Algol 60, 60 ans après

Thierry Dumont [1]

L'anniversaire d'Algol semble être un peu passé inaperçu. Pourtant il méritait bien qu'on s'intéresse à lui, ce grand-père de la plupart des langages de programmation procéduraux contemporains. L'auteur des ces lignes a récupéré cinq livres dans les rejets de la bibliothèque de mathématiques de son université : deux volumes de [15] [2] *(des programmes écrits dans les années soixante), et trois tomes d'*Actes de congrès de l'AFCAL(TI) [3] *(1960, 61 et 63) qui ensemble donnent une bonne idée de la pratique du calcul et de la programmation d'il y a 60 ans. Le point de vue ici est différent de celui de* [7]*, qui s'intéresse à la réalisation des premiers compilateurs : peut-on étudier, compiler et exécuter les programmes de* [15]*, apprendre et pratiquer le langage Algol 60 ? Les* Actes des congrès de l'AFCAL(TI) *permettent-ils d'évaluer l'adaptation du langage aux pratiques et aux besoins de l'époque de leur publication ? Enfin, comme* [15] *présentent des résultats de calculs, peut-on extraire des connaissances de ces résultats ?*

1. Institut Camille Jordan.
2. Produit des Recherches coopératives sur programme (RCP) 30 et 136 du CNRS.
3. Association française du calcul, puis Association française du calcul et de traitement de l'information.

DOI:10.48556/SIF.1024.20.65 **1024** – Bulletin de la SiF, numéro 20, Novembre 2022, pp. 65–76

Une présentation rapide d'Algol 60

Naissance

En octobre 1955, se tient à Darmstadt un symposium international d'où ressort la nécessité d'unifier les langages qui commencent à se développer dans le plus grand désordre [4] vers un modèle de langage indépendant des machines. Des meetings se succèdent : à l'automne 1957 [5], en avril 58 (meeting de l'ACM) et enfin en mai et juin 58 à Zurich. Ils aboutissent à la définition d'Algol 58. Malheureusement, Algol 58 est plein d'inconsistances : lors de réunions à Paris en novembre 59 et janvier 60, auxquelles participent entre autres J. Backus, P. Naur, H. Rutishauser et le français B. Vauquois, on procède à une reconception complète du langage qui devient *Algol 60*. Mais des inconsistances persistent et, après des meetings à Münich (1962) et à Delft (1963), sort sous l'égide de l'IFIP [6] un *Revised Algol Report*. Finalement, les difficultés d'implantation mènent l'IFIP à créer un *subset of* Algol 60. On trouvera une histoire partielle mais beaucoup plus détaillée dans P. Naur [13].

Le langage

Le texte de H. Rutishauser [16] est une remarquable description d'Algol 60 et aussi un magnifique cours de programmation, très pédagogique, accompagné d'exemples provenant tous du calcul scientifique (de l'époque). Il contient aussi la description du langage sous la forme de Backus-Naur (développée précisément pour Algol). Le langage décrit est celui du *subset of* Algol 60. On pourra aussi consulter le *Revised Report on Algorithmic Language Algol 60* [14].

La présentation ci-dessous suppose que le lecteur est un peu familiarisé avec les langages procéduraux actuels. On ne détaillera pas ce qui fait aujourd'hui partie d'une « culture générale » de la programmation, culture qui n'existait pas en 1960.

Types

Ils sont en nombre très limité (3) : `real, integer, boolean`. On remarque qu'il n'y a qu'une seule précision pour les flottants, et pas de type `complex`. On peut construire des tableaux de ces types primitifs avec des bornes entières quelconques :

```
real array a[-5:30];
integer array b,c[0:50,-11:30];
boolean array bl[0:20];
```

4. Organisé par la *Geselshaft für angewandte Mathematik und Mechanik* (GAMM).
5. Organisé par la GAMM et l'*Association for Computing Machinery* (ACM).
6. *International Federation for Information Processing*.

