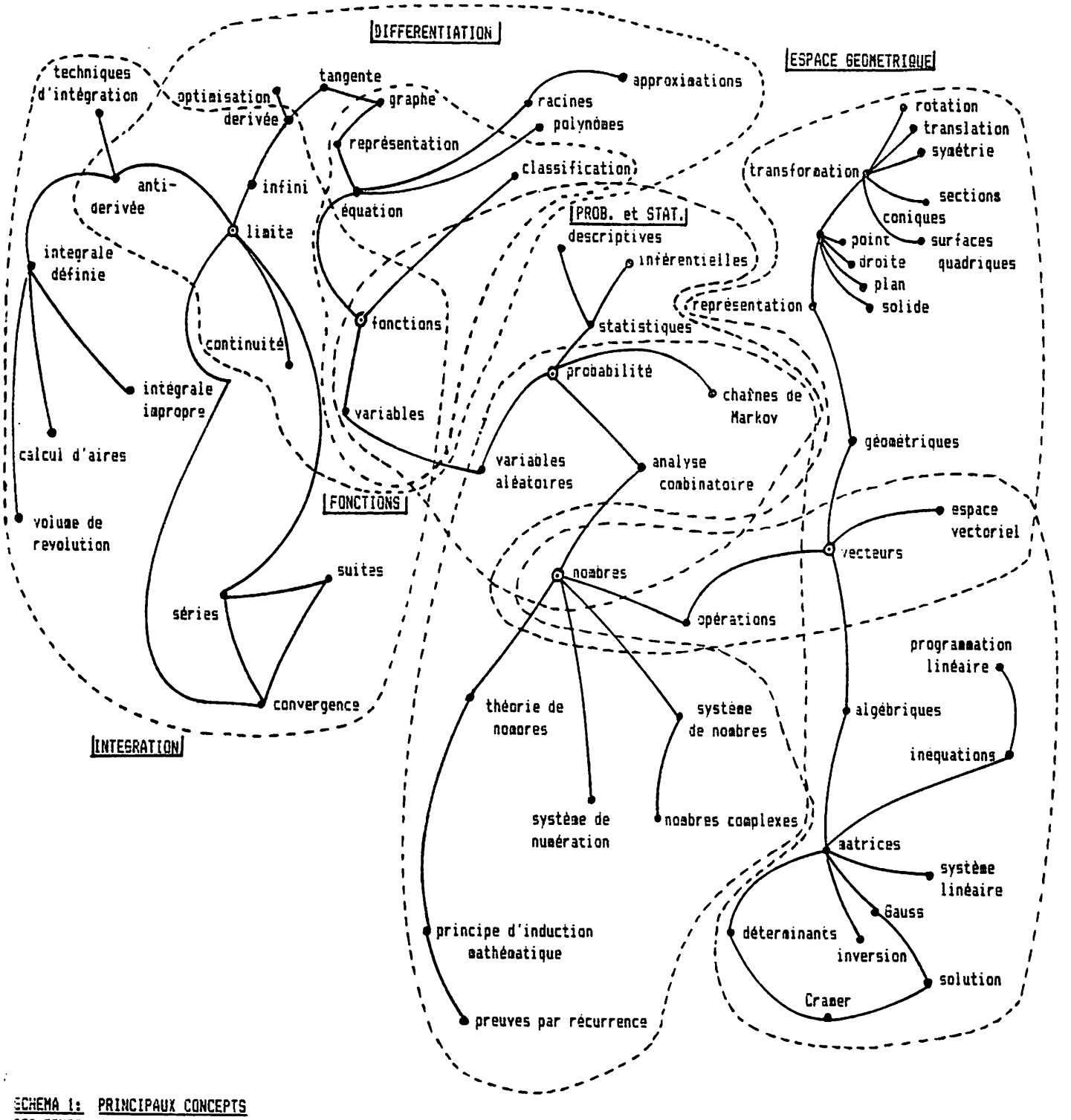
Tout cela est-il bien raisonnable, considérant l'exiguité du milieu québecois?

Même sans vouloir réinventer la roue, c'est-à-dire même en utilisant le plus possible les outils existant déjà, il est certain que leur organisation dans un environnement intégré nécessiterait une mise de fonds appréciable. Mais quand on considère les montants qui sont encore investis annuellement dans la production de logiciels éducatifs à usage plutôt restreint, cette dépense n'est pas aussi considérable qu'elle le paraît à première vue. Ce projet est à la mesure d'un Québec, qui possède les compétences et les ressources nécessaires pour le mener à terme et ainsi répondre à des demandes légitimes du secteur de l'éducation.

Tout au long de ce rapport, la question linguistique n'a été que brièvement évoquée. Il n'en demeure pas moins que, mis à part les logiciels éducatifs développés ici, tous les outils utilisables au collégial ne font usage que de l'anglais. Si, pour faciliter la transférabilité des applications, cet état de choses est jugé acceptable dans l'entreprise, il l'est beaucoup moins en éducation où il impose aux apprenants un va-et-vient entre les notations française, anglaise, mathématique et informatique. Le Québec, on l'a assez répété, est dans une situation particulière en Amérique du Nord. Mais le droit à la différence se paye à même les ressources du milieu: si nous ne nous donnons pas les outils culturels nécessaires à notre développement, qui nous les donnera?

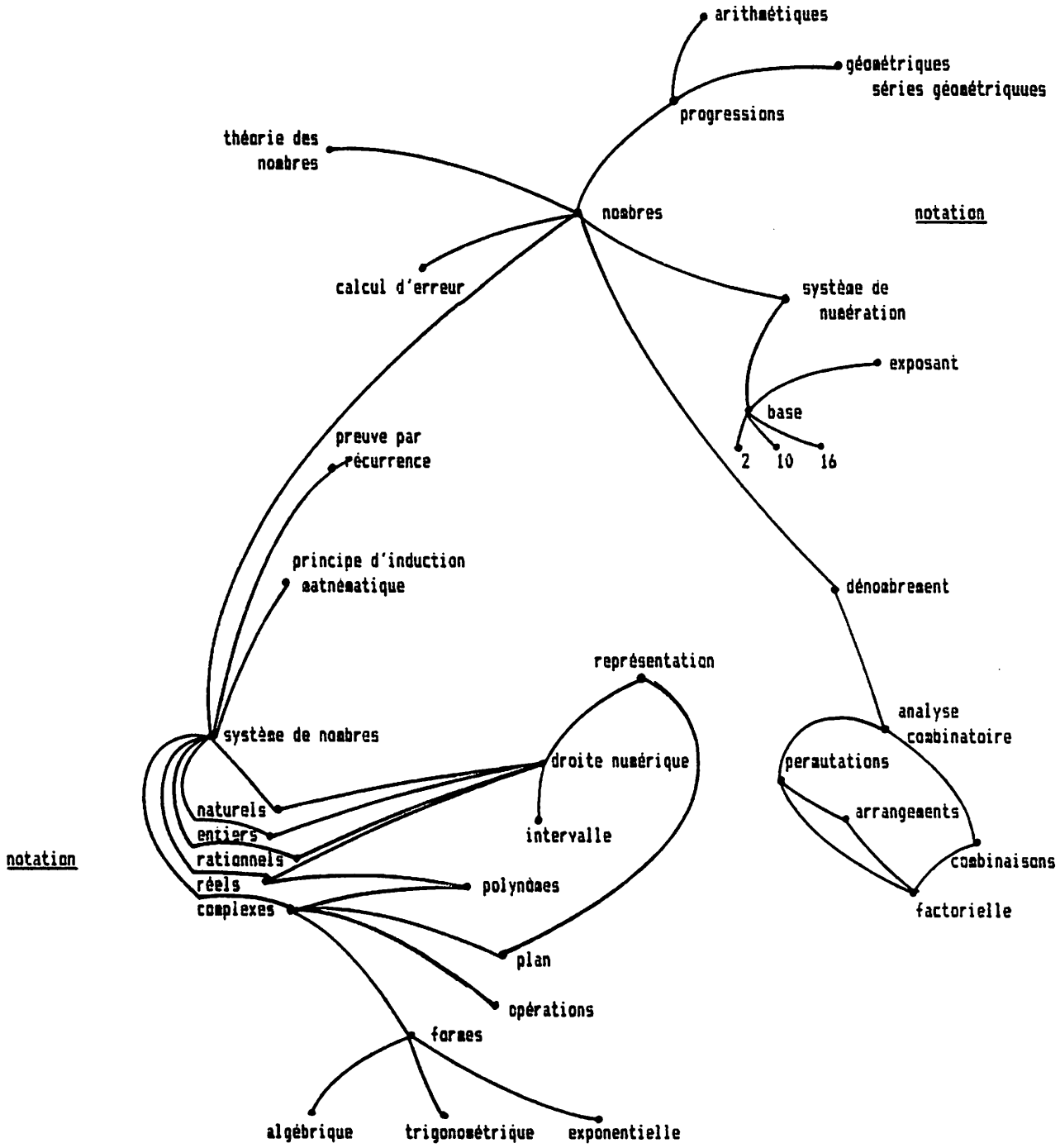
Cette dernière considération ne doit toutefois pas mener à la conclusion qu'il suffirait de traduire des logiciels existants pour obtenir un environnement de programmation intégré répondant aux besoins de l'enseignement et de l'apprentissage des mathématiques. Cet outil puissant, et qui devrait être conçu pour évoluer et s'adapter aux configurations futures des programmes de formation et des systèmes informatiques, n'existe pas actuellement. Il correspond à de nouveaux objectifs et permettrait enfin à l'éducation de travailler avec des outils créés pour l'apprentissage, plutôt que de continuer à adapter l'apprentissage aux outils disponibles.

