QUELLE ALGORITHMIQUE POUR LE LYCEE ?

Jean-Pierre FERRIER
Irem de Lorraine

La réflexion ici présentée s'appuie sur une double expérience, dans le cadre de l'enseignement en série L. C'est d'abord la coordination de la fabrication d'une série de sujets pour le baccalauréat, pour laquelle il a fallu intégrer les contraintes spécifiques de l'examen. C'est ensuite l'organisation d'une formation pour les professeurs de mathématiques de l'Académie pilotée par l'Inspection pédagogique régionale [F2], occasion de se dégager de la lettre des programmes, tout en intégrant les contraintes liées aux conditions de l'enseignement considéré.

D'ailleurs, à la fin de la formation, l'idée de suggérer à l'institution d'adjoindre le thème aux programmes de mathématiques des autres filières, notamment scientifiques, a été explicitement formulée.

Commençons par un avertissement. De temps en temps le ton de notre texte pourra paraître irrespectueux envers l'institution. A-t-on cependant pensé aux collègues qui doivent sans cesse s'adapter à de nouvelles initiatives ? Les tester plus longuement n'aurait pas été un luxe. Nous adressons au passage une prière dans ce sens pour l'avenir.

Très vite le texte quittera le niveau des réflexions générales. Celui qui attendrait plus de hauteur pourra consulter l'exposé de Vladimir Latocha [L]. Ce dernier y décrit certaines vertus de la formation à l'algorithmique, comme celles du travail en groupe que facilitent les fonctions, vertus que nous ne pourrons qu'entrevoir. Il s'agira juste pour nous de rendre l'élève un peu plus alerte, un peu plus critique.

Même si nous ne nous inscrivons pas dans la ligne des programmes actuels, jusqu'à envisager d'autres options, nous nous situons dans un contexte extrêmement contraint, comme celui du baccalauréat L et des programmes qui y préparent, lequel a été le cadre de la formation mentionnée. Notre objectif sera en particulier très limité : ne pas se disperser, ne pas déflorer la matière. Malgré ces difficultés, nous essayons de faire une proposition qui ait une cohérence d'ensemble.

Cela étant, nous chercherons aussi à coller d'aussi près que possible au cours de mathématiques. Cela nous amènera à réfléchir sur le programme disciplinaire lui-même. Cela vaudra pour le fond comme pour la forme. Nous essaierons d'intégrer l'écriture algorithmique dans le même moule que l'écriture mathématique à ce niveau.

Par conséquent, une fois écartées certaines grandes options, nous n'envisagerons plus, question par question, un éventail de possibilités entre lesquelles chacun pourrait choisir. Nous serons déjà heureux d'en disposer d'une, même si elle ne nous satisfait pas pleinement. De même ne faudra-t-il pas s'étonner que le ton soit souvent celui d'une prescription, dans le genre : « je vous propose un traitement léger ; il n'y aura pas d'antibiotiques ». Si certains renoncements ne font pas non plus plaisir à l'auteur lui-même, il les assumera.

Problématique et objectifs.

Il y a quelques décennies déjà, un enseignement d'informatique a été mis en place, indépendant de celui des disciplines classiques. Il est vite apparu que cet enseignement n'avait aucun contenu scientifique, parce qu'il était réduit à l'étude de l'impact social de cette science. Heureusement c'est une autre option qui est prise aujourd'hui, celle d'enseigner un peu d'algorithmique à l'intérieur même des contours du programme de mathématiques.

Avec les aménagements qu'apporte la version définitive du programme de seconde par rapport au projet soumis à consultation, il ne s'agit plus d'enseigner l'algorithmique comme un chapitre parmi d'autres, mais d'en faire un thème transversal. C'est donc une invitation à opérer une réelle fusion entre les deux disciplines, ce que nous ne pouvons qu'approuver.

Il y a malheureusement un écueil. Le programme actuel reste sous-tendu par le choix obsessionnel d'employer « l'outil informatique », même si la chose n'est plus explicitée dans la dernière mouture [1]. L'ordre fondamental des priorités est d'abord les machines et ensuite les mathématiques. Dans ces conditions la place de l'algorithmique est limitée à une interface entre les deux. Cela explique certaines confusions que l'on rencontrera.

La bonne problématique n'est pas de savoir quels éléments d'algorithmique il convient de plaquer sur un enseignement de mathématiques déjà défini. Ce serait plutôt de voir comment redéfinir ce dernier pour

1 « Depuis une dizaine d'années, le développement de l'usage de logiciels (calculatrice ou ordinateur) a permis de développer chez l'élève la capacité d'expérimenter, suscitant le sens de l'observation ou faisant naître de nouvelles questions relatives à la nature de la démonstration ». Ce passage, en lignes 2 à 5 du document ressource « algorithmique » [D2], renvoie au seul outil, détruisant la perspective de s'appuyer sur la « pensée algorithmique» évoquée en ligne 2. C'est bien d'une obsession qu'il s'agit. Par exemple, que la nature de la démonstration mathématique ait été affectée par l'outil, probablement au travers du triptyque infernal observer/conjecturer/démontrer, laisse absolument rêveur.

profiter de ce que peut apporter la science informatique, au niveau même des concepts, pour les clarifier et les illustrer. Cela n'a pas été fait. Au moins, en série L, l'algorithmique accompagnait un programme comportant par ailleurs de l'arithmétique et le raisonnement par récurrence. La première comprend naturellement une part importante d'algorithmes. Le second connaît une variante très intéressante du côté de l'algorithmique. Il y avait donc une matière à exploiter, même si les documents d'accompagnement pour cette série n'ont pas su en profiter.