(ici `b` et `c` sont deux tableaux de mêmes bornes). Les limites sont incluses.

Algol 60 connaît les `label` (étiquettes) et les `switch` ; ces derniers sont très semblables aux `switch` du langage C. On verra plus loin que les uns et les autres peuvent être arguments de procédures.

Structures de contrôle

Elles sont au nombre de trois seulement !

(1) `if`. Voici deux exemples :

```
if x<0 then y:=1;
if x<0 then y:=1 else
begin
        z:=1;
        w:=3
end
```

(2) La boucle `for`. Elle n'opère pas forcément sur des entiers (à vos risques et périls !) :

```
for x:=10 step -0.1 until 0.2 do
        s:=s+exp(x);
```

La boucle `for` est peut-être la partie la plus étrange du langage. Ainsi :

```
for z:=10,11,9,8 do outreal(1,z);
```

va produire : 10 11 9 8 [7]. Mais, voici plus délicat :

```
integer k;
for k:= 10,k/3 while k>0 do
outinteger(1,k);
```

va produire : 10 3 1 ; ce qui correspond à la boucle `for(k=10;k>0;k=k/3)` en langage C.

Algol permet des constructions intéressantes :

```
real array a[1:n,1:n];
for i:=1 step 1 until n do
for j:=1 step 1 until n do
a[i,j]:= if j<i then 0 else 1;
```

(3) Enfin, Algol 60 possède l'infâme `goto`, mais on peut en limiter considérablement l'usage : par exemple l'utiliser pour sortir conditionnellement d'une boucle car il n'y a pas d'instruction *break*.

7. Notons que les entrées-sorties n'étaient pas normalisées en Algol 60. `outreal` n'est qu'une réalisation possible.

Blocs

Les blocs `begin...end` permettent d'intéressantes allocations dynamiques :

```
n:=10;
begin
        integer array y[if n=1 then 0 else 2:
        if n=10 then n else 2*n];
end;
```

`own` permet de déclarer des variables persistantes dans un bloc :

```
begin
        own integer array y[0:20];
end;
```

Procédures

Donnons l'exemple d'une procédure qui calcule un zéro approximatif d'une fonction strictement croissante par bissection en supposant que $f(a) < 0$ et $f(b) \geqslant 0$:

```
procedure bisect(eps,f,a,b);
real a,b,eps; real procedure f;
begin
        real x;
        for x:= (a+b)/2
                while b-a>eps do
                        if f(x)<0 then a:=x else b:=x;
end bisect;
```

Algol 60 permet d'emboîter les procédures (une procédure `B`, dans une procédure `A` n'est pas visible de l'extérieur de `A`). Par défaut, les paramètres sont passés par référence, mais on peut les passer par valeur en les déclarant `value`. Ainsi, dans l'exemple précèdent, on aurait pu écrire :

```
procedure bisect(eps,f,a,b);
value eps;
real a,b,eps; real procedure f;
```

Les paramètres effectifs correspondants à `a` et `b` dans le programme appelant sont modifiables mais pas celui correspondant à `eps`. Passer une étiquette comme paramètre permet, entre autres, de simuler un système d'exceptions, indisponible en Algol 60 :

```
integer procedure facto(n,exception);
value n;
integer n; label exception;
begin
        if n<0 then goto exception;
```

```
        if n=1 or n=0 then
                facto := 1
        else
        facto:= n*facto(n-1,exception);
end;
begin
integer n,k;
        for n:=12 step -1 until -1 do
                k:= facto(n,excep);
        goto fin;
        excep: outstring(1,"erreur.");
fin: end
```

C'est à peu près tout ! Avec les quelques lignes ci-dessus et le texte de Rutishauser [16], le lecteur doit pouvoir programmer en Algol 60. Mais, comment ne pas citer les mots de C.A.R. Hoare : « *Voici un langage très en avance de son temps, il n'a pas seulement été une amélioration de ses prédécesseurs mais aussi une amélioration de presque tous ses successeurs* ».