CONCLUSION



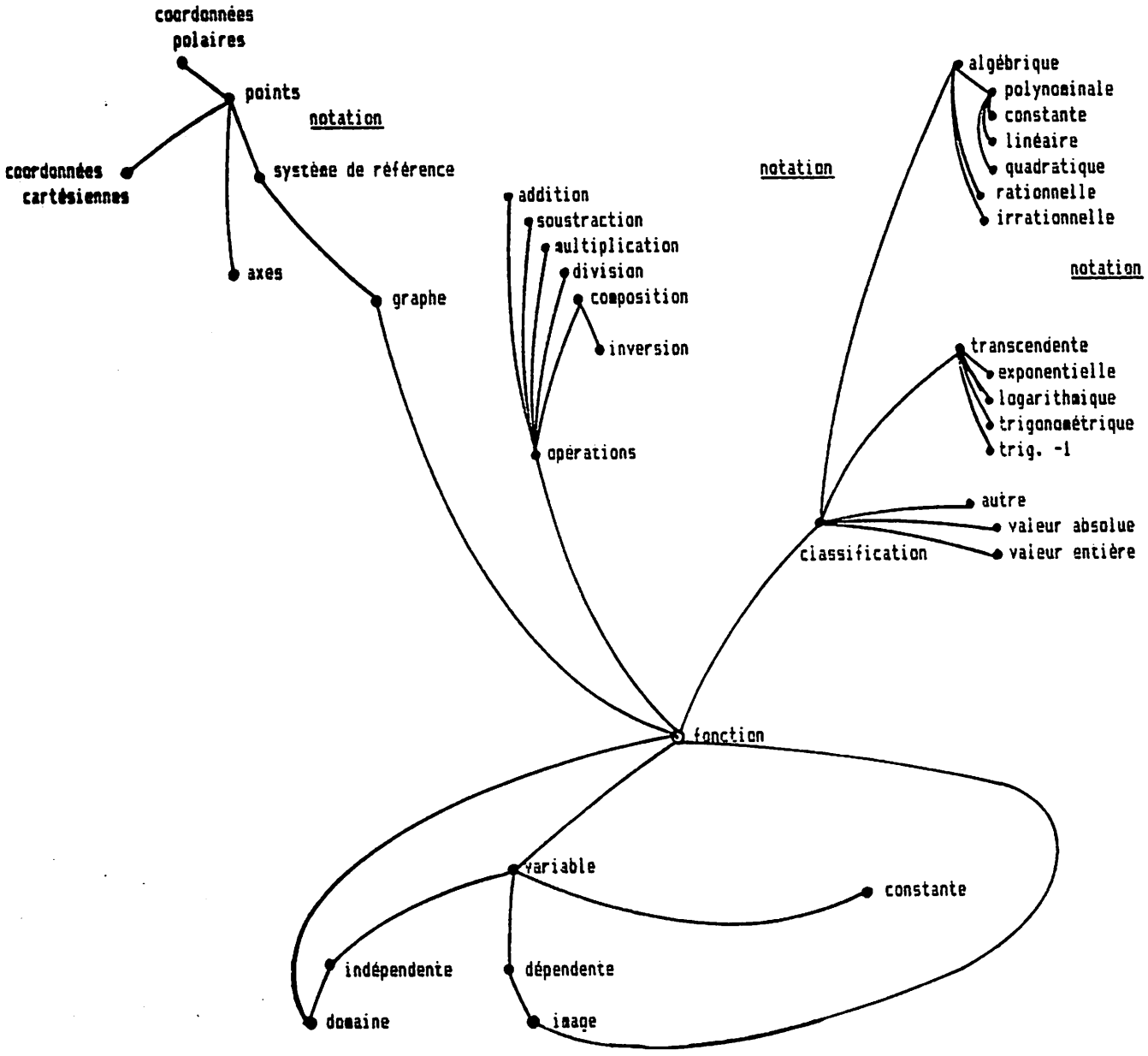
SCHEMA 1: PRINCIPAUX CONCEPTS DES COURS DE MATHEMATIQUES DE NIVEAU COLLEGIAL (101, 103, 105, 203, 307, 337)