Voici une idée qu'on pourrait creuser pour accompagner un enseignement théorique sur les fonctions, qui m'a été suggérée par un collègue informaticien. Il s'agirait de s'appuyer sur le Lambda calcul de Church pour l'initiation à l'algorithmique. Ce sont les fonctions elles-mêmes qui sont au cœur de ce calcul. Les notations ne sont pas exactement celles de la théorie des ensembles, mais elles sont voisines et compatibles. De toute façon le choix de $f(x)$ fait en mathématiques pour noter la valeur en x de la fonction f, choix qui vient de ce que $f(x)$ était au début une notation abrégée pour « fonction de x », n'est pas idéal ; on aurait aussi bien pu choisir l'ordre $x\ f$.

Aujourd'hui cette option n'est pas envisageable, compte tenu du manque de temps accordé et de l'importance de la formation des enseignants qu'il faudrait mettre en place. On se contentera donc de l'algorithmique impérative et de la notion de variable informatique sur laquelle elle s'appuie. Cela implique que les fonctions doivent être traitées, en mathématiques, d'une façon compatible, essentiellement comme des variables dépendant d'autres[2]. Or, sur l'ensemble du lycée et même du collège, les programmes privilégient toujours la vision ensembliste[3], laquelle est évidemment dénaturée[4] et engendre un résultat peu convaincant[5]. Cela étant, quelques éclairs les traversent de temps à autre. On parle ainsi à l'occasion d'une grandeur qui évolue en fonction d'une autre[6], alors que ce point de vue devrait être assumé complètement.

Formalisation.

Compte tenu du temps limité qui peut être consacré à l'algorithmique au lycée, notamment en classe de Seconde, mieux vaut ne pas avoir au départ d'ambition exagérée. Cela est vite apparu dans la formation, plus encore pour le bac. On traitera convenablement un petit nombre de points plutôt que de toucher à tout et de rester à la surface des choses.

Réduire les ambitions. On ne pourra pas envisager d'enseigner tout un corpus exploitable tel quel dans un enseignement futur d'informatique. On sera obligé de prendre un point de vue assez naïf. Maintenant il faudra bien sûr se garder de donner des idées

2 C'est la masse m du fil de cuivre qui dépend de sa longueur l, la pression p qui dépend du volume v ou de la température t, la gravité g qui dépend de l'altitude h ; c'est aussi la surface s du triangle équilatéral qui dépend du côté a ; c'est bien sûr $y = ax + b$, mais la dépendance peut exister indépendamment d'une « formule » qui l'exprime ; en revanche ce n'est pas la donnée abstraite de la famille (f(x)), donc d'une « courbe », d'un « nuage de points » ou d'un « tableau de valeurs » qui la représentent en tant que « processus » (document ressource « fonctions », p 4-5 [D3]).

3 Cela se voit sur les notations ; on demande de ne pas confondre la fonction f et sa valeur $f(x)$ en x, où x est aussi une valeur ; on exclut la possibilité pour x d'être une variable et pour f de désigner à la fois la fonction et sa valeur en x.

4 Comme tout ce qui relève d'une transposition didactique et comme le dit Yves Chevallard ; la théorie des ensembles est une théorie, avec ses axiomes, pas un vague langage.

5 Notamment en privant les élèves de la règle de la chaîne $dz/dx = dz/dy.dy/dx$ pour dériver.

6 Une aire qui varie en fonction d'une longueur (document ressource « fonctions », p 2 [D3]).

fausses, comme le libellé du programme et les documents qui l'accompagnent risquent d'y conduire [7]. Le mieux sera donc d'assumer avant tout la modestie du propos. Le pire serait de déflorer la matière à laquelle on prétend initier les élèves.

Par exemple il serait sage de glisser sur la question de la déclaration des variables. Certes discuter sur les types, en liaison avec leur traduction ensembliste, serait intéressant, mais ce serait trop demander. Par conséquent une liste des variables utilisées pourra être établie en marge d'un algorithme, un peu comme la liste des ingrédients figure en marge d'une recette de cuisine. C'est d'ailleurs plus ou moins l'option prise dans le document ressource.

Quelles variables considérer ? Des variables entières à coup sûr. Des variables réelles peut-être. Des tables à l'extrême rigueur. En revanche les variables booléennes sont à introduire, en relation avec le travail sur la logique, même si cela n'a pas été envisagé dans le programme. En matière de structures de données, on s'en tient au strict minimum.

Par ailleurs, pour la même raison, il n'y aura pas de procédures, donc de fonctions [8]. L'algorithmique impérative n'est pas le meilleur cadre pour cela. A fortiori la récursivité est-elle exclue. C'est déjà avec beaucoup de réticence que l'on envisagera de placer une itération à l'intérieur d'une autre. On préfèrera se ramener à une itération simple, ce qui est souvent un bon exercice.

