Algol 60, 60 ans après

Séminaire Codes Sources

Thierry Dumont

7 Avril 2022

A l'origine...

Rejets de la bibliothèque de maths (Lyon 1) :

- Actes des congrès de l'AFCAL(TI) (3 tomes).
- Procédures Algol en analyse numérique (2 tomes).

Plan

1. Un peu d'histoire : naissance d'Algol 60.
2. Présentation informelle du langage.
3. Sa « réception ».
4. Algol 60 dans la pratique de l'époque.
5. Variants proches et descendance.
6. Rejouer des programmes écrits en Algol 60.
7. Rejouer des vieux programmes, qu'est-ce que ça nous apprend ?

Naissance

- Darmstadt, Octobre 1955. Symposium international.
  Unification ?
  GAMM [1] Subcommmity for Programming Languages.
- Automne 1957 : GAMM + ACM.
- Philaldelphie, avril 1958. Meeting ACM.
- Zurich, Mai/Juin 1958 => Algol 58.

1. Geselshaft für angewandte Mathematik und Mechanik.

Naissance

- Paris 11/1959 et 01/1960.
  Backus, Naur, Rutishauser, Vauquois...
  Redesign *complet* du langage. Algol 60.

Naissance

- Paris 11/1959 et 01/1960.
  Backus, Naur, Rutishauser, Vauquois...
  Redesign *complet* du langage. Algol 60.
- Rome 62, IFIP.
  Inconsistances => *Revised Algol Report.*

Naissance

- Paris 11/1959 et 01/1960.
  Backus, Naur, Rutishauser, Vauquois...
  Redesign *complet* du langage. Algol 60.
- Rome 62, IFIP.
  Inconsistances => *Revised Algol Report.*
- IFIP : Münich (1962), Delft (1963) :
  Difficile à implanter => IFIP *Subset of Algol 60.*

Naissance

- Paris 11/1959 et 01/1960.
  Backus, Naur, Rutishauser, Vauquois...
  Redesign *complet* du langage. Algol 60.
- Rome 62, IFIP.
  Inconsistances => *Revised Algol Report.*
- IFIP : Münich (1962), Delft (1963) :
  Difficile à implanter => IFIP *Subset of Algol 60.*

Fortran :

- 1954 : brouillon (IBM).
- 1957 : premier compilateur.
- 1960 : disponible sur IBM 709, 650, 1620 et 7090.

Algol 60 : présentation informelle

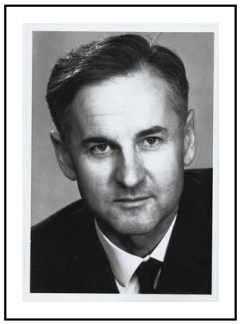
« Voici un langage très en avance sur son temps, il n'a pas seulement été une amélioration de ses prédécesseurs mais aussi une amélioration de presque tous ses successeurs »

—— C.A.R. Hoare.

Algol 60 : présentation informelle

Documents aisément consultables :

➔ Heinz Rutishauser (1918-1970) : *Description of Algol 60* (1967).



➔ Revised Report on Algorithmic Language Algol 60
`http://homepages.cs.ncl.ac.uk/cliff.jones/publications/OCRd/BBG63.pdf`

Le texte de Heinz Rutishauser

- 324 pages.
- Non seulement une présentation d'Algol 60, mais aussi : nombreux exemples. Un cours de programmation !
- Historique.
- La notation BNF d'Algol 60 (créée pour Algol 60).

A quoi ça ressemble ?

Résolution de $\sum_{k=0}^{n} a_k x^k = 0$ par la méthode de Newton.
(fragment de programme, sans les déclarations des variables)

```
begin
rep:  f := g := 0 ;
horn: for k := n step −1 until 0 do
      begin
          g := g × x + f ;
          f := f × x + a[k]
      end ;
      delta := −f/g ;
      x := x + delta ;
      if abs(delta) > eps then goto rep
end
```

A quoi ça ressemble ?

Résolution de $\sum_{k=0}^{n} a_k x^k = 0$ par la méthode de Newton.
(fragment de programme, sans les déclarations des variables)

```
begin
rep:   f := g := 0 ;
horn:  for k := n step −1 until 0 do
       begin
          g := g × x + f ;
          f := f × x + a[k]
       end ;
       delta := −f/g ;
       x := x + delta ;
       if abs(delta) > eps then goto rep
end
```

- Les blocs **begin**...**end**.
- Les affectations (multiples) par :=
- La boucle **for**.
- Les étiquettes et les **goto**.

Les types

real integer boolean label switch

- une seule précision pour les flottants,
- pas de type **complex**.

Les types

real integer boolean label switch

- une seule précision pour les flottants,
- pas de type **complex**.

et les tableaux des types primitifs :

real array
integer array
boolean array

Il y a des chaînes de caractères, mais rien pour les manipuler.

Les types

real integer boolean label switch

- une seule précision pour les flottants,
- pas de type **complex**.

et les tableaux des types primitifs :

```
real array
integer array
boolean array
```

Il y a des chaînes de caractères, mais rien pour les manipuler.
Les tableaux ont des bornes quelconques :

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

Les types

real integer boolean label switch

- une seule précision pour les flottants,
- pas de type **complex**.

et les tableaux des types primitifs :

```
real array
integer array
boolean array
```

Il y a des chaînes de caractères, mais rien pour les manipuler.
Les tableaux ont des bornes quelconques :

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

integer array   aa,bc,cd[-5:30];
```

- les boucles **for** n'opèrent pas obligatoirement sur les entiers :

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

- les boucles **for** n'opèrent pas obligatoirement sur les entiers :

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

- les **if** :

```
if x<0 then y:=1;
```

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

Des **for** assez étonnants..

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

  va donner : 10 11 9 8.
  ```

Des **for** assez étonnants..