La réception d'Algol 60 — Les premiers compilateurs

L'intérêt pour Algol et la floraison des compilateurs

L'intérêt pour Algol peut se mesurer à la liste des pays où le journal « Algol Bulletin » était reçu ; parmi ceux-ci les pays d'Europe de l'Ouest et d'Amérique du Nord, la Tchécoslovaquie, l'Allemagne de l'Est, la Pologne, l'URSS, le Mexique et la Chine Populaire. Une liste incomplète des nombreuses implantations est donnée dans [21] ; la réalisation d'un compilateur par C. Pair et ses collègues est décrite dans [7] ; les *Actes de l'AFCALTI* (1963) mentionnent aussi un compilateur pour Bull Gamma 60. On trouve même la trace de la réalisation d'un compilateur à l'université de Pékin [6].

L'échec

Algol a échoué à devenir un langage sinon universel, du moins un langage répandu, pour finalement disparaître. Les raisons sont nombreuses. Notons déjà qu'IBM a sorti un brouillon du langage Fortran en 1954, un premier compilateur dès 1957 et que des compilateurs Fortran existeront sur IBM 709, 650, 1620 et 7090, ce qui, ajouté à la puissance d'IBM à l'époque, donnait un sérieux avantage à Fortran. P. Mounier-Kuhn donne dans [12] une description détaillée de « l'ascension et la chute » d'Algol en France, de l'enthousiasme des universitaires, avec le soutien d'institutions comme le CNRS, jusqu'au recul et à l'abandon dans les plans industriels, tout en montrant le rôle fondateur de l'intérêt pour Algol dans le développement des bases de l'informatique en France. Dans [13], P. Naur explique

« qu'il était sous-entendu que le domaine [d'application du langage] était celui qui sera plus tard appelé *traitement de l'analyse numérique* ou même *calcul scientifique*. À cette époque, les autres applications commençaient juste à émerger, ..., tandis que le *calcul scientifique* pouvait reposer sur une vieille tradition de méthodes de calcul et d'analyse mathématique... »

Cette assertion mérite d'être revisitée. Qu'était le calcul scientifique en 1960 ? Les *Actes des congrès de l'AFCAL* permettent d'y répondre : en 1960, on calcule... ce qui est accessible, avec les limitations dues à la faible puissance et à la taille mémoire très limitée des calculateurs : systèmes linéaires de petite taille, interpolation, résolution de systèmes d'équations différentielles simples (avec des méthodes explicites). L'attaque frontale de problèmes de physique (par la résolution des équations aux dérivées partielles) est à peine envisagée. Calculer des tables numériques n'est pas encore désuet : J.M. Laborde réalisera des tables de fonctions en 1963 [10] !

En ce sens là, Algol 60 est parfaitement adapté au calcul scientifique de l'époque. L'existence d'un seul type de flottants le montre aussi : les problèmes résolus sont plutôt bien posés, ne serait-ce que parce qu'ils sont de petite taille, et il n'y a pas de réel besoin de précision étendue.

Mais on peut aussi remarquer qu'en 1963, l'intérêt du calcul scientifique se déplace vers la résolution des équations aux dérivées partielles [8]. En 1964, la NASA lance le développement de son code Nastran [4] (calcul de structures par éléments finis) entièrement en Fortran IV. On assiste au basculement de l'analyse numérique et du calcul scientifique vers la résolution numérique des équations aux dérivées partielles. Le livre de Temam [19] intitulé « Analyse Numérique » est un manifeste de *l'École Lions* : l'analyse numérique, et donc le Cacul scientifique, ce sera à présent la résolution numérique des équations aux dérivées partielles. Mais l'implantation de la méthode des éléments finis nécessite de manipuler des structures de données relativement complexes (maillages, matrices creuses, etc.) pour lesquelles on ne voit pas vraiment d'avantages à adopter Algol 60 plutôt que Fortran. Tous les grands codes éléments finis entrepris dès lors et jusqu'aux années 90 seront d'ailleurs programmés en Fortran (IV ou 77), pour le meilleur ou pour le pire (plutôt pour le pire).