Dégager les entrées/sorties. Un des défauts constatés dans les exemples fournis dans document d'accompagnement de la série L, qu'on retrouve dans le document ressource pour la classe de seconde « algorithmique » [D2], est le mélange entre déclarations, entrées et initialisations [9]. Nous avons déjà parlé de la question des déclarations. On ne devrait pas en trouver, mêlées au reste, dans les algorithmes.

Les entrées et sorties sont la communication d'un algorithme avec l'extérieur. Ce sont ses yeux ou oreilles et ses organes locuteurs. Mieux vaut ne pas les inclure dans le traitement proprement dit, alors qu'il convient absolument de le faire pour les diverses initialisations. C'est ce que la première page du document ressource « algorithmique » [D2] semble vouloir, en demandant d' « identifier les données d'entrée, de sortie, le traitement », mais qui est contredit quelques pages plus loin [10] : la première étape est une « préparation du traitement » dont le contenu est flou ; le « traitement des données » mêle la lecture, l'affectation et l'écriture.

Une raison supplémentaire de bien mettre à part les entrées et sorties est qu'elles ne font pas partie des langages de programmation professionnels, comme le langage C ou même les implémentations professionnelles du langage Pascal. Elles sont en effet dans le système d'exploitation. Dans ce fait se trouve d'ailleurs la raison pour laquelle ces langages sont difficiles à importer dans la classe.

7 Par exemple, la liste de compétences donnée en page 3 du document ressource « algorithmique » [D2] peut laisser croire que l'objet principal de l'algorithmique est de comprendre « un algorithme préexistant », de le « modifier », d'en « identifier les éléments » ; même la conception d'un algorithme n'est évoquée que par une « mise au point », consistant toujours à « identifier » les boucles, les tests etc.

8 Dans un langage impératif, une fonction est une procédure particulière, avec un paramètre de sortie ; cela n'empêchera pas d'utiliser des fonctions mathématiques dans les programmes, de concevoir un programme entier fabriquant une fonction, ou de consacrer un programme à l'étude d'une fonction.

9 Dans l'algorithme des pages 21 à 30, on trouve une partie déclarative dont le statut est peu clair et le contenu incomplet ; les variables ne sont pas initialisées ; voir les extraits **(b)** et **(d)** en annexe.

10 Page 6 exactement.

Dans l'idéal de l'apprentissage, les entrées et sorties sont groupées dans un bloc en début et un autre en fin. Cela peut obliger à différer l'affichage des résultats en les plaçant préalablement dans une liste. Nul n'est parfait.

Insister sur les trois traitements. Il y a plusieurs raisons à l'impératif de réduction de l'algorithmique à quelques traitements types. C'est d'abord le souci de limiter les ambitions. C'est encore celui de faciliter la traduction en programmes. C'est plus fondamentalement un principe fondateur de la Science, qui est une démarche permanente de simplification. On pourrait d'ailleurs tirer parti de cette réduction pour engager la preuve d'un algorithme, s'il le fallait.

On répartira donc les traitements dans les trois types : séquentiel, conditionnel et itératif. Pour le dernier on n'envisagera qu'une seule forme, l'itération « tant que... ». Là encore le double souci indiqué s'applique. Le choix de cette variante n'est pas fortuit. On peut y ramener simplement les itérations « Pour... » et « ...jusqu'à... » alors que l'opération inverse est moins simple. Surtout le fait de poser la condition avant d'opérer le traitement rapproche de la pratique mathématique et rend plus naturelle l'initialisation. Autant que possible, mieux vaudrait ne montrer aux élèves que des algorithmes structurés [11].

Le programme pour la classe de seconde et le document ressource « algorithmique » qui l'accompagne ont fait des choix incompatibles entre eux et avec le nôtre. Le premier se limite au cas d'un nombre fixé d'itérations. Le second envisage les trois « structures répétitives », mettant l'accent maladroitement sur les instructions de « contrôle » [12]. Surtout il s'appuie un peu trop sur un langage, SCRATCH en l'occurrence, pour exclure dans les exemples de traduction la seule forme que nous ayons retenue.

Algorithmique et programmation, traduction en programmes.

S'il n'existait pas de machines et de programmes exécutables, l'informatique n'aurait certainement pas l'impact qu'on lui connaît aujourd'hui. Par ailleurs l'élève à qui l'on a demandé d'écrire un algorithme et qui s'est acquitté de sa tâche aimerait bien voir sa production « tourner ». Aussi est-il judicieux de penser à cette récompense que constitue l'exécution d'un vrai programme. Nous y reviendrons.

Cela étant, il ne faudrait pas confondre algorithmique et programmation. C'est, hélas, ce que l'on trouvait dans le programme de la série L et que l'on retrouve encore dans le programme de Seconde. Le document ressource est très loin de lever toutes les ambiguïtés. L'insistance avec laquelle il est fait appel à des exemples de programmation dans des langages très variés en témoigne.

Les programmes. Prendre l'algorithmique par le bout de la programmation serait une erreur. D'abord la programmation est le plus souvent dépendante de la machine. Ensuite il ne suffit pas toujours d'apprendre quelques mots-clés d'un langage informatique ; il faut

11 A ce sujet on aurait pu trouver mieux que l'algorithme de la page 23 et le programme SCILAB de la page 24, où l'on sort au milieu d'une itération, même si les rédacteurs assument leur choix « pragmatique » ; voir l'extrait **(c)** en annexe.