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

  va donner : 10 11 9 8.

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

  va donner : 10 3 1.

Des **for** assez étonnants..

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

  va donner : 10 11 9 8.

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

  va donner : 10 3 1.

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

  9 8 7 6 5 4 3 2 1
  ```

Assez intéréssant :

```
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;
```

Procédures

```
begin
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;

real procedure fonc(x);
  real x;
  begin
   fonc := x*x-1;
  end fonc;
```

```
begin
real a,b,eps;

a:=-1; b:=+2; eps :=1.e-8;
bisect(eps,fonc,a,b);
outreal(1,a); outreal(1,b); outstring(1,"\n")
end
```

Une fonction

```
comment factorielle, recursive;
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;
```

Une fonction

```
comment factorielle, recursive;
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;
```

- **value**.
- **label** : joue *ici* le même rôle qu'une exception dans les langages modernes.
- Une fonction est une procédure qui renvoit quelque chose (type de base).

```
comment le programme principal;

begin
 integer n,k;

 for n:=12 step -1 until -1 do
          k:= facto(n,excep);
 goto fin;

 excep: outstring(1,"erreur n= ");
        outinteger(1,n);
        outstring(1,"\n");

fin: end
```

```
comment le programme principal;

begin
 integer n,k;

 for n:=12 step -1 until -1 do
          k:= facto(n,excep);
 goto fin;

 excep: outstring(1,"erreur n= ");
        outinteger(1,n);
        outstring(1,"\n");

fin: end
```

- Noter la boucle **for** avec incrément négatif.
- Les étiquettes pour gérer une exception.

Langage avec blocs

```
integer n;
n:=20:
begin
   a= integer array[1:n,-3:2*n];
   integer k;
   ...
end;
```

- allocation dynamique dans les blocs.

Langage avec blocs

```
integer n;
n:=20:
begin
   a= integer array[1:n,-3:2*n];
   integer k;
   ...
end;
```

- allocation dynamique dans les blocs.

On peut même écrire :

```
integer n;
n:=20:
begin
a=real array[if n=1 then 0 else n/2:if n>2 then
                                    n else 2*n];
   ...
end;
```

own : variables persistantes

```
real procedure trux(x,init);
boolean init; integer x;
begin
 own integer  z;
 if init then  z:=1;

 x:=x+1;
 if x>3 then z:=z+2;

 trux:=z;
end;
```

switch

assez semblable au switch du C.

```
switch S:=label1,label2,label3;

integer i;
i:=2;
goto S[i];

label1: outstring(1,"un");
        goto fin;
label2: outstring(1,"deux");
       goto fin;
label3: outstring(1,"trois");

fin: outstring(1,"\nfin\n");
```

On peut passer un **switch** comme paramètre à une procédure !

Procédures emboîtées

Un moyen d'ajouter de l'abstraction.

```
begin
 integer procedure xx(n,a);
  integer n; integer array a;
  begin
  integer su;
   integer procedure sum(x);
     integer array x;
     begin
       integer s,i;
       s:=0;
       for i:=1 step 1 until n do
           s:= s+x[i];
     sum := s;
    end sum;
  su := sum(a);
  xx := 2 * su;
  end xx;
```

```
comment main----------------------;
integer n,r,i;
n:=10;
begin
 integer array y[1:n];
 for i:=1 step 1 until n do y[i] :=i;
 r:= xx(n,y);
end
comment end main--------------------;