I NOMBRES



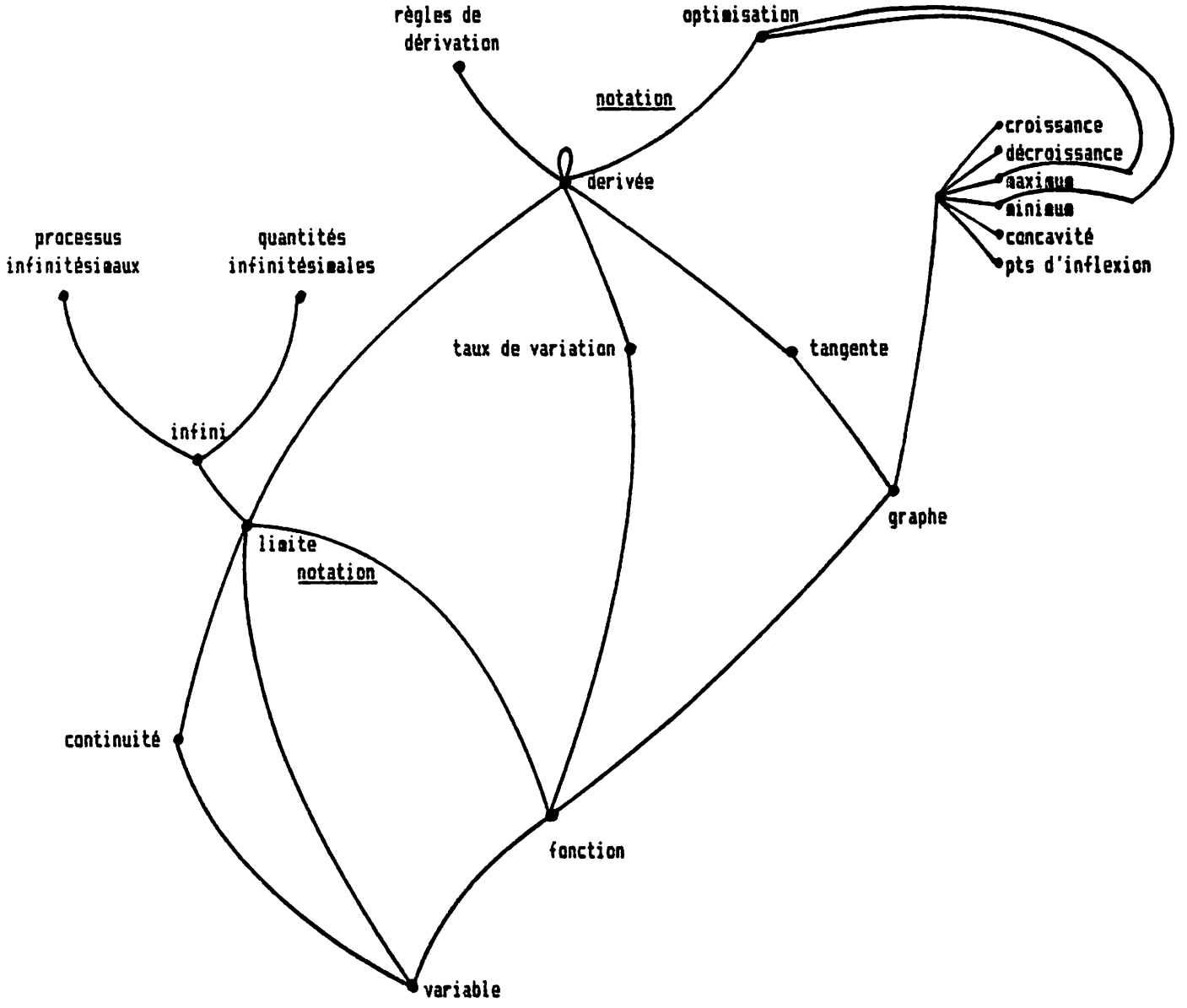
SCHEMA 2

II FONCTIONS ET VARIABLES



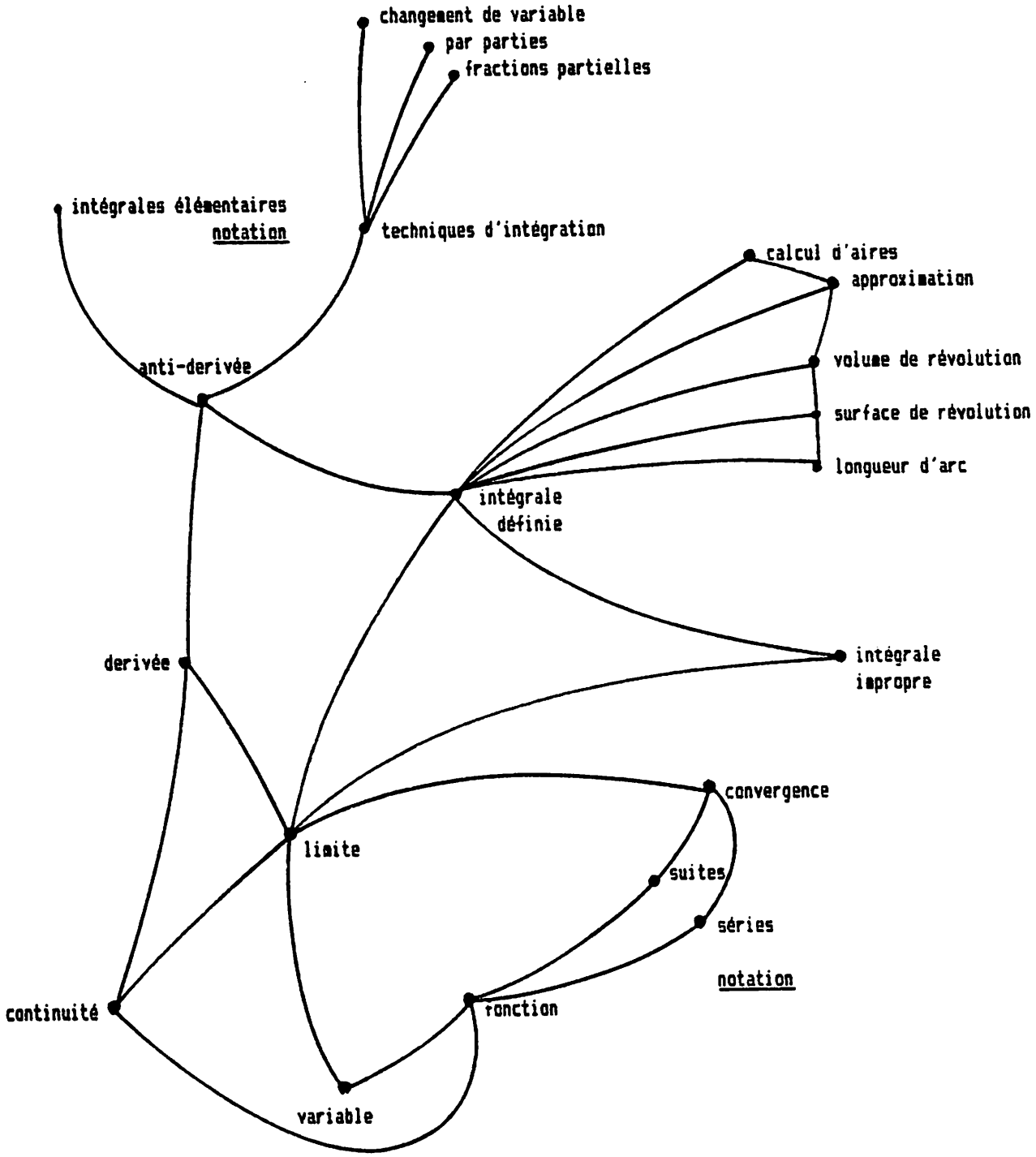
SCHEMA 3

III DIFFERENTIATION



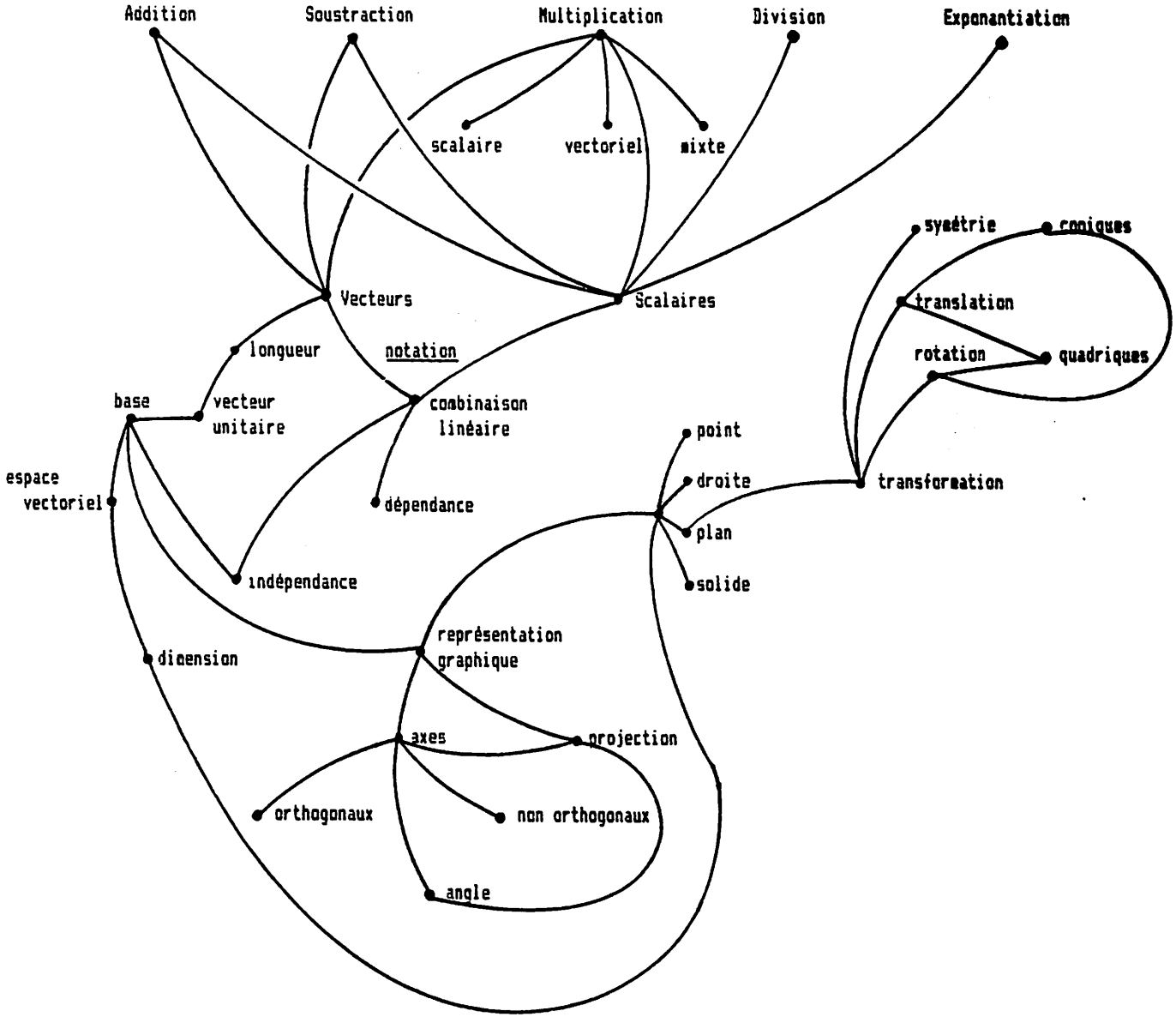
SCHEMA 4

IV INTEGRATION



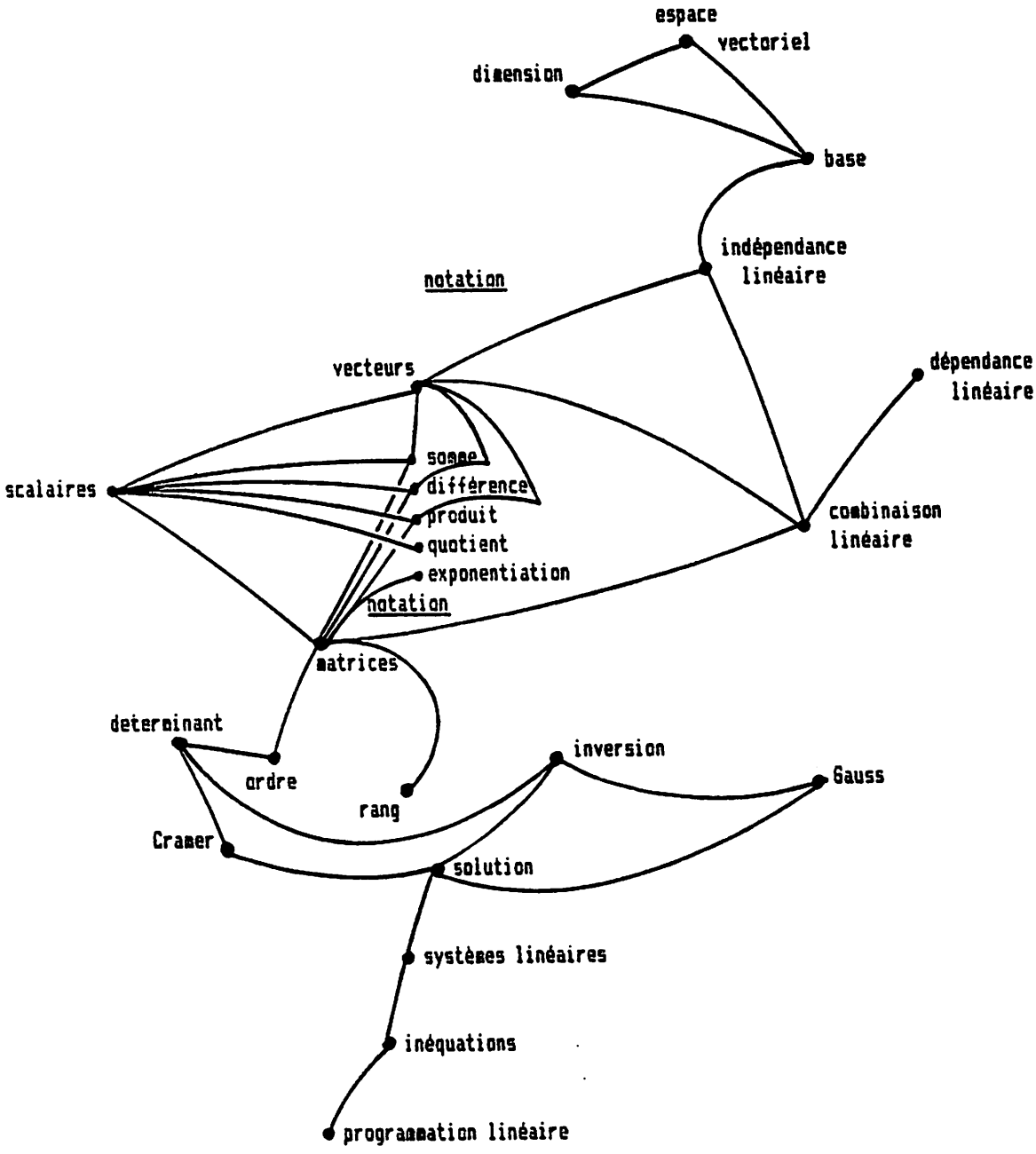
SCHEMA 5

V GEOMETRIE



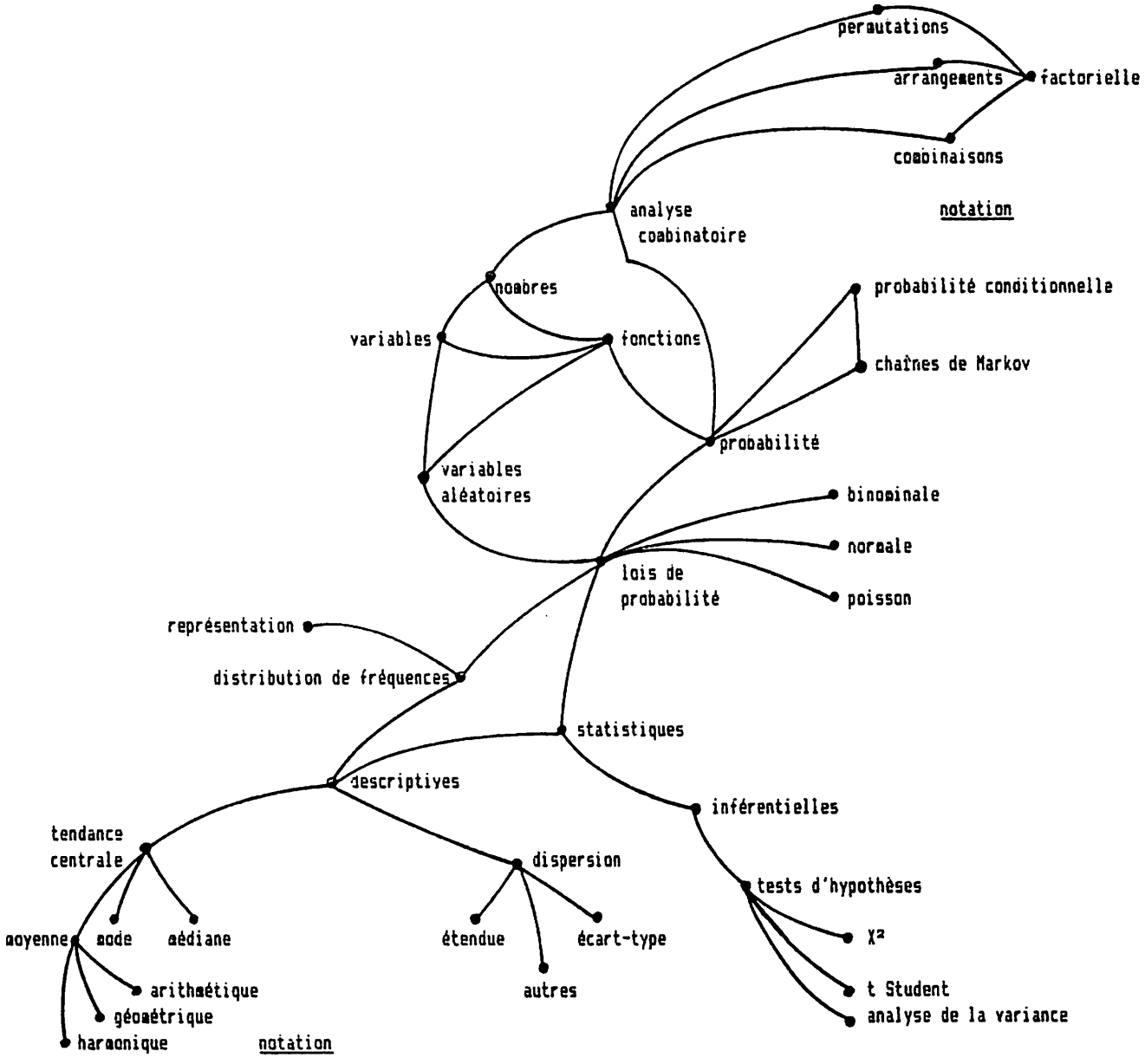
SCHEMA 6

VI ALGEBRE LINEAIRE



SCHEMA 7

VII PROBABILITES ET STATISTIQUES



SCHEMA 8

APPENDICE B

```
╔═══════════════════════════════════════╗
‖           The APL*PLUS/PC             ‖
‖    Application Development System      ‖
‖═══════════════════════════════════════‖
‖  Version 4.2   Serial Number 435051    ‖
‖      Copyright 1984, STSC, Inc.       ‖
‖  All rights reserved.  Unauthorized    ‖
‖    reproduction of this software is    ‖
‖    prohibited and violates U. S.       ‖
‖    Copyright Laws.                     ‖
‖  APL*PLUS is a registered trademark    ‖
‖    and service mark of STSC, Inc.      ‖
╚═══════════════════════════════════════╝