Les calculs en précision étendue apparaîtront vite comme nécessaires, dès que la taille des problèmes à résoudre pourra augmenter (le conditionnement des systèmes linéaires croît comme h^{-2} (h : pas de discrétisation, taille des éléments finis...) pour des problèmes elliptiques comme l'équation de Poisson : ce sont donc des problèmes assez mal conditionnés).

Les enfants d'Algol 60

On ne s'intéresse ici qu'aux enfants directs, nés avant 1970. Créé par Ershov, en URSS à Novosibirsk, *Alpha* ajoute à Algol 60 des manipulations de tableaux et de

8. Se référer à la table ronde de l'AFCALTI sur les EDP dans les actes de 1963.

matrices avec des « slices ». Conçu comme successeur d'Algol 60, *Algol 68* [11] est extrêmement rigoureux, mais très complexe (« *no implementations and no users* »). C'est un échec. Au moins un compilateur existait au début des années 70 : celui de *Royal Radar* en Grande Bretagne. Conçu par N. Wirth, qui n'a pas voulu suivre le mouvement Algol 68, *Algol W* [18] a été en vogue jusqu'au début des années 80, au moins dans l'enseignement. Il ajoute les types `long real, complex, long complex, bits`, « un ensemble ordonné de valeurs simples » avec `record`, la notion de référence à un `record` avec `reference`, ainsi qu'une bibliothèque standard.

Développé au *Norvegian Computing Center*, *Simula 67* est probablement le premier langage à classes, implantant des coroutines et nombre de concepts qui deviendront populaires bien plus tard.

Les machines Burroughs, à partir du B5000 en 1961, étaient des machines à architecture de pile et leur système d'exploitation était programmé avec des extensions d'Algol 60, successivement appelées *Espol et Newp*, ceci bien avant Multics (programmé en PL/1) et les différents Unix (programmés en C). Notons que des compilateurs sont disponibles sous Linux pour Simula 67 [2] et Algol 68 [1].

Faire revivre des programmes écrits en Algol 60

Pour rejouer les programmes écrits en Algol 60, il faut disposer d'un compilateur. Deux au moins sont disponibles : *GNU Marst* qui est peut être un peu en déshérence et *jff-algol*, réalisé et maintenu par Jan van Katwijk, disponible sur GitHub [20] ; c'est ce compilateur que nous avons utilisé.

On choisit ensuite un programme dans [15], par exemple la méthode de Bairstow qui permet de trouver les racines d'un polynôme $P(x)$ à coefficients réels sans utiliser d'arithmétique complexe (absente d'Algol 60). Pour cela, on cherche un diviseur quadratique de P, et on réitère le processus jusqu'à avoir trouvé toutes les racines de P. Plus précisément, B et C étant donnés, on a, en divisant P par (x^2+Bx+C) :

$$P(x) = (x^2+Bx+C)\ Q(x) + (Rx+S),$$

ce qui définit une application $\mathscr{F} : (B,C) \mapsto (R,S)$. On cherche alors, par la méthode de Newton, un couple (B,C) qui annule R et S. Le calcul des dérivées partielles de R et S par rapport à B et C est donné dans [8]. Pour implanter la méthode, il suffit de savoir diviser des polynômes par des trinômes, la méthode de Newton ne nécessitant que la résolution de systèmes linéaires de taille 2.