end
```

Bilan

Algol 60 apparaît comme un langage très moderne pour son temps :

- Paramètres déclarés `value`.
- Récursif.
- Procédures dans les procédures.
- Indices des tableaux : `real array x[-30:20]`.
- etc.

Apparaît comme simple aujourd'hui...mais en était-il ainsi il y a 60 ans ?

La réception d'Algol 60

- Algol 60 est difficile à implanter (à l'époque).
  - La compilation est une science naissante.
  - Les machines sont peu adaptées (entre autres : les piles sont plus ou moins difficiles à implanter).
- Fortran est là avant et bénéficie du soutien d'IBM (le Microsoft de l'époque). Fortran IV apparaît en 1962.

Au début des années 70, Algol 60 a pratiquement disparu (il existe un compilateur sur la gamme IRIS de la CII).

- Peter Naur, Jørn Jensen : implantation complète en 1961 (Danemark).
- Gamma 60 (Compagnie des machines Bull) : compilateur en 1962.
- Claude Pair : réalisation d'un compilateur, achevé en 1965, sur IBM 1620 [2].

« ALGOL 60 sera fondamental dans la constitution d'une science informatique, par les notions qu'il introduit et les problèmes que pose sa compilation ».

2. C.Pair a appris la programmation entre autres avec Marion Créhange, dont il a dirigé la thèse d'état. M.Créhange, née en 1937 est décédée le 28 mars 2022.

Algol 60 dans la pratique de l'époque

Que calculait-on, que programmait-on autour de 1960 ?
Que pouvait-on espérer calculer, vu la faible puissance des ordinateurs ?

Les congrès de l' AFCAL(TI) (1960,61 et 63) :
AFCAL (J. Kuntzmann, 1957 (ou 1959 ?)) $\rightarrow$ AFCALTI (1962) $\rightarrow$ AFIRO (1969) $\rightarrow$ AFCET.

Les problèmes de calcul auxquels ont s'intéresse

- Calcul Scientifique = Analyse numérique, applications, implantation en machine, machines, nombres flottants.
- Non numérique : graphes, recherche opérationelle, traduction automatique, gestion, etc.

Le Calcul recouvre la plus grande partie des actes.

- Problèmes numériques *de base :* algèbre linéaire sur des matrices pleines,intégration numérique, etc.
- Problèmes de petite taille (évidemment, vu les capacités des machines).

Il n'est pas étonnant que, à la fin des années 50, la simple précision (un seul type **real** en Algol) ait pu paraître suffisante.

Pas ou peu de problèmes de physique : pas d'équations aux dérivées partielles ni de grands systèmes d'EDOs.

Le code JADE (EDF) est un contre-exemple ; calcul *extrême* : stockage des matrices sur bande...

- Frémissement en 63 (table ronde) . Conscience d'un retard ?
  - US : développements de codes éléments finis (NASTRAN).
  - même en Chine : Fengkang : théorie et pratique des éléments finis (calcul d'un barrage).Xia Peisu

Variants et descendance

Burroughs algol's : ESPOL & NEWP

B5000 en 1961.
Architecture à pile, directement prévue pour implanter Algol.

Burroughs algol's : ESPOL & NEWP

B5000 en 1961.
Architecture à pile, directement prévue pour implanter Algol.

Les langages ESPOL et NEWP : larges extensions d'Algol.
Système programmé en ESPOL puis NEWP.

Premier (?) système d'exploitation programmé dans un langage évolué.

Alpha

Ershov, URSS (Novosibirsk).
Algo 60 + tableaux, matrices et « slices ».

Algol-W

Créé par N. Wirth, en vogue jusqu'au début des années 80.

Manuel de référence :
`http://www.algol60.org/docsW/AlgolW_STAN.pdf`

Rajout de types :

- **long real**, **complex**, **long complex**, **bits**, **reference**
- **record** : "un ensemble ordonné de valeurs simples".
  reference : s'entend comme référence à un **record**

- Quelques différences syntaxiques :
  - tableaux **integer array a(1 : :10,2 : :20)**.
  - **procedure truc(real array a,b(*,*)) ;**

- Quelques différences syntaxiques :
  - tableaux **integer array a(1 : :10,2 : :20)**.
  - **procedure truc(real array a,b(*,*)) ;**
- Exemple de **record** :

```
record NODE (reference (NODE) left,right)
```

- Quelques différences syntaxiques :
  - tableaux **integer array a(1 : :10,2 : :20)**.
  - **procedure truc(real array a,b(*,*)) ;**
- Exemple de **record** :

```
record NODE (reference (NODE) left,right)
```

- Une bibliothèque standard est définie (fonctions mathématiques, mais aussi mesure du temps, etc.).
- Appel de procédures écrites dans d'autres langages.
- Gestion d'exceptions.

Algol-68

`http://www.softwarepreservation.org/projects/ALGOL/book/Lindsey_van_der_Meulen-IItA68-Revised.pdf`
Conçu comme successeur d'Algol 60.

Extrêmement rigoureux, mais très complexe “no implementations and no users".

En Angleterre, compilateur du « Royal Radar »(Algol 68 R).

Algol-68

`http://www.softwarepreservation.org/projects/ALGOL/book/Lindsey_van_der_Meulen-IItA68-Revised.pdf`
Conçu comme successeur d'Algol 60.

Extrêmement rigoureux, mais très complexe “no implementations and no users".

En Angleterre, compilateur du « Royal Radar »(Algol 68 R).

- opérateurs définissables.
- parallélisme (sémaphores).
- ...

Simula-67

Norwegian Computing Centre, Oslo.

Orienté objets (classes, méthodes), coroutines, *parallélisme*, modules...

Simulation à évènements discrets (les auteurs faisaient un cours sur les systèmes d'exploitation).

L'intrus : PL/1

Introduit par IBM.

Sans fondements théoriques sérieux.

A la couleur de l'algol... mais ce n'est pas de l'algol.

Le système Multics était programmé en PL/1.
Question : que donne cette instruction en PL/1 ?