12 Nous parlons ici de maladresse parce que ce n'est pas dans l'esprit de la programmation structurée. Les tout petits algorithmes qu'on fera écrire, souvent limités à une seule itération, n'ont pas de raison de s'affranchir de la structuration, qui est très importante dans la vérification des programmes.

éventuellement s'habituer à ce qu'on appelle un environnement de programmation.

Il est possible de réduire ce dernier obstacle. Là réside le succès rencontré jadis par le langage Basic ou aujourd'hui par les langages machine spécialisés des calculatrices. Il faudra malgré tout prévoir un investissement non négligeable en temps, dont le côté formateur sera parallèlement très limité. Dans le même ordre d'idées, qui aurait l'idée de faire apprendre à l'école le maniement d'un téléphone mobile, la programmation d'un magnétoscope ou d'une machine à laver ? Il est vrai que l'école sert parfois de relais à Microsoft en choisissant de mettre l'apprentissage de Word ou d'Excel dans ses programmes.

Il est également possible de réduire la dépendance par rapport à la machine. Il suffit de bénéficier d'un bon langage algorithmique implanté dans les machines les plus courantes. C'était le pari du langage Pascal de Niklaus Wirth, conçu au départ pour l'enseignement mais doté de compilateurs performants et à la base des premières applications pour le Macintosh et du logiciel typographique TeX de Donald Knuth. Aujourd'hui l'industrie utilise plutôt le langage C. Bien qu'il soit décrit comme de bas niveau, il est déjà très supérieur aux langages machine spécialisés et finalement assez proche du Pascal. Maintenant l'obstacle pour l'un et l'autre est la difficulté d'accès. Nous verrons cependant que des possibilités existent.

En tout cas inventer un nouveau langage de programmation serait une aberration. C'est ce qui, étrangement, a été fait pour la série L, avec un résultat bien peu probant [13].

Le choix retenu aura dû concilier au mieux les deux impératifs suivants, lesquels sont plus ou moins contradictoires :

1) la lisibilité,

2) la facilité de transcription en programme exécutable.

L'exigence 1) élimine déjà un certain nombre de propositions envisagées dans le document d'accompagnement pour la seconde.

Voici des extraits de programmes écrits pour :

TI	**Casio**
Input N	? -> N
0 -> X	0 -> X
0 -> C	0 -> C
Y -> D	Y1 -> C
FOR(K,1,N)	Y1 -> D
K^2 -> X	For 1-> K To N
If Y1 > D	K^2 -> X
Then	If Y1>D
Y1 -> D	Then Y1 -> D
X -> C	Ifend
End	Next
End	

C'est simplement l'horreur !

Et que dire de Linotte [14] où l'on lit un certain nombre de joyeusetés comme « xA est un nombre vide », de Scratch qui demande « à A attribuer 5 » quand ce n'est pas « à X attribuer position x » ?

La langue naturelle. Le choix qui s'est très vite imposé pour la formation en série L a été de privilégier l'emploi de la langue naturelle.

13 Etant donné le peu de place qu'occupent les mathématiques de la série L dans les préoccupations des scientifiques en général, et le peu d'impact qu'un tel investissement peut avoir, c'est effectivement étrange.

14 Voir l'extrait **(a)** en annexe.

Pour une mystérieuse raison, on tient toujours à parler de langage, alors que le terme est réservé aux animaux et aux machines. Rappelons que la langue est dite naturelle quand elle est indigène. Cependant passer d'une langue à une autre est bien plus facile que traduire les unes dans les autres certaines conventions locales du symbolisme mathématique inventé par certains pédagogues zélés.

Le mérite de la langue naturelle est sa lisibilité. C'est du moins vrai quand on a le souci d'en préserver les qualités. Par exemple une des exigences formulées dans la formation citée plus haut était d'atteindre un niveau de qualité linguistique au moins égal à celui des recettes de cuisine. Notamment on fera des phrases comportant des verbes [15], on n'oubliera pas la ponctuation et l'on y soumettra l'usage des majuscules [16]. On utilisera aussi l'infinitif pour les instructions, comme c'est l'usage dans les exercices de mathématiques à un certain niveau et comme c'est aussi l'usage dans une bonne recette [17].

L'inconvénient majeur de la langue naturelle est l'absence de parenthèses explicites. Cependant on peut jouer sur la ponctuation [18]. Ce n'est pas très glorieux, mais avec un peu de soin et pour des algorithmes simples, comme ce sera toujours le cas au lycée, il est possible d'écrire des algorithmes très faciles à traduire en programmes. La formation l'a montré.

Cela étant, il ne faut pas se priver du symbolisme mathématique. L'insertion de ce dernier dans la grammaire usuelle a été parfaitement codifiée : les termes sont des groupes nominaux et les relations des propositions. C'est ainsi qu'après réflexion la meilleure manière d'écrire une affectation est encore d'utiliser le signe d'égalité.

On écrira donc :

$$y = 2x + 3$$

de préférence à : « affecter à y la valeur $2x + 3$ » ou quelque autre lourdeur équivalente. Certains objectent que le signe = en mathématiques a le plus souvent un tout autre sens. De fait, il y est très polysémique et c'est, comme toujours, le contexte qui tranche.