0 BANAPLDL SAVED 2/12/1985 16:40:12
Printer options reset
APL characters downloaded
Use outport 10 when using DARBIN
```

Laboratoire

(L. Lapointe et M. Delisle, Collège de Sherbrooke)

1. Entrez sur le réseau. Recouvrez votre bloc de travail.

2. Un peu de logique (primitives logiques)

```
      3 = 5
0
      ~1
0
      ~0
1
      P ← 1 1 0 0
      Q ← 1 0 1 0
      ~P
0 0 1 1
      P ∧ Q
1 0 0 0
      P ∨ Q
1 1 1 0
      (3 > 2) ∧ 10 < 50
1
      (3 > 10) ∧ 10 < 50
0
      (3 > 10) ∨ 10 < 50
1
      1 0 0 1 1 / 5 10 15 20 25
5 20 25
```

3. Vous êtes propriétaire d'une quincaillerie. La semaine dernière, vous avez vendu 12 marteaux, 9 planches, 24 crochets et 4 tournevis. Cette semaine, vous avez vendu respectivement 10, 14, 7 et 5 pièces de ces mêmes articles.

Formons la matrice des items:

```
ITEMS ← 4 9 ⍴ 'MARTEAUX PLANCHES CROCHETS TOURNEVIS'
```

et la matrice des quantités:

```
  QUANT ← ⍉ 2 4 ⍴ 12 9 24 4 10 14 7 5
```

et vérifions:

```
        ITEMS                QUANT
  MARTEAUX              12 10
  PLANCHES               9 14
  CROCHETS              24  7
  TOURNEVIS              4  5
```

Pour juxtaposer ces deux matrices, il faut encoder (transformer en caractères) la matrice numérique:

```
      ITEMS, ⍕QUANT
 MARTEAUX  12 10
 PLANCHES   9 14
 CROCHETS  24  7
 TOURNEVIS  4  5
```

a) Vous arrive-t-il de vendre plus d'une douzaine d'un article en une semaine?

```
     QUANT > 12
                              (,QUANT > 12) /,QUANT
 0 0                    14 24
 0 1
 1 0
 0 0
```

c) Donner les items ou au moins une des deux valeurs est supérieure à 12.

```
      ∨/QUANT > 12
 0 1 1 0
```

d) Quel est le numéro de ces lignes dans la matrice?

```
      ⍳ ← NL ← (∨/QUANT > 12) / ⍳4
 2 3
```

e) Donner les items dont la vente a dépassé la douzaine au moins une fois.

```
      ITEMS [NL;]
 PLANCHES
 CROCHETS
```

<u>Envoyez à l'imprimante (et annexez à votre rapport) votre travail sur les numéros 4 et 5</u>.