```
a = b = c
```

Des compilateurs Algol W, Algol 68 et Simula 67 existent sous Linux.

Rejouer des programmes écrits en Algol 60

Compilateurs

1. GNU Marst : peut être un peu en déshérence.
2. **Jan van Katwijk : jff-algol**
   « Vivant ». L'auteur a effectué l'adaptation aux mots clés français :
   jff-algol -F prog.alg.
   Deux passes :
   - Traducteur vers C.
   - Compilation du C.

   Disponible sur Github :
   `https://github.com/JvanKatwijk/algol-60-compiler`

Les Procédures Algol en Analyse Numérique

2 tomes.

- RCP 30. Responsable : J. Kuntzmann.
  Fin et publication en 1967.
  Procédures *de base*.
- RCP 136. Responsable : N. Gastinel.
  Publication en 1970.
  Algorithmes moins standard (Gastinel) :
  Ouverture aux thèses et aux études récentes.

Besançon, Clermont-Ferrand, Grenoble, Lille, Nancy (1967), IBP (1967), Toulouse (1967).

1967 : cadre (Préface de Jean Kuntzmann)

- outils pour...
- *...contribuera, nous l'espérons, à promouvoir le langage Algol...*

« Disons d'abord combien de telles initiatives, amenant des laboratoires éparpillés aux quatre coins de la France à travailler ensemble, sont fructueuses ».

1967 : cadre (Préface de Jean Kuntzmann)

- outils pour...
- *...contribuera, nous l'espérons, à promouvoir le langage Algol...*

« Disons d'abord combien de telles initiatives, amenant des laboratoires éparpillés aux quatre coins de la France à travailler ensemble, sont fructueuses ».

- *« Ce travail collectif n'est pas anonyme ».*
- Rassemblement / contrôle /critique de programmes existants.
- Écriture planifiée de nouveaux programmes.
- Séances de travail pour l'homogénéisation de l'ensemble.
- Tests *« par de nombreux passages en machine ».*

Conditions matérielles

- pas d'internet !
- téléphone rare et cher, pas de fax.
- pas de TGV.

Conditions matérielles

- pas d'internet !
- téléphone rare et cher, pas de fax.
- pas de TGV.
- différentes machines, plus ou moins difficiles à programmer.
- flottants non normalisés.
- cartes perforées (voire même ruban perforé).

Grenoble, largement en avance, n'aura son IBM 360/67 sous CP/CMS (« temps partagé », conversationnel) qu'en 1967 (ou 68).

Contenu : a) 1967.

- **Équations non linéaires**
- **Algèbre linéaire**
- **Éléments propres** toutes les méthodes classiques, LR (Rutishauser $\simeq$ 50) (et pas QR Francis $\simeq$ 56, Kublanovskaya (61)).
- **Syst. diff.** *... par la méthode classique de Runge–Kutta....*
- **Équ. intégrales, problèmes intégro-différentiels**
- **Approximation.**

Contenu : a) 1967.

Il n'est pas rare qu'il existe plusieurs méthodes (une bonne, une moins bonne) pour le même problème ; par exemple :

- Résolution de $AX = B$ par la méthode de Gauss :
  1. Sans choix du pivot (N. Gastinel, Grenoble),
  2. Avec la règle du pivot max. (Saya, Grenoble).
- Ajustement polynomial aux moindres carrés :
  1. Naïf (Eberhard, Grenoble).
  2. Polynômes orthogonaux (Hisleur, Grenoble).

Contenu : a) 1967.

Il n'est pas rare qu'il existe plusieurs méthodes (une bonne, une moins bonne) pour le même problème ; par exemple :

- Résolution de $AX = B$ par la méthode de Gauss :
  1. Sans choix du pivot (N. Gastinel, Grenoble),
  2. Avec la règle du pivot max. (Saya, Grenoble).
- Ajustement polynomial aux moindres carrés :
  1. Naïf (Eberhard, Grenoble).
  2. Polynômes orthogonaux (Hisleur, Grenoble).

Mais pas de comparaison numérique.

Explication probable : la rareté de la ressource de calcul et la taille limitée des problèmes qu'on peut résoudre.

Contenu : b) 1970.

- (T) *Calcul rapide d'une transformée de Fourier* (A. Eberhard, Grenoble).

(T) : Résultats de thèses.

Contenu : b) 1970.

- (T) *Calcul rapide d'une transformée de Fourier* (A. Eberhard, Grenoble).
- (T) Estimations d'erreur *backward*.

(T) : Résultats de thèses.

Contenu : b) 1970.

- (T) *Calcul rapide d'une transformée de Fourier* (A. Eberhard, Grenoble).
- (T) Estimations d'erreur *backward*.
- Accélération de la convergence (ε et ρ algorithmes).

(T) : Résultats de thèses.

Contenu : b) 1970.

- (T) *Calcul rapide d'une transformée de Fourier* (A. Eberhard, Grenoble).
- (T) Estimations d'erreur *backward*.
- Accélération de la convergence (ε et ρ algorithmes).
- (T) Algorithme de Strassen (C. de Polignac, Grenoble). (thèse :*Méthodes optimales de calcul de produits de matrices*, 1970. -disponible en ligne-).

(T) : Résultats de thèses.

Contenu : b) 1970.

- (T) *Calcul rapide d'une transformée de Fourier* (A. Eberhard, Grenoble).
- (T) Estimations d'erreur *backward*.
- Accélération de la convergence (ε et ρ algorithmes).
- (T) Algorithme de Strassen (C. de Polignac, Grenoble). (thèse :*Méthodes optimales de calcul de produits de matrices*, 1970. -disponible en ligne-).
- ... et quelques équations intégrales, mais pas d'EDOs !

(T) : Résultats de thèses.

Style de programmation, implantation

- Styles : tous à peu près identiques.
- Implantation :
  La machine utilisée est rarement citée.
  Peu d'exemples numériques (rien de systématique).

1) La méthode de Bairstow (Sir Leonard Bairstow, 1880–1963). Tome 1.

Programme codé par Lagouanelle (Toulouse).

1) La méthode de Bairstow (Sir Leonard Bairstow, 1880–1963). Tome 1.

Programme codé par Lagouanelle (Toulouse).

Trouver les racines réelles et complexes d'un polynôme à coefficients réels *sans utiliser d'arithmétique complexe.*

1) La méthode de Bairstow (Sir Leonard Bairstow, 1880–1963). Tome 1.

Programme codé par Lagouanelle (Toulouse).

Trouver les racines réelles et complexes d'un polynôme à coefficients réels *sans utiliser d'arithmétique complexe.*

On cherche un facteur quadratique d'un polynôme $P(x)$ dont on veut calculer les racines.
Étant donnés B et C réels, on a :

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

ce qui définit

$$\mathcal{F} : \begin{pmatrix} B \\ C \end{pmatrix} \mapsto \begin{pmatrix} R \\ S \end{pmatrix}.$$

On annule $\mathcal{F}$ par la méthode de Newton.

La méthode de Bairstow

Le calcul de la jacobienne de $\mathcal{F}$ est astucieux :

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

$$0 = \frac{\partial P}{\partial C} = (x^2 + Bx + C)\frac{\partial Q}{\partial C} + Q(x) + x\frac{\partial R}{\partial C} + \frac{\partial S}{\partial C}.$$

$$-Q(x) = (x^2 + Bx + C)\frac{\partial Q}{\partial C} + x\frac{\partial R}{\partial C} + \frac{\partial S}{\partial C}.$$

$$-xQ(x) = (x^2 + Bx + C)\frac{\partial Q}{\partial B} + x\frac{\partial R}{\partial B} + \frac{\partial S}{\partial B}.$$

Le code extrait des procédures