On saura que, pour ce qui ne peut être qu'une affectation, il aura le sens qu'il convient : placer dans la variable y la valeur qui est présentement dans la variable x. C'est le choix du langage C. Bien sûr on peut légitimement préférer le := du langage Pascal, symbolisme qui est d'ailleurs assez répandu en mathématiques quand on expose au tableau.

Pour l'égalité dans une condition, on emploiera aussi le signe = qui prendra alors un autre sens, son sens mathématique le plus courant. Le langage C utilise = = pour éviter les confusions. Ce n'est pas une raison de faire de même dans la version en langue naturelle.

L'exécution. Comme on l'a dit, il est important de s'occuper de l'exécution des algo-

15 Le document ressource « algorithmique » [D2] nous gratifie de phrases sans verbe comme « f, la fonction à étudier » ; voir les extraits **(b)** et **(d)** en annexe.

16 Ce n'est malheureusement pas ce qu'on trouve dans les exemples « en langage naturel » du document ressource « algorithmique » [D2] : il n'y a aucune ponctuation et beaucoup de majuscules ; voir notamment les extraits **(a)**, **(b)**, **(c)**, **(d)** en annexe.

17 A l'exclusion d'un indicatif ambigu ou d'une seconde personne de l'impératif bêtifiante, comme on en trouve dans le document ressource « algorithmique » [D2] ; voir les extraits **(a)**, **(b)**, **(c)**, **(d)** en annexe.

18 Un langage comme SCRATCH n'a pas ces qualités.

rithmes. En série L, on a mis l'accent sur l'exécution à la main, en pensant à l'épreuve du baccalauréat. En conséquence on devait pratiquement se limiter à des algorithmes opérant sur des nombres entiers. D'ailleurs la première exigence était de comprendre ce que faisait un algorithme donné en l'exécutant pas à pas.

En classe, on peut être un peu plus ambitieux, mais il faut éviter de gaspiller son temps en cherchant à faire apprendre à tous les élèves un jeu d'instructions spécialisées. Par conséquent on peut mettre en parallèle deux stratégies.

1) Laisser les élèves disposant de calculatrices programmables le soin de les programmer eux-mêmes, en dehors de la classe, pour présenter le résultat à tous, en classe. Il suffit de donner les quelques règles de traduction des traitements de base, présentées sous la forme d'un petit tableau, mais seulement à ces élèves.

2) Montrer, en classe, l'exécution d'un programme écrit par le professeur. On en trouvera un exemple dans la formation mentionnée. Le programme ci-dessous, qui calcule un nombre de diviseurs, réside dans une page html où l'on trouve un formulaire pour les entrées et sorties. Il lui est associé une procédure Javascript dans l'entête de la page. Dans cette dernière, le professeur aura écrit à l'avance les instructions d'entrée et de sortie (à ne pas regarder) et il se contentera d'écrire la partie en gras en la commentant ..et en évitant, autant que se peut, les ésotérismes. Ce ne peut être envisagé qu'à titre tout à fait exceptionnel, pour faire toucher du doigt les écueils de la programmation.

Dans la classe

Le travail en classe doit essentiellement concerner la conception d'un algorithme. Il ne semble pas que les rédacteurs du programme y aient même pensé. Il fal-

```
                    function divi()
                    {
(initialisations)   var n=0; var p=0; var k=1; var list = 'diviseurs :  ';

(entrée)            n = document.exec.var1.value;

                    while (k <= n)
                          {
                          if (n % k == 0) {p = p + 1; list = list + k + ', '};
                          k = k + 1
                          };
                    list = list + '  nombre de diviseurs =  ' + p;

(sortie)            document.exec.var3.value = list;
                    };
```

lait s'y attendre. En mathématiques on ne démontre pas, on contemple ; en informatique on ne conçoit pas, on regarde. Ce n'est pas « identifier les boucles, les tests, des opérations d'écriture, d'affichage » qu'on attend du programmeur ; son travail est bien d'écrire le programme.

Si un algorithme est bien conçu, il sera déjà pratiquement démontré. Pour autant on ne soulèvera pas devant les élèves la question de la preuve d'un algorithme, laquelle ne consiste pas à « valider un programme simple » par quelques essais [19].

Dans ce qui suit, nous cherchons à montrer que l'écriture d'un algorithme peut servir, en même temps, de support pour la pensée. On a mis en gras ce qu'on écrirait. Cela dit l'élaboration devrait être plus progressive et plus détaillée que celle que ce que montrons.

Voici un exemple sans la moindre originalité parmi ceux qui ont été proposés pour la série L et repris dans la formation. Il s'agit tout bêtement de calculer le PGCD de a et b où a, b sont des nombres entiers tels que $0 \leq b < a$.

L'idée est la suivante. Supposant b non nul, le PGCD ne change pas si l'on remplace a et b par b et le reste de a dans la division par b. La transformation à itérer est ainsi :

$$\boldsymbol{r = a \bmod b\ ;}$$

$$\boldsymbol{a = b\ ;\ b = r}.$$

Ici l'on doit prendre soin de l'ordre dans lequel on modifie a et b.