La figure 1 montre quelques extraits du code, programmé par Lagouanelle à Toulouse (les programmes de [15] ne sont pas anonymes). Surprise : c'est de l'Algol en Français, ce qui est possible, car Algol 60 fait une distinction entre une forme du langage pour les publications et des formes « pratiques », tenant compte aussi des contraintes matérielles [9]. Comment transcrire ce code ? un OCR s'avère insuffisant :

9. Des caractères n'étaient pas disponibles sur certaines machines comme ^ ou [ ].

il faut s'armer de patience et tout compléter, corriger vérifier. Le problème des mots clés en français n'est pas grave : on peut écrire un petit script pour les transcrire en anglais, mais Jan Van Katwijk a ajouté une option à son compilateur pour traiter l'« Algol en français ».

Une fois le code compilé et exécuté, il faut vérifier, dans la mesure du possible, qu'il est correct. Pour cela une programmation de la même méthode dans un autre langage (ici en Julia) permet de vérifier dans une certaine mesure la correction du programme, aux erreurs de troncature près. On peut aussi s'intéresser à des cas où les coefficients sont rationnels (ou même algébriques) et utiliser SageMath [5] qui permet de calculer dans une extension algébrique.

Le programme semble bien être juste !

Que dire de la qualité de la programmation ? On remarque qu'il y a quatorze `goto` dans le code originel, et qu'il n'y a pas de découpage en sous-procédures (et donc pas de procédures imbriquées). Clairement, ce n'est pas le mieux programmé de la collection, ce qui contraste avec les codes réalisés par exemple à Grenoble, qui ont

```
PROCEDURE BAIRSTOW(N,A,S0,P0,EPS1,EPS2,MAXITER,X,Y,DIVERG) ;
          VALEUR N,A ; ENTIER N,MAXITER ; REEL S0,P0,EPS1,EPS2;
          TABLEAU A,X,Y ; ETIQUETTE DIVERG ;
       COMMENTAIRE
        BAIRSTOW CALCULE LES RACINES DE P(X)=A(0)X**N+A(1)X**(N-1)+..A(N)
        POLYNOME DE DEGRE N.SES COEFFICIENTS,REELS,SONT RANGES DANS LE

      DEBUT TABLEAU B,C[-2:N] ;
            REEL R,T1,Y1,Y2,S,P,S1,P1,DELTA,DISCR,S2,P2,X1,X2 ;
            ENTIER I,K,NITER ;
             B[-2]:= B[-1]:=0 ;
             C[-2]:= C[-1]:=0 ;
             T1 := EPS1 ;
            SI N=0 ALORS ALLERA  FIN ;
            SI N=1 ALORS DEBUT B[0]:=A[0]; B[1]:=A[1];
                                ALLERA  DERZERO
                         FIN ;
       TESTO : I:=N+1 ;
               POUR I:=I-1 TANTQUE  A[I]=0 ET I > 1 FAIRE
               DEBUT X[I]:=0 ;Y[I]:=0 ;N:=N-1 ;
               FIN ;
       DEGTEST: SI N > 2 ALORS ALLERA  BOUCLE ;
                 S2:=-A[1]/A[0] ;
                 P2:= A[2]/A[0] ;
               ALLERA  ETIQ2 ;
               S:=S0 ; P:=P0 ;
       BOUCLE : EPS1 :=T1 ; NITER :=0 ;
               SI A[N-2]=0 ALORS DEBUT
               SI (ABS(S)+ABS(P)=0) ALORS DEBUT
               S:=0.1 ; P:=0.2 ; FIN FIN ;
         ITER : NITER :=NITER+1 ;
```

FIGURE 1. Méthode de Bairstow (extrait)

tous à peu près le même style, excellent, faisant probablement apparaître une culture locale commune de la programmation.

On peut aussi reprogrammer la méthode de Bairstow en Algol 60, en s'inspirant du code écrit en Julia. On remarque que le code Julia se transcrit facilement en Algol, avec *un seul* `goto`. La totalité de ces morceaux de code, ainsi que le code originel, sont disponibles sur GitHub [9].