```
PROCEDURE BAIRSTOW(N,A,SO,PO,EPS1,EPS2,MAXITER,X,Y,DIVERG) ;
          VALEUR N,A ; ENTIER N,MAXITER ; REEL SO,PO,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
```

```
TABLEAU A,LE TEST D'ARRET POUR UN COUPLE DE RACINES EST
ABS(S(K+1)-S(K))'INFER'EPS1'ET'ABS(P(K+1)-P(K))'INFER'EPS1.SI
APRES MAXITER ITERATIONS CE TEST N'A PAS JOUE,ON FAIT EPS1=
10*EPS1 TANT QUE EPS1'INFEG'EPS2.
EN CAS DE NON CONVERGENCE ON SORT A DIVERG (ETIQUETTE QUI DOIT
FIGURER DANS LE PROGRAMME]SO ET PO SONT LES VALEURS INITIALES
DE S ET P,ON LES CHOISIRA,DE PREFERENCE,DE FAIBLE MODULE.IL EST
INUTILE DE PRENDRE MAXITER 'SUPER'50.POUR EPS1,ON PRENDRA LA PRE-
CISION DE LA MACHINE UTILISEE.LA SOLUTION EST RANGEE DANS LES
TABLEAUX X ET Y QUI CONTIENNENT RESPECTIVEMENT LES PARTIES REELLE
ET IMAGINAIRE ;
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 ;
         SI NITER > MAXITER ALORS DEBUT
          EPS1:=10*EPS1 ;
         SI EPS1 > EPS2 ALORS
         ALLERA  DIVERG ;
          NITER :=1 FIN ;
         POUR I:=0 PAS 1 JUSQUA N FAIRE
         DEBUT B[I]:=A[I]+S*B[I-1]-P*B[I-2] ;
               C[I]:=B[I]+S*C[I-1]-P*C[I-2] ;
         FIN ;
         C[N-1]:=C[N-1]-B[N-1] ;
         S1:=B[N]*C[N-3]-B[N-1]*C[N-2] ;
         P1:=B[N]*C[N-2]-B[N-1]*C[N-1] ;
```

```
        SI N=1 ALORS ALLERA  DERZERO ;
        SI N=0 ALORS ALLERA  FIN ;
SUITE : POUR K:=0 PAS 1 JUSQUA N FAIRE
        A[K]:=B[K] ;
        ALLERA  DEGTEST ;
DERZERO : X1:=-B[1]/B[0] ;
          X[N]:=X1 ; Y[N]:=0 ;
FIN   : FIN BAIRSTOW ;
```

3 - EXEMPLE D'UTILISATION

On se propose de calculer par Bairstow les racines du polynôme

$$P_4(X) = X \uparrow 4 - 2X \uparrow 3 + X \uparrow 2 + 2X - 2$$

on pose MAXITER = 50, SO = 0, PO = 0, EPS1 = 0.0000001

EPS2 = 0.01 .

Programme