4. Si ces items coûtent respectivement 12.98$, 4.25$, 1.37$, et 6.49$,

a) combien vous a rapporté la vente de ces items au cours de la 1ère semaine et au cours de la 2e semaine? (Obtenez les deux résultats en une seule opération)

```
   C ← 1 4 ⍴ 12.98 4.25 1.37 6.49
   C+.×QUANT
252.85 231.34
```

b) combien a-t-elle rapporté au total pour les deux semaines?

```
      +/C+.×QUANT
484.19
```

5. Voici le solde du compte de 10 importants clients d'une caisse populaire:

```
COMPTES ← 135 27 ¯15 12 ¯25 ¯57 240 172 18 29
```

Tapez les expressions qui donneront la réponse aux questions suivantes:

a) Est-ce que tous les soldes sont positifs?

```
      ∧/COMPTES > 0
0
```

b) Combien de comptes sont dans le rouge?

```
      +/COMPTES < 0
3
```

c) Combien de comptes ont une balance positive>

```
      +/COMPTES > 0
7
```

d) Y a-t-il des balances supérieures à 200$?

```
      ∨/COMPTES > 200
1
```

e) Combien?

```
      +/COMPTES > 200
1
```

f) Y a-t-il des soldes entre 120$ et 175$?

```
      ∨/(COMPTES > 120) ∧ COMPTES < 175
1
```

g) Combien?

```
      +/(COMPTES > 120) ∧ COMPTES < 175
2
```

h) Quel est le total de tous les soldes?

```
      +/COMPTES
536
```

i) Quel est le total des soldes dans le rouge?

```
      +/COMPTES < 0
3
```

```
GW-BASIC 2.01
(C) Copyright Microsoft 1983,1984

Olivetti Personal Computer - GW-BASIC Rev. 1.02
Copyright (C) by Olivetti, 1984 - all rights reserved

62093 Bytes free
Ok
```

PROBLEME: Utiliser l'algorithme de la division d'Euclide pour trouver si un nombre est premier.

```
list
10 PRINT "TEST POUR NOMBRES PREMIERS"
20 INPUT "DONNE UN NOMBRE"; N
30 IF N = 2 THEN 130
40 IF N = INT (N/2) * 2 THEN 150
50 IF N = 3 THEN 130
60 D = 3
70 Q = INT (N/D)
80 R = N - Q * D
90 IF R = 0 THEN 150
100 IF D ^ 2 > N THEN 130
110 D = D + 2
120 GOTO 70
130 PRINT N; "EST PREMIER"
140 PRINT: PRINT: GOTO 20
150 PRINT N; "N'EST PAS PREMIER"
160 GOTO 140
170 END
Ok_

1LIST   2RUN 3LOAD"  4SAVE"  5CONT6,"LPT" 7TRON8TROFFKEY    0SCREEN

TEST POUR NOMBRES PREMIERS
DONNE UN NOMBRE? 53
 53 EST PREMIER


DONNE UN NOMBRE? 86
 86 N'EST PAS PREMIER


DONNE UN NOMBRE? 1237E5
 1.237E+08 N'EST PAS PREMIER


DONNE UN NOMBRE? 28 * 37
?Redo from start
DONNE UN NOMBRE? 2388751
 2388751 N'EST PAS PREMIER
```

PROBLEME: TRACER UN SYSTEME D'AXES A GRADUATION VARIABLE
DE FACON A POUVOIR REPRESENTER LE GRAPHE DE N'IMPORTE
QUELLE FONCTION A L'ECHELLE DESIREE.

```
PO GRAD :A :N
REPETE :N [AV :A GA 90 AV 1 RE 2 AV 1 DR 90]
RE :N * :A
FIN

PO AXESGRAD :UX :UY
FENETRE CT ORIGINE
REPETE 2 [GRAD :UY ENTIER 200 / :UY DR 90 GRAD :UX ENTIER 320 / :UX DR 90
FIN

PO REMPLACER :A :B :L
SI VIDEP :L [RT :L]
SI EGALP PREMIER :L :A [RT PH :B REMPLACER :A :B SP :L]
RT PH PREMIER :L REMPLACER :A :B SP :L
FIN

PO GRAPHE :F :X2
SI XCOR > :X2 [STOP]
BC FPOS COUPLE XCOR + 1 :F
GRAPHE :F :X2
FIN

PO COUPLE :X :F
RETOURNE LISTE :X EXECUTE :F
FIN

PO IMAGE :X :F
RETOURNE EXECUTE :F
FIN

PO TGRAPHE :F :INT
LC FPOS COUPLE PREMIER :INT :F
GRAPHE :F DERNIER :INT
FIN

PO TRACER :F :INT :UX :UY
TGRAPHE (PHRASE [:UY * (] REMPLACER ":X [( :X / :UX )] :F [)]) :INT
FIN
```

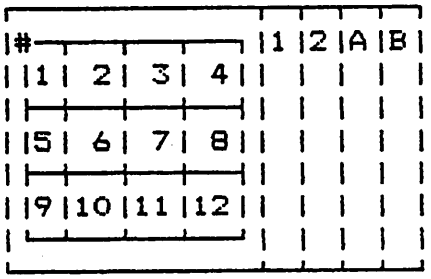
```
PROBLEME: COMPARER PLUSIEURS ALGORITHMES PERMETTANT
DE DETERMINER SI UN NOMBRE EST PREMIER

PO ERATO :NOMBRE
SI :NOMBRE = 1 [RETOURNE []]
SI PREMIER? :NOMBRE [RETOURNE METSDERNIER :NOMBRE ERATO (:NOMBRE - 1)]
RETOURNE ERATO (:NOMBRE - 1)
FIN

PO PREMIER? :NOMBRE
RETOURNE ET ( :NOMBRE = PPD :NOMBRE ) NON (:NOMBRE = 1)
FIN

PO PAIR? :NOMBRE
RETOURNE RESTE :NOMBRE 2 = 0
FIN

PO DIVISEUR? :DIVISEUR :DIVIDENDE
RETOURNE RESTE :DIVIDENDE :DIVISEUR = 0
FIN
PO PPD :NOMBRE
SI :NOMBRE = 2 [RETOURNE :NOMBRE]
SI PAIR? :NOMBRE [RETOURNE 2]
RETOURNE PPDSA 3 :NOMBRE
FIN

PO PPDSA :CANDIDAT :NOMBRE
SI DIVISEUR? :CANDIDAT :NOMBRE [RETOURNE :CANDIDAT]
SI :CANDIDAT > RC :NOMBRE [RETOURNE :NOMBRE]
RETOURNE PPDSA :CANDIDAT + 2 :NOMBRE
FIN

PO FERMATEST :NOMBRE
RELIE "A (2 + HASARD :NOMBRE - 2)
RETOURNE ( EXPOMOD :A :NOMBRE :NOMBRE ) = :A
FIN

PO EXPOMOD :BASE :EXPOSANT :MODULO
SI :EXPOSANT = 0 [RETOURNE 1]
SI PAIR? :EXPOSANT [RETOURNE RESTE PUIS (EXPOMOD :BASE (:EXPOSANT / 2) :MODULO
 2 :MODULO]
RETOURNE RESTE :BASE * (EXPOMOD :BASE (:EXPOSANT - 1) :MODULO) :MODULO
FIN
?ECRIS PREMIER? 281
VRAI
?
?ECRIS PREMIER? 329
FALSE
?
?ECRIS FERMATEST 281
VRAI
?
?ECRIS FERMATEST 329
FALSE
?
?ECRIS ERATO 300
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 21
 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
```

PROBLEME: UTILISER LES SOMMES DE RIEMANN POUR ESTIMER
L'AIRE SOUS LA COURBE.