Quand faut-il s'arrêter ? Lorsque le résultat est acquis. Pour cela il faut envisager la situation la plus simple. Ce peut être celle où b divise a, mais c'est encore mieux celle où b est nul ; le PGCD est alors a. Et dans le cas contraire la transformation est valide. Noter qu'en mathématiques on aime bien initialiser les récurrences à leur tout début.

D'où l'algorithme :

Tant que $b > 0$ faire :

$$\boldsymbol{r = a \bmod b\ ;}$$

$$\boldsymbol{a = b\ ;\ b = r.}$$

Afficher a.

Bien sûr on peut l'améliorer ; pour ne pas détruire les valeurs a et b par exemple.

L'arithmétique offre un bon cadre à l'algorithmique. A l'inverse le monde numérique est plein de traquenards. Prenons, dans le docu-

19 Le point avait été abordé lors de la formation académique. La preuve du programme de calcul de PGCD qui suit n'est pas difficile. D'abord l'itération se termine : la suite des valeurs prises par b est strictement décroissante. Ensuite elle fournit bien le résultat cherché. La construction le donne, mais on veut le retrouver a posteriori, en remarquant que le PGCD de a et b est ce qu'on appelle un *invariant de boucle* : pour chaque itération, sa valeur à la fin égale celle en début. Or elle vaut, en entrée, le PGCD cherché et, en sortie a.

Cela dit, on peut se servir de l'algorithme pour *démontrer* qu'il existe un diviseur commun dont tous les autres sont des diviseurs. Pour cela on remarque que l'ensemble des diviseurs communs à a et b est aussi un invariant de boucle. En sortie, on obtient les diviseurs d'un nombre qui sera le PGCD d.

On peut aussi bien établir le théorème de Bézout. En effet l'idéal $a\mathbf{Z} + b\mathbf{Z}$ est encore un invariant de boucle et sa valeur en sortie est l'idéal $d\mathbf{Z}$.

Après que notre article ait été soumis, le numéro 78 de la revue a publié un article d'Henri Lombardi qui montre que derrière la plupart des démonstrations classiques se cachent des algorithmiques. Notre propos était plus modeste. Nous nous contentions de montrer que la démarche algorithmique pouvait fournir un moyen puissant de démonstration, sans chercher à l'associer étroitement à la démarche démonstrative usuelle.

ment ressource « algorithmique » [D2], l'exemple du test de la monotonie d'une fonction [20]. Nous n'en discuterons pas la pertinence.

Sachant que les valeurs sont discrétisées et les calculs approchés, c'est la signification même de la monotonie qui est altérée. Plutôt que de conclure stupidement que « la fonction semble monotone », disons qu'elle l'est à telle précision p sur les valeurs de la variable. Ensuite l'étude du sens de l'accroissement doit être évalué en laissant la place à un terme d'erreur e. Enfin la précision choisie doit être plus grande que l'erreur pour que le test soit significatif. Tout cela n'est pas simple et nous n'insisterons pas davantage là-dessus.

Sachant que $f(b) - f(a)$ a un sens s au départ, on teste les accroissements tant qu'on n'a pas rencontré un sens incompatible. A partir de valeurs de x et de $y = f(x)$, on fait ceci :

$x = x + p$; $z = f(x)$;

$d = z - y$; $y = z$;

« tester la compatibilité de d avec s ».

Au passage on aura évité de calculer deux fois la même valeur de f.

On fera l'opération :

Tant que x < b et que « la compatibilité est vérifiée »

et on l'initialisera par :

$x = a$; $y = f(a)$; « la compatibilité est vérifiée ».

Bien sûr la compatibilité sera une variable booléenne ; c'est une occasion d'en parler.

L'algorithme du même document recherchant le maximum d'une fonction d'abord croissante, puis décroissante, aurait pu être intéressant [21]. Un pas p étant donné, on cherche la première valeur de la variable x faisant apparaître une descente, donc située au-delà du sommet ; on repart alors en sens contraire après avoir divisé le pas par 10.

La première chose est de se donner une précision pr, pour effectuer l'opération

Tant que $|p| > pr$
…

Cette dernière concernera un pas p (positif ou négatif) et une valeur x de la variable; éventuellement on aura besoin de la valeur y de la fonction en x pour éviter un nouveau calcul. Si la descente est trouvée, on repart d'où l'on est en sens inverse avec un nouveau pas. Sinon on passe à la valeur suivante de x. On itèrera donc ce qui suit.

si « la descente est trouvée »
faire $p = -p/10$

sinon faire :

$x = x + p$; $z = f(x)$;

si z < y alors « la descente est trouvée » ;

$y = z$.

20 Pour un algorithme faisant aussi peu de choses que celui de la page 23, il est désolant de voir l'apparente technicité de la traduction SCILAB de la page 24 ; on a gonflé les détails en passant à côté de l'essentiel ; voir l'extrait **(c)** en annexe.

21 Tel qu'il est présenté, l'algorithme est certainement « ambitieux ». Il faudrait déjà en donner la preuve. Cependant sa présentation est maladroite. On impose une précision liée à l'intervalle, sans doute au-delà du raisonnable. On place une itération dans une autre. On fait calculer deux fois la même valeur. On entre et on termine salement. L'effort d'analyse en langue naturelle est malgré tout louable. En revanche la traduction PYTHON, qui n'est d'ailleurs pas fidèle, demande une technicité très accessoire. Voir l'extrait **(d)** en annexe.