Une expérience « d'archéologie numérique »

Dans les *Procédures*, il n'est pas fait mention des machines utilisées, qui diffèrent pourtant d'un centre à un autre. Les nombres flottants ne sont pas normalisés à l'époque, pas plus que la taille de leur mantisse. On peut tenter de deviner le type de machine utilisé en se servant des résultats fournis.

Ainsi, quelle était la machine utilisée à Grenoble avant 1967 (avant l'arrivée de l'IBM 360) ? Page 280 du tome 1 des *Procédures*, on trouve le code d'un ajustement polynomial aux moindres carrés, écrit par André Eberhard (auteur aussi d'une remarquable Transformée de Fourier rapide). L'auteur utilise une méthode naïve : ajuster les coefficients a_i de $p(x) = \sum_{i=0}^{m} a_i x^i$ en résolvant $\min_{\{a\}} \sum_{i=1}^{n} (p(x_i) - y_i)^2$. C'est une méthode très instable car la matrice du système linéaire à résoudre est un avatar de la matrice de Hilbert [10]. De plus, le système linéaire est résolu en utilisant la routine `gresolsyslin` (Tome 1 des *Procédures*, due à N. Gastinel). C'est une élimination de Gauss sans choix du pivot, ce qui est aussi source d'instabilité. Mais l'instabilité numérique peut être utile...

Quelle était donc la précision des flottants utilisés ?

Le problème résolu dans les *Procédures* consiste à ajuster un polynôme de degré 4 parmi 21 points (x_i, y_i) où les x_i sont équirépartis dans $[0,1]$ et $y_i = \exp(x_i)$. L'auteur trouve pour $a_i, i = 0,4$ les valeurs 0.10003376×10, 0.99835205, 0.50927734, 0.14135742 et $0.69702148 \times 10^{-1}$. On se sert alors de SageMath [5] qui, utilisant la bibliothèque Mpfr [3], permet de calculer avec des flottants de précision quelconque. On transcrit les programmes des *Procédures* dans SageMath obtenant ainsi un programme à peu près analogue au code Algol, mais paramétré par la précision des flottants (le nombre de bits de la mantisse). Le conditionnement [17] du système linéaire qu'on résout est de 689500, d'où on déduit qu'en calculant avec 350 bits de précision, on obtient une solution a^{ref} avec une erreur relative de l'ordre de 10^{-100} sur les coefficients a, donc pratiquement exacte. Définissant la distance à la solution « exacte » par $\mathrm{d(a)} = \max_{\mathrm{i}=0,4} |\mathrm{a_i} - \mathrm{a_i^{ref}}|$, on peut calculer cette distance en fonction de la précision du calcul. La distance de la solution calculée dans les *Procédures* à la solution « exacte » est de $1.48724517163778\ 10^{-3}$. En reportant cette distance sur

10. On trouve aussi dans [15] une implantation de la bonne méthode utilisant des polynômes orthogonaux.

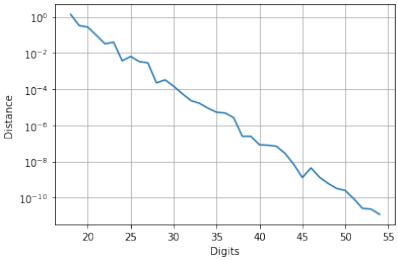

FIGURE 2. Distance / Précision.

FIGURE 3. Dr. Follamour : un IBM 7044.

le graphique de la figure 2, on trouve que le flottant de la machine utilisée avait une précision de 27 bits.

Quelles étaient les machines dotées d'un flottant avec une mantisse de 27 bits ? Il y avait des Dec PDP, quelques machines chez Univac et, chez IBM, la série 7000. Les PDP étaient des minis (des machines de laboratoire) et Univac était très peu répandu dans le monde de la recherche. Il reste IBM et on se souvient alors qu'il existait à cette époque un Centre de recherche IBM à Grenoble qui a beaucoup collaboré avec l'IMAG. La machine était donc un IBM, série 7000, plus précisément un IBM 7044 [11] (une machine célèbre, voir la figure 3).