```
DEBUT
PROCEDURE BAIRSTOW(N,A,SO,PO,EPS1,EPS2,MAXITER,X,Y,DIVERG) ; CORPS
                    DE PROCEDURE ;
ENTIER I,N,MAXITER; REEL SO,PO,EPS1,EPS2 ;
                  LIRE(N,MAXITER,SO,PO,EPS1,EPS2) ;
DEBUT TABLEAU X,Y[1:N],A[0:N] ;
       POUR I := 0 PAS 1 JUSQUA N FAIRE LIRE(A[I]) ;
       BAIRSTOW(N,A,SO,PO,EPS1,EPS2,MAXITER,X,Y,DIVERG) ;
POUR I := 1 PAS 1 JUSQUA N FAIRE ECRIRE(I,X[I],Y[I]) ;
ALLERA  FINPROG ; DIVERG : ECRIRE( " DIVERGENCE " , " BAIRSTOW " ) ;
FINPROG : FIN FIN ;
```

Rejouer

- OCR et beaucoup de patience.
- Le problème des mots clés en français : modifier le compilateur ou écrire un traducteur (Python, facile).
- E/S. non normalisées : pas grave.

Que dire de ce programme ?

Que dire de ce programme ?

Il est probablement juste.

- Méthode de Bairstow en Julia.
- Résultats certifiés à l'aide de `sagemath` (polynômes à coefficients entiers : calculs exacts dans les nombres algébriques + arithmétique d'intervalle).

Le style :

- 14 `goto`.
- Pas de découpage en sous procédures (et donc pas de procédures imbriquées).

Ce n'est pas le programme le mieux codé !

Le style :

- 14 `goto`.
- Pas de découpage en sous procédures (et donc pas de procédures imbriquées).

Ce n'est pas le programme le mieux codé !

On peut ré-écrire le code avec un seul goto (Algol 60 n'a pas de *break* pour sortir des boucles).

Il est intéressant de remarquer que le code Julia se transcrit facilement en Algol 60, en gardant la structure.

```
integer procedure bairstow(poly,lp,a,b,iterm,eps,
   ......
begin
    boolean procedure Newton(p,a,b,degre,iterm,eps
    ......
    begin
        real procedure NewtonStep(p,lp,a,b,iterm);
        ....
        begin
            ......
        end NewtonStep;
        .....
        begin
          resid := NewtonStep(p,degre+1,a,b,iterm);
          if sqrt(resid)<eps then
             begin
                ok := true;    goto outloop;
             end;
      end;
outloop;
```

Tout est disponible sur github :

`https://github.com/Thierry-Dumont/Vintage_Computing`

... et sera donc conservé *pour l'éternité* par *Software Heritage :*
`https://www.softwareheritage.org/?lang=fr`.

Note : on pourrait s'intéresser à :
« Le Défi Décennal de Reproductibilité est une invitation pour les chercheurs à essayer d'exécuter le code qu'ils ont créé pour une publication scientifique qui a plus de dix ans. »
`http://rescience.github.io/ten-years/`

Rejouer de vieux programmes, qu'est ce que ça peut nous apprendre ?

Une enquête numérique.

Quelle était la machine utilisé à Grenoble pour le calcul juste avant 1967 ?

L'IBM 360 est arrivé en 1967.
L'association ACONIT ne sait pas y répondre, les chercheurs en poste à l'époque non plus.
On peut tenter de résoudre le problème *par le calcul.*

Page 280 du tome 1 des *Procédures Algol en Analyse Numérique* : ajustement polynomial aux moindres carrés, par André Eberhard (auteur aussi d'une remarquable transformée de Fourier Rapide).

Page 280 du tome 1 des *Procédures Algol en Analyse Numérique* : ajustement polynomial aux moindres carrés, par André Eberhard (auteur aussi d'une remarquable transformée de Fourier Rapide).

Méthode naïve : ajuster les coefficients a de

$$p(x) = \sum_{i=0}^{m} a_i x^i.$$

c'est à dire minimiser

$$\min_{\{a\}} \sum_{i=1}^{n} (p(x_i) - y_i)^2.$$

On considère la matrice $B_{ij} = x_i^j$.
Les coefficients sont solution de :

$$ {}^tB\, B\ a = {}^tB\ \ y. $$

On considère la matrice $B_{ij} = x_i^j$.
Les coefficients sont solution de :

$$ {}^tB\,B\;a = {}^tB\;\;y.$$

Méthode très instable (la matrice est un avatar de la matrice de Hilbert).

La bonne méthode implantée aussi dans les *Procédures Algol* utilise des polynômes orthogonaux.

La résolution du système linéaire est obtenue en utilisant la routine GRESOLSYSLIN (Tome 1, N. Gastinel).

Élimination de Gauss sans choix du pivot maximum => Méthode instable.

La résolution du système linéaire est obtenue en utilisant la routine GRESOLSYSLIN (Tome 1, N. Gastinel).

Élimination de Gauss sans choix du pivot maximum => Méthode instable.

Tous ces défauts vont nous servir...

Algol n'a qu'un type **real**.

Quel était donc la précision des flottants utilisés pour l'ajustement polynomial aux moindres carrés ?

Le problème résolu dans le livre

- x : 21 points équirépartis sur $[0, 1]$
- $y = \exp(x)$.
- On ajuste un polynôme de degré $m = 4$.

Le problème résolu dans le livre

- x : 21 points équirépartis sur $[0, 1]$
- $y = \exp(x)$.
- On ajuste un polynôme de degré $m = 4$.

L'auteur trouve les valeurs a_i, $i = 0, 4$:

0.10003376×10, 0.99835205, 0.50927734, 0.14135742, $0.69702148 \times 10^{-1}$.

Expertise avec SageMath

SageMath permet de calculer avec des nombres flottants de précision quelconque :