Enfin :

***p* = *b* – *a* ; *x* = *b* ; *y* = f(*b*) ;**
« **la descente est trouvée** » **.**

en sera l'initialisation. On notera que l'on sort avec la bonne valeur de p. Le maximum est alors $x - p$ à plus ou moins $|p|$ près.

Conclusion

Nous avons vu que mettre en place un enseignement d'algorithmique posait surtout beaucoup de problèmes. S'il fallait retenir une priorité, ce serait, une fois les principes posés, de laisser les élèves être auteurs d'algorithmes. C'est comme avec la stratégie développée par Frédéric Pham, où l'on « dessine » avant de « voir » et d'« interpréter ».

C'est comme l'écriture qu'il faut pratiquer de pair avec la lecture. C'est comme pour l'étude des fonctions, pour laquelle les élèves doivent produire des courbes représentatives, sans machine évidemment, avant d'en considérer.

ANNEXE

Extraits commentés d'algorithmes proposés par l'institution

(*tirés du document ressource* « algorithmique » [D3])

Extrait (a)
page 14 (quatrième sommet)

Variables

x_A …

Entrées

Saisir x_A …

Traitement

x_I **prend la valeur** $(x_A + x_C)/2$

…

Sortie

Afficher x_D, y_D

La structure générale de cet algorithme est peu critiquable. On peut juste se demander l'intérêt de la liste des variables, dans la mesure où elle n'est pas commentée. Cependant un premier écueil vient du mélange des modes. Surtout la traduction LINOTTE vaut le détour.

```
Rôles :
    xA est un nombre vide
    …
    réponse est un texte valant «Le point D a pour coordonnées : («
Actions :
    tu affiches «xA=»»
    tu demandes xA
    …
    tu ajoutes (xA+xC)/2 dans xI
    …
    tu ajoutes xD dans réponse
    …
```

On utilise l'indicatif à la seconde personne du singulier. On touche, avec ce langage, les limitations de la stratégie consistant à vouloir faire parler la machine comme un être humain. Cela produit des formules incompatibles avec les mathématiques : qu'est-ce qu'un « nombre vide »? Et surtout un niveau zéro de langue : « tu ajoutes dans réponse ».

Extrait (b)
page 21 (recherche d'extremum).

Variables

a, b les bornes de l'intervalle d'étude
f, la fonction à étudier
N, le nombre d'intervalles
x, la valeur « courante »
y, la valeur correspondante de f(x)

Initialisation

min **prend la valeur** f(a)
…
pas **prend la valeur** (b-a)/N
…

On ne comprend pas le sens de la déclaration de a, b, f et N ; si ce sont bien des variables, il faudra qu'elles prennent une valeur. Quand sera-ce le cas ? Il n'y a pas de saisie prévue. En fait ce seraient plutôt des paramètres d'entrée pour une procédure. Cependant il n'a jamais été envisagé de procédure dans l'algorithmique pour la seconde.

En revanche, x et y sont bien des variables, des variables locales de la procédure. Mais que dire des variables min et pas qui apparaissent dans l'initialisation ? Pourquoi ne les a-t-on pas déclarées. Ici pas serait un paramètre d'entrée et min un paramètre de sortie.

Cet exemple a du être obtenu par copier coller, sans que l'effort d'adaptation ait été fait.

Extrait (c)
page 24 (test de monotonie).

…
Pour k **variant de 1 à** N
Si (f(x+pas) –f(x) **n'est pas de même signe que** sens **) alors**
Affiche « la fonction n'est pas monotone »
Affiche *x* et *x*+pas
Fin du programme
x **prend la valeur** *x*+pas
Affiche « La fonction semble monotone ».

On n'a pas reproduit ici le début de l'algorithme, lequel mériterait les mêmes remarques que celles qui ont été formulées pour l'extrait précédent. D'autres commentaires s'imposent :

On remarque l'absence presque totale de ponctuation et une utilisation des majuscules non conformes à la grammaire commune. L'algorithme ne peut être lu de façon fluide, comme le serait une recette bien rédigée. Par ailleurs on y mélange l'indicatif et l'impératif. Dans un langage impératif, on donne des ordres à la machine, comme à l'utilisateur d'une recette.

Surtout on a là un exemple de programmation non structurée. Le programme s'achève alors que l'itération est en cours. Cela veut dire que la condition de contrôle (ici $k \leq N$) ne la contrôle pas vraiment. Certes le programme ne prévoit qu'un nombre fixe d'itérations ; cette exigence n'est cependant pas respectée ici dans l'esprit.

La traduction SCILAB est donnée, qui comprend ceci :

```
iIf sens==0
    disp(«Les images f(a) et f(b) sont égales : le programme ne fonc-
    tionnera pas»);
    end ;
disp(« f(a) = «+string(f(a))+» et f(b) = «+string(f(b));
for k=1:N
    if ((f(x+pas) – f(x))*sens<0)
           disp («La fonction n'est pas monotone»);
           return;
    end;
    x=x+pas;
end;
disp(«La fonction semble monotone»);
```

Sa syntaxe est semblable à celle du langage C. Le principal problème est qu'il n'est pas conforme à l'algorithme ci-dessus. L'affichage est différent et il y a surtout un test d'entrée qui n'avait pas été prévu. Sinon il est doublement non structuré. Or cela n'avait rien d'obligatoire.