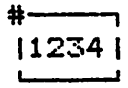
```
PO AIRE :BASE :HAUTEUR
RETOURNE PRODUIT :BASE :HAUTEUR
FIN

PO AIRE.INF :X :DX :F :N
SI :N = 0 [RETOURNE 0]
RETOURNE SOMME AIRE :DX IMAGE :X :F AIRE.INF :X + :DX :DX :F :N - 1
FIN

PO AIRE.MED :X :DX :F :N
SI :N = 0 [RETOURNE 0]
RETOURNE SOMME AIRE :DX IMAGE (2 * :X + :DX) / 2 :F AIRE.MED :X + :DX :DX :F :N
 - 1
FIN

PO AIRE.SUP :X :DX :F :N
SI :N = 0 [RETOURNE 0]
RETOURNE SOMME AIRE :DX IMAGE :X + :DX :F AIRE.SUP :X + :DX :DX :F :N - 1
FIN


?RELIE "F [(EXP :X) / :X]
?
?ECRIS IMAGE 1 :F
2.718281828
?
?ECRIS AIRE.INF .5 1 :F 10
2413.009979
?
?ECRIS AIRE.MED .5 1 :F 10
3755.949157
?
?ECRIS AIRE.SUP .5 1 :F 10
5868.331838
?
?ECRIS AIRE.MED .5 1 :F 100
 4.277839865E+0041
?
?ECRIS IMAGE .5 :F
3.297442542
?
?ECRIS IMAGE .01 :F
101.0050167
?
```

```
Q'Nial: Educational Version 3.06 (PCDOS 2.0) Sept 29, 1985
Copyright (c) Queen's University, 1983, 1984, 1985
clear workspace
profile loading

     N GETS COUNT 10
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬──┐
|1|2|3|4|5|6|7|8|9|10|
└─┴─┴─┴─┴─┴─┴─┴─┴─┴──┘

     (N*(N+1))/2
┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
|1.|3.|6.|10.|15.|21.|28.|36.|45.|55.|
└──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘

     PROD COUNT 5
120
     SUM (COUNT 20 POWER 2)/ 20
143.5
     NOMBRES GETS COUNT 20
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬──┬──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|
└─┴─┴─┴─┴─┴─┴─┴─┴─┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘


     SUM NOMBRES
210


     TALLY NOMBRES
20


     MOYENNE IS (SUM NOMBRES / TALLY NOMBRES)
     MOYENNE
10.5



est_pair IS OPERATION Nombres {
      FOR Item WITH Nombres DO
         IF Item mod 2 = 0 THEN
            write Item ;
         ENDIF ;
      ENDFOR }


     EST_PAIR NOMBRES
2
4
6
8
10
12
14
16
18
20
```

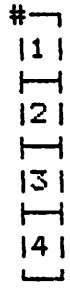
```
      TABLE 1 2 `A `B
```



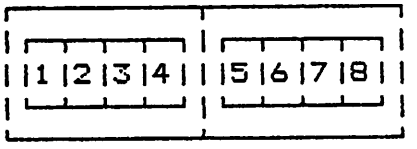

```
      POST 1234
```




```
      POST 1 2 3 4
```




```
      TABLE := (1 2 3 4) (5 6 7 8)
```




```
      2. = (40/20)
1
      2 = (40/20)
0
      34 * 45 * 8769
?Integer overflow
      34. * 45. * 8769.
1.34166e+007
```

CAL: The Mathematics Discovery Language

MODULE: TWO ENVIRONMENT: cal STATE: cal

MESSAGE: Press any key to continue.

Algebraic evaluation | Decimal evaluation | LISP evaluation | UNDO

```
? 1/2 + 1/3]

 (5 / 6)
 NIL


? 1/2 + 1/3]

 .8

? 1/2 +.1/3]

 .83333333333333333333

? (x - y/2)^4]

Error detected. Please check syntax and try again.

 NIL



? x y]

 (NIL NIL)


? (x - y/2)^4]

 X^4 -2 * X^3 *y + (3 / 2) * X^2 *Y^2 - (1 / 2) * X *Y^3 + (1 / 16) * Y^4

 NIL
```

WELCOME TO THE MATHCAD SELF-RUNNING DEMO!

In this demo, you'll see MathCAD in action.

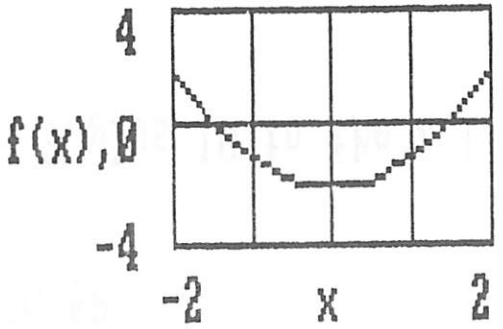
MathCAD handles equations . . .

$$x := \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} \qquad x = 1.414$$

. . . text . . .

The above equations shows the positive root of the quadratic formula.

. . . and graphs . . .



. . . all on the same screen.

(Press [F6] now to go on ...)

[F5] Go back
[F6] Continue
[F7] Clear & repeat
[F8] Back to menu

"MATHCAD EQUATIONS ARE EASY TO ENTER

For example, suppose you want to compute: 3 plus 10 to the 2.1 power divided by 4.5

You just type the same keys you might punch into a calculator . . . ⟶ 3 + 10 ^ 2.1 / 4.5 =

. . . and MathCAD displays the equation and computes the result. ⟶ $3 + \frac{10^{2.1}}{4.5} = 30.976$

You can edit an existing equation just as easily. WATCH THE SCREEN AS WE ADD THE SQUARE ROOT OF 200 TO THE EQUATION ABOVE.

(Press [F6] now and watch the screen ...)

[F5] Go back
[F6] Continue
[F7] Clear & repeat
[F8] Back to menu

```
muSIMP-83 4.06 (12/18/83)
MS-DOS Version
Copyright (C) 1982 The SOFT WAREHOUSE
Licensed by MICROSOFT Corp.
```

Problème soumis par P. Ponzo (Char, 1987)

On veut construire un mur de ciment autour d'une piscine circulaire de 30 pieds de rayon. La section transversale du mur doit avoir une base horizontale de 3 pieds, une hauteur de 5 pieds, et l'autre côté doit suivre une courbe polynomiale de degré 3. Déterminer l'equation de cette courbe de façon à minimiser le volume de ciment nécessaire pour construire ce mur.

```
% FAISONS COINCIDER LES DIMENSIONS PERPENDICULAIRES DU MUR %
% AVEC LES AXES X = 0 ET Y = 0 %
% POSONS LE POLYNOME DE DEGRE 3 PASSANT PAR (3,0) %
Y: (3 - x) * (a x^2 + b x + c);
@: (3 - x) (c + x b + x^2 a)

?


% REMPLACONS X PAR 0 DANS Y %
Y0: EVSUB (Y, x, 0);
@: 3 c

? % CALCULONS C EN POSANT Y0 = 5 %
C: SOLVE (Y0 == 5, c);
@: [c == 5/3]

? % REMPLACONS C PAR SA VALEUR DANS Y %
YC: EVSUB (Y, c, 5/3);
@: (3 - x) (5/3 + x b + x^2 a)

? % TROUVONS LA DERIVEE DY = dy/dx %
DY: EXPAND (DIF (YC, x));
@: -5/3 + 3 b + 6 x a - 2 x b - 3 x^2 a

? % REMPLACONS X PAR 0 DANS DY %
DY0: EVSUB (DY, x, 0);
@: -5/3 + 3 b

? % TROUVONS LA VALEUR DE B QUI ANNULE LA DERIVEE %
B: SOLVE (DY0 == 0, b);
@: [b == 5/9]

?
```

```
? % REMPLACONS B PAR SA VALEUR DANS Y ET DANS DY %
YB: EVSUB (YC, b, 5/9);
@: (3 - x) (5/3 + 5/9 x + x^2 a)

? DY: EVSUB (DY, b, 5/9);
@: -10/9 x + 6 x a - 3 x^2 a

? % TROUVONS LA DERIVEE DY EN X = 3 %
DY: EVSUB (DY, x, 3);
@: -10/3 - 9 a

? % POUR QUE DY SOIT <= 0 EN X = 3, IL FAUT QUE A SOIT >= -10/27 %
A: -10/27;
@: -10/27

? % TROUVONS LE VOLUME DU SOLIDE DE REVOLUTION %
V: DEFINT (2*#PI* (x+30) * YB, x, 0, 3);
@: (6225 #PI + 4293 #PI a)/10

? % TROUVONS LA DERIVEE DV = dv/da %
DV: FCTR (DIF (V, a));
@: 4293/10 #PI

?


? % COMME DV > 0, LE VOLUME MINIMUM DERA POUR A MINIMUM, A=-10/27 %
% CALCULONS LE VOLUME MINIMAL %
V: EVSUB (V, a, -10/27);
@: 927/2 #PI

? % TROUVONS LE POLYNOME DE DEGRE 3 SATISFAISANT LES CONTRAINTES %
Y: EVSUB (YB, a, -10/27);
@: (3 - x) (5/3 + 5/9 x - 10/27 x^2)

?
```