```
sage: RealNumbers = RealField(20)
sage: RealNumbers
Real Field with 20 bits of precision
```

Le type double (du C) est donné par `RealField(53)`.
(SageMath utilise la bibliothèque GNU MPFR).

L'expérience

On reproduit le code d'André Eberhard le plus fidèlement possible (y compris la routine de résolution de système linéaire) dans SageMath, mais paramétré par la précision des flottants.

SageMath, c'est du Python.
Les **real** du code Algol sont remplacés par des nombres déclarés comme appartenant à **RealField(pres)**.

On peut calculer que les erreurs sur la matrice et sur les y sont multipliées par 689500 (conditionnement de la matrice).

On peut calculer que les erreurs sur la matrice et sur les y sont multipliées par 689500 (conditionnement de la matrice).

On en déduit que si on calcule avec une précision de 350 bits, (`RealNumbers = RealField(350)`), on aura une erreur inférieure à 10^{-100} sur la solution (sur les coefficients ajustés a_i).
Cela fournit une solution de référence *presque exacte*.

On compare alors la solution donnée dans le livre à la solution de référence :

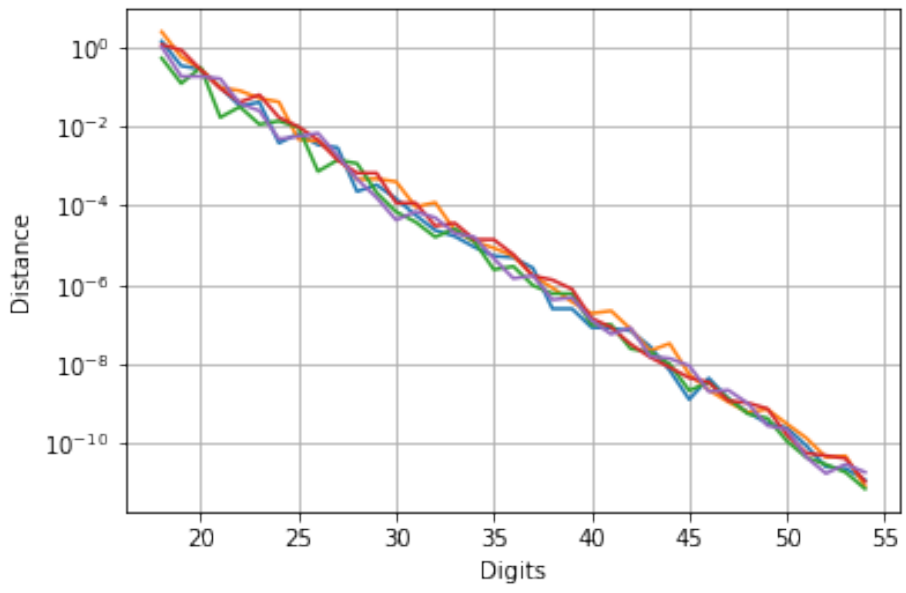
- Référence (a^{ref}) : 1.0000309,0.99863862,0.51016735,0.13987017,0.069541564
- Livre (a) : 1.0003376,0.99835205,0.50927734, 0.14135742, 0.069702148.

On compare alors la solution donnée dans le livre à la solution de référence :

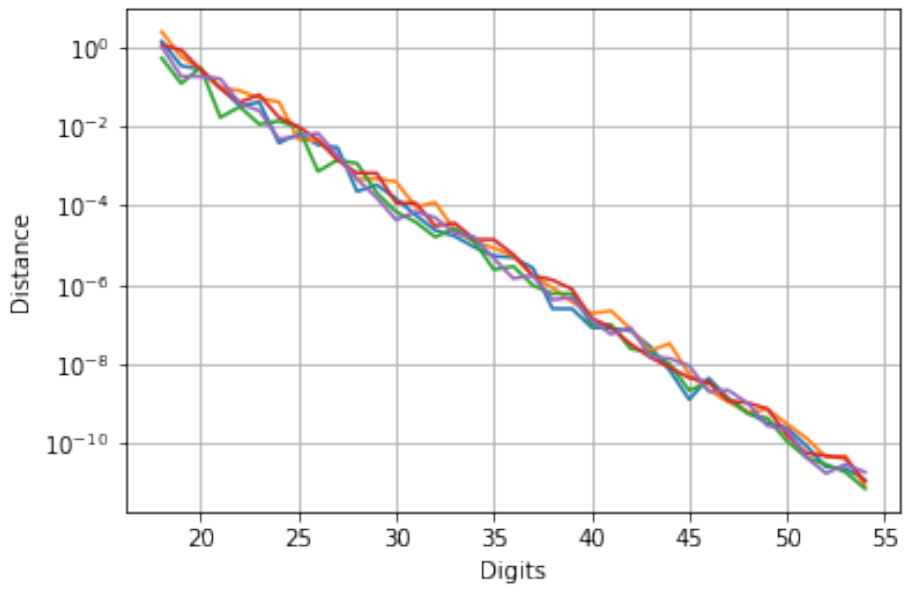
- Référence (a^{ref}) : 1.0000309,0.99863862,0.51016735,0.13987017,0.069541564
- Livre (a) : 1.0003376,0.99835205,0.50927734, 0.14135742, 0.069702148.

et ont définit la distance entre les deux comme :
$||a - a^{ref}|| = \max_i |a_i - a_i^{ref}|)$

On fait le même calcul (distance entre la solution calculée et la solution de référence) en faisant varier varier la précision :

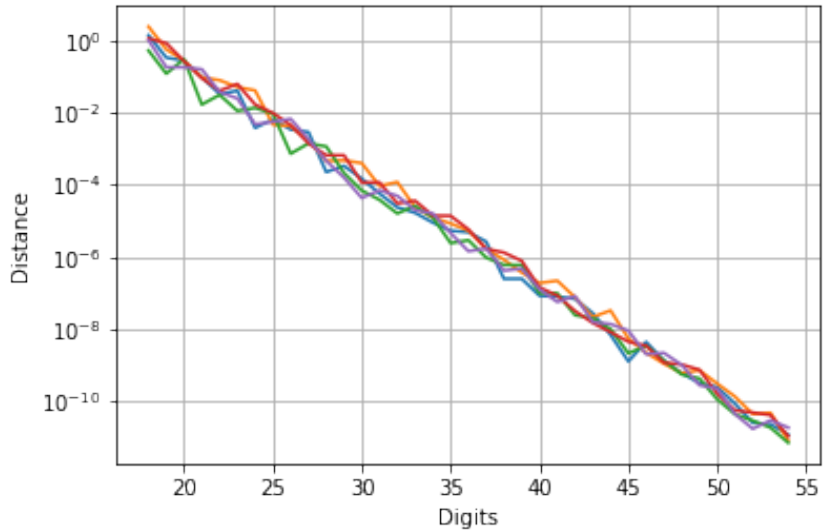


On fait le même calcul (distance entre la solution calculée et la solution de référence) en faisant varier varier la précision :



En reportant la distance entre la solution du livre et la solution exacte, on va trouver la précision de la machine utilisée.

Distance (référence,livre)= 1.48724517163778e-3
Précision donnant le résultat le plus proche 27 bits.



Précision	26 bits	27 bits	28 bits
RNDN	3.3314164e-3	2.843135192e-3	2.2173732e-4
RNDD	4.1279873e-3	1.686581078e-3	4.4976756e-4
RNDZ	7.1382152e-4	1.378291446e-3	1.13415082e-3
RNDU	4.4330444e-3	1.378291446e-3	6.4586957e-4
RNDA	6.5693935e-3	1.795536675e-3	4.5276324e-4

Il ne semble pas qu'il ait existé de machine avec 26 ou 28 bits de précision. Avec les "doubles" actuels la distance serait de $2,3\ 10^{-11}$.

Mais alors, c'était quoi cette machine ?

Candidates :

- chez DEC, les PDP 6,. . . ,10.
- chez IBM, la Série 7000.