Voici une version, en langage C, évitant cet écueil, à laquelle il resterait juste à afficher la valeur de la variable booléenne introduite.

D'abord sous une forme compatible avec les notations précédentes :

```
x = a; monotone = true;
while ((x < b) et compatible)
        {monotone = ((f(x+pas)-f(x))*sens >= 0);
        x = x+pas};
```

Ensuite une version, plus exigeante, tirée de l'analyse que nous avons faite :

```
x = a; y = f(a); compatible = true;
while ((x < b) et compatible)
        {x = x+p; z = f(x);
        d = z-y; y = z;
        compatible = (d*sens >= 0)};
```

Extrait (d)
page 27 (recherche d'extremum).

Variables

f, la fonction
a,b les bornes de l'intervalle d'étude.

Initialisation

pas **prend la valeur** (b-a)/10
x **prend la valeur** a

Traitement

Pour k **variant de** 1 **à** 10
Tant que [f(x+pas) **est supérieur ou égal** f(x)**]**
x **prend la valeur** x+pas
x **prend la valeur** x+pas
pas **prend la valeur** –pas/10

Sortie

Affiche x, x+ 20*pas, f(x+20*pas)

Les remarques faites pour d'autres extraits s'appliquent : déclarations de statut ambigu, absence d'entrées alors que la sortie est prévue, absence de ponctuation, majuscules inso-

lites, mélange entre indicatif et impératif ..On notera que remplacer le signe ≥ par sa version littérale n'apporte absolument rien, si ce n'est une expression barbare, et que cette curieuse précaution s'accommode mal de l'usage des crochets.

Plus fondamentalement, cet algorithme place une itération à l'intérieur d'une autre, utilisant deux types différents d'itération à la fois. Or une seule suffit, comme notre analyse l'a montré.

Regardons, pour terminer, la traduction PYTHON de l'algorithme, laquelle soulève beaucoup de questions également :

```
def f(x)
    return 3.0*x-x**2

a=-1.0
b=3.0
pas=(b-a)/10
x=a
for i in range(8) :
    while (f(x+pas)>f(x)) :
           x=x+pas
    x=x+pas
    pas=-pas/10
print x
print x+20*pas
print f(x+10*pas)
```

D'abord le programme commence par la définition d'une fonction. C'est un peu gênant pour une algorithmique qui n'a pas envisagé de parler des fonctions informatiques. Par ailleurs les extrémités a, b de l'intervalle sont introduites comme des constantes.

En même temps, cela montre la faiblesse de l'analyse algorithmique. Il ne sert à rien de dire que *f* est une variable ; il faut au moins expliquer qu'on va fournir la fonction *f* à la machine, de même que les extrémités de l'intervalle, ce qui est fait dans la traduction.

Ensuite aux ésotérismes provoqués par les nécessités dont on vient de parler, s'ajoutent quelques autres. La notation `1.0` indique que la valeur sera prise par une variable réelle, sur laquelle les calculs se feront en virgule flottante.

Enfin pourquoi « `range(8)` » plutôt que « `range(10)` » pour traduire « **Pour** k **variant de** 1 **à** 10 » ? Comme dans un autre extrait, on a la forte impression ici qu'aucun véritable effort d'adaptation n'a été consenti.

Références

[B] Bertrandias (Françoise) .- Mathématiques pour les sciences de la nature et de la vie, Presses universitaires de Grenoble, Grenoble, 1997

[D1] Direction générale de l'enseignement scolaire.- Programme pour la classe de seconde - Algorithmique - Mathématiques, lycée. Le contenu se trouve sur le site Eduscol à l'adresse

http://media.eduscol.education.fr/file/Programmes/20/1/pgm2nde2009_109201.pdf

[D2] Direction générale de l'enseignement scolaire .- Ressources pour la classe de seconde - Algorithmique - Mathématiques, lycée. Le contenu se trouve sur le site Eduscol à l'adresse

http://media.eduscol.education.fr/file/Programmes/17/8/Doc_ress_algo_v25_109178.pdf

[D3] Direction générale de l'enseignement scolaire.- Ressources pour la classe de seconde - Fonctions - Mathématiques, lycée. Le contenu se trouve sur le site Eduscol à l'adresse
http://media.eduscol.education.fr/file/Programmes/18/1/Doc_ressource_fonctions_109181.pdf

[F1] Ferrier (Jean-Pierre) .- La notion de fonction.
http://www.iecn.u-nancy.fr/~ferrier/Fonc.pdf

[F2] Ferrier (Jean-Pierre) .- Formation pilote de l'IPR de Lorraine du PAF 2008. Le contenu de cette formation se trouve sur le site de l'Irem de Lorraine à l'adresse

http://www.irem.uhp-nancy.fr/Algo/Algo.htm

[L] Latocha (Vladimir) .- L'algorithmique entre rigueur et créativité, exposé aux journées inter académiques, décembre 2009

http://www.iecn.u-nancy.fr/~latocha/Enseignement/Divers/exposeLatocha.pdf

[W] Wirth (Nikaus) .- Algorithms + Data Structures = Programs, Prentice-Hall Series in Automatic Computation