Les IBM de la série 7000 avaient des mots de 36 bits et une mémoire pouvant aller jusqu'à 32 kmots. En simple précision, la mantisse avait une taille de 27 bits, mais il existait une double précision avec une mantisse de 54 bits [12]. En 1963, IBM a introduit des machines moins coûteuses, avec moins d'instructions dont faisait partie

11. Jean-François Maître (Pr. émérite, École centrale Lyon) a retrouvé un article de la presse locale l'annonçant.

12. Algol 60 avait un seul type de flottants, mais les compilateurs Fortran de l'époque avaient une option pour passer tous les flottants en « double ». Une option identique aurait certainement été possible pour Algol 60.

le 7044. Parmi les fonctionnalités sacrifiées, il y avait... la double précision. Pourquoi n'a-t-on pas acheté une machine munie de la double précision ? Peut-être qu'on n'en avait pas les moyens, car les unités flottantes étaient fort coûteuses à l'époque, ou peut être qu'on s'est dit qu'on n'en avait pas besoin vu les calculs qu'on projetait d'effectuer (voir plus haut !). Mais la vérité est peut être un mélange des deux !

Références

[1] Genie algol 68g. http://progopedia.com/implementation/algol68g.

[2] Gnu cim. https://www.gnu.org/software/cim/.

[3] The gnu mpfr library. https://www.mpfr.org/.

[4] Nastran. https://fr.wikipedia.org/wiki/Nastran.

[5] Sagemath. https://www.sagemath.org/.

[6] Chen kunqiu, prix xia peisu (china computer federation). https://www.ccf.org.cn/Computing_history/Updates/2021-04-28/727169.shtml, 2020.

[7] Marion Créhange, Pierre Lescanne, and Alain Quéré. L'informatique de claude pair. https://doi.org/10.48556/SIF.1024.18.91.

[8] Thierry Dumont. Bairstow's method. https://github.com/Thierry-Dumont/Vintage_Computing/blob/master/Bairstow/Doc/bairstow.pdf.

[9] Thierry Dumont. Vintage computing. https://github.com/Thierry-Dumont/Vintage_Computing, 2019.

[10] Jean Marcel Laborde. *Tables Numériques de Fonctions élémentaires ...* Dunod, 1963.

[11] C.H. Lindsey and S.G. Van der Meulen. Informal introduction to algol 68. http://www.softwarepreservation.org/projects/ALGOL/book/Lindsey_van_der_Meulen-IItA68-Revised.pdf, 1977.

[12] Pierre Mounier-Kuhn. Algol in france : From universal project to embedded culture. *IEEE Annals of the History of Computing*, 36(4) :6–25, 2014.

[13] P. Naur. The european side of the last phase of the development of algol 60- acm sigplan. https://dl.acm.org/doi/pdf/10.1145/960118.808370?download=true, 1978.

[14] Peter Naur (editeur). Revised report on algorithmic language algol 60. http://homepages.cs.ncl.ac.uk/cliff.jones/publications/OCRd/BBG63.pdf.

[15] etc Noël Gastinel, Préfacier. *Procédures Algol en analyse numérique*. Éditions du CNRS, 1967.

[16] Heinz Rutishauser. Description of algol 60. http://www.algol60.org/docs/Rutishauser_Description_of_ALGOL_60_1967.pdf, 1967.

[17] Michelle Schatzman. *Analyse numérique : cours et exercices pour la licence*. InterEditions, 1991.

[18] Richard L. Sites. Algol w, reference manual. http://www.algol60.org/docsW/AlgolW_STAN.pdf.

[19] Roger Temam. *Analyse numérique*. Presses universitaires de France, 1970.

[20] Jan van Katwijk. Algol 60 compiler. https://github.com/JvanKatwijk/algol-60-compiler.

[21] Wikipedia. Algol 60. https://en.wikipedia.org/wiki/ALGOL_60.