COURS 201-203-77 NOM__
SESSION 88-1 NOM__

---

21 $\int (x^3+x^2+x+1)\,dx$ $\qquad \frac{x^4}{4}+\frac{x^3}{3}+\frac{x^2}{2}+x+K$

Expressions MuMATH

```
? INT (X^3+X^2+X+1,X);
@: X + X^2/2 + X^3/3 + X^4/4
```

Résultat: identique X équivalent ___
faux ___ pas de résultat ___

Commentaires: Il donne la reponse dans l'ordre contraire de celle du livre.

.....

22 $\int \frac{dx}{2x^2+3}$ $\qquad \frac{1}{\sqrt{6}} \arctan\left(\frac{\sqrt{2}\,x}{\sqrt{3}}\right)+K$

Expressions MuMATH

```
? INT (1/(2*X^2+3),X);
@: ATAN (2 X/6^(1/2))/6^(1/2)
```

Résultat: identique ___ équivalent X
faux ___ pas de résultat ___

Commentaires: On a eu de la difficulté à trouver une même reponse, l'ordinateur utilise les paramètres des cadrans

.....

23 $\int \frac{dx}{x^2+4x}$ $\qquad \frac{1}{4}\ln\left|\frac{x}{x+4}\right|+K$

Expressions MuMATH

```
? INT (1/(X^2+4*X),X);
@: #I #PI/4 + LN (2 X/(8 + 2 X))/4
```

Résultat: identique ___ équivalent X
faux ___ pas de résultat ___

Commentaires: Il nous a donné des constantes et il n'a pas fait de simplifications

.....

REFERENCES

ARCOUET, M., AUBE, M., BRACKE, D., CLOUTIER, J.-F., Apprendre à utiliser l'environnement Smalltalk au Secondaire, C.S.R. Meilleur. Rapport de recherche.

BORK, A. (1981). Learning with Computers. Bedford, Mass.: Digital Press.

CHORAFAS, D.N. (1986). Fourth and Fifth Generation Programming Languages. Vol.1, Intergrated Software, Database Languages, and Expert Systems. New York: McGraw-Hill Book Company.

DAVIS, R. (1984). Learning Mathematics, A Cognitive Science Approach. Norwood, N.J.: Ablex Pub. Corp.

DYKSTRA, D. (1987). Microsoft QuickBASIC 2.0. Byte. Vol.12, no.2. Peterborough, N.H.: McGraw-Hill Inc.

EDWARDS, J.R., HARTWIG, G. (1985). Benchmarking the Clones. Byte. Vol.10, no.11. Perterborough, N.H.: McGraw-Hill Inc.

GIARD, J., HAGUEL, M.J. (1985). L'apprentissage du calcul différentiel et intégral par la programmation en Logo. Rapport de recherche. Collège de Sherbrooke.

GIARD, J. (1986). Style d'apprentissage, orientation et apprentissage des mathématiques chez les étudiants du collégial. Rapport de recherche. Collège de Sherbrooke.

GLASGOW, J.I., JENKINS, M.A., HENDREN, L.J. (1986). A Programming Language for Learning Environments. Comput. Intell. Vol.2.

GROGONO, P. (1988). Meaning and Process in Mathematics and Programming, For the Learning of Mathematics, sous presse.

HANCOCK, C. (1988). Context and Creation in the Learning of Computer Programming, For the Learning of Mathematics, Vol. 8, no. 1.

HAYES-ROTH, F. (1984). Knowledge Based Expert Systems. Computer. Vol.7, no.10.

KEARSLEY, G.P. (1987). Artificial Intelligence and Instruction; Applications and Methods. Addison-Wesley.

KOLB, D.A. (1976). Learning Style Inventory, Self-Scoring Test. Case Western Reserve University, Cleveland, Ohio.

MAYER, R.E., DYCK, J.L., VILBERG, W. (1986). Learning to Program and Learning to Think: What's the Connection? Communications of the ACM. Vol.29, no.7.

MUNRO, A. (1987). Choosing a Programming Langage. Macworld. Vol.4, no.10. San Francisco, CA: PCW Communications Inc.

NICKERSON, R.S. (1986). Using Computers: The Human Factors of Information Systems. Cambridge, Mass.: The MIT Press.

PAPERT, S. (1980). Mindstorms. New York: Basic Books.

PASK, G. (1976). Conversation Theory. Applications in Education and Epistemology. New York: Elsevier Publishing Co.

PEA, R.D. (1987). Cognitive Technologies for Mathematics Education in Schoenfeld, A.H. Cognitive Science and Mathematics Education. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

RAMSEY, H.R., ATWOOD, M.E., KIRSHBAUM, P.J. (1978). A Critically Annotated Bibliography of the Litterature of Human Factors in Computer Systems. Englewood, Colorado: Science Applications Inc.

SCHOENFELD, A. H. (1987). Cognitive Science and Mathematics Education, Hillsdale, N.J.: Lawrence Erlbaum Ass.

SLOMAN, A. (1984). Beginners Need Powerful Systems in Yazdani, M. New Horizons in Educationnal Computing. Chichester: Ellis Horwood Limited.

TAYLOR, C.F. (1984). The Master Handbook of High-Level Microcomputer Languages. Blue Ridge Summit, PA: TAB Books Inc.

VYGOTSKY, L.S. (1978). Mind in Society. Cambridge MA: Harvard University Press.

YAZDANI, M. (1984). New Horizons in Educationnal Computing. Chichester: Ellis Horwood Limited.

Toute personne intéressée par l'enseignement et l'apprentissage des mathématiques par la programmation, et désireuse d'exprimer des commentaires, d'adresser des questions ou de participer au développement de cette approche pédagogique, est priée de communiquer avec l'auteure de ce rapport à l'adresse indiquéeci-dessous, en fournissant les renseignements demandés. On peut commander des exemplaires de ce rapport en procédant de la même façon.

Nom _____ Occupation _____

Adresse _____ Niveau de l'intervention scolaire (s'il y a lieu)

_____ _____

_____ _____

Expérience pertinente (s'il y a lieu)
(logiciels et langages utilisés, matériel didactique, activités, cours,...)

Commentaires, questions... (Annexer des feuilles au besoin)

Adresser à: Jacqueline T. Giard
Collège de Sherbrooke,
475 rue Parc,
Sherbrooke, Qc
J1H 5M7

Tél. (819) 564-6319

ou: 703 rue Gariépy,
Sherbrooke, Qc
J1E 1J1

(819) 563-4342

Mathématiques et programmation

Analyse de besoins et inventaire de ressources au collégial

1532 - 0