- des machines chez UNIVAC.
- et peut-être d'autres.
- DEC PDP : *minis.* Ne correspondent pas à un laboratoire prestigieux.
  UNIVAC : absent de la recherche française.

Mais alors, c'était quoi cette machine ?

Candidates :

- chez DEC, les PDP 6,. . . ,10.
- chez IBM, la Série 7000.
- des machines chez UNIVAC.
- et peut-être d'autres.
- DEC PDP : *minis.* Ne correspondent pas à un laboratoire prestigieux.
  UNIVAC : absent de la recherche française.
- IBM, *élémentaire mon cher Watson !* il existait un *Centre de Recherche IBM* à Grenoble qui a beaucoup collaboré avec l'IMAG.

L'ordinateur était donc probablement un IBM série 7000. Finalement Jean-François Maître (Pr. Émérite, École Centrale Lyon) a confirmé par un article de la presse locale que c'était un **IBM 7044**.

L'ordinateur était donc probablement un IBM série 7000. Finalement Jean-François Maître (Pr. Émérite, École Centrale Lyon) a confirmé par un article de la presse locale que c'était un **IBM 7044**.

Un célèbre IBM 7044

Les IBM série 7000

- mots de 36 bits.
- RAM *jusqu'à* 32 k mots.
- simple précision : mantisse de 27 bits.
- **double précision** : mantisse de 54 bits.

Les IBM série 7000

- mots de 36 bits.
- RAM *jusqu'à* 32 k mots.
- simple précision : mantisse de 27 bits.
- **double précision** : mantisse de 54 bits.

Algol 60 n'avait qu'un type **real**, mais pourquoi pas une option de compilation pour passer automatiquement les **real** en **double précision** (comme chez CII) ?

« In 1963, IBM introduced lower cost machines with a similar architecture, but fewer instructions and simplified I/O, called the IBM 7040 and 7044 ».

...parmi les instructions manquantes : la double précision (54 bits de mantisse).

« In 1963, IBM introduced lower cost machines with a similar architecture, but fewer instructions and simplified I/O, called the IBM 7040 and 7044 ».
...parmi les instructions manquantes : la double précision (54 bits de mantisse).

Grenoble :

Manque d'argent ou bien considérait-on la double précision comme pas nécessaire ?

Noël Arnaud (1919-2003),

- Pataphysicien (Conférent Majeur de l'Ordre de la Grande Gidouille),
- Président de l'Oulipo, de la Société des Amis d'Alfred Jarry,
- Co-fondateur de l'Internationale Situationiste,
- Fondateur de l'ouvroir de Cuisine Potentielle, etc...

et auteur, entre autres de *Poèmes algol* :

Noël Arnaud (1919-2003),

- Pataphysicien (Conférent Majeur de l'Ordre de la Grande Gidouille),
- Président de l'Oulipo, de la Société des Amis d'Alfred Jarry,
- Co-fondateur de l'Internationale Situationiste,
- Fondateur de l'ouvroir de Cuisine Potentielle, etc...

et auteur, entre autres de *Poèmes algol* :

ALGORICHE

Si pour faire fin
faut faire faux
pour faire faux
faut faire fin.