LES CALCULATEURS ELECTRONIQUES ET LE DEVELOPPEMENT DES MATHEMATIQUES NUMERIQUES

par François GENUYS
(*Compagnie I.B.M.-France,* Paris)

Le développement actuel des Mathématiques numériques n'a été possible qu'avec le développement des moyens de calcul. Le calcul à la main, avec ou sans machine de bureau, tend à être remplacé par le calcul automatique. Deux types de machines le réalisent :

a) *Les calculateurs analogiques :* les divers nombres considérés sont représentés par des grandeurs physiques (courant électrique, niveau d'eau...), et l'on utilise des lois physiques pour simuler les calculs à effectuer.

b) *Les calculateurs digitaux :* ceux-ci travaillent sur des nombres de développement limité dans une certaine base de numération.

Je donnerai d'abord quelques indications sur les calculateurs digitaux. Les divers nombres cités correspondent aux performances atteintes par les machines les plus puissantes.

A. — DESCRIPTION D'UN CALCULATEUR DIGITAL

1° Codage de l'information :

Les constructeurs n'utilisent actuellement pour coder l'information que des dispositifs binaires, c'est-à-dire à deux états (courant ou pas courant, sens d'une magnétisation, présence ou absence de trou dans une carte), ceci pour des raisons essentiellement technologiques. Je désignerai par position binaire ou par bit (abréviation de l'anglais binary digit) tout atome d'information à deux états ; les deux états seront notés 0 et 1.

Au moyen de ces organes élémentaires, il faut pouvoir coder les nombres (les facteurs) sur lesquels opère la machine et les opérations (les instructions) que celle-ci devra exécuter. Souvent, ce codage est redondant, de manière à dépister des erreurs éventuelles : par exemple, le nombre de 1 dans le codage est pair ou, encore, un nombre est toujours accompagné de son reste modulo 7.

2° Représentation des nombres :

Si le système binaire est utilisé, il est naturel de représenter un chiffre au moyen d'un bit. En revanche, si l'on utilise le système décimal,

il faudra au moins 4 bits pour représenter un chiffre, mais, puisque l'on peut former $2^4 = 16$ combinaisons avec 4 bits, cette représentation est redondante (certaines combinaisons seront à exclure). On utilise de nombreux codes ; la remarque que $C_5^2 = 10$ a conduit à utiliser le code 5 dont 2 : code à cinq bits dont deux, et deux seulement, sont égaux à 1 ; par exemple :

Chiffre	Code
0	0 0 0 1 1
1	0 0 1 0 1
2	0 0 1 1 0
3	0 1 0 0 1
4	0 1 0 1 0
5	0 1 1 0 0
6	1 0 0 0 1
7	1 0 0 1 0
8	1 0 1 0 0
9	1 1 0 0 0

Le signe d'un nombre nécessite un bit. Pour coder un caractère alphabétique, il faut au moins 5 bits (32 possibilités).

Le codage d'un nombre sera la réunion des codages des chiffres formant ce nombre. On le fait soit en *virgule fixe*, soit en *virgule flottante*. En virgule fixe, les facteurs sont des nombres entiers ; l'utilisateur peut évidemment imaginer qu'une virgule est placée entre deux des chiffres, mais, celle-ci n'étant pas codée, il devra suivre avec soin la position de la virgule dans les diverses opérations. En virgule flottante, en revanche, les nombres sont mis la sous la forme :

$$B^n f \quad \text{avec} \quad B^{-1} \leqslant f < 1$$

où B est la base de numération ; n s'appelle l'exposant et f la partie fractionnaire ; le codage d'un nombre comprend alors son exposant et sa partie fractionnaire, sur lesquels le calculateur opère automatiquement et simultanément au moyen d'instructions convenables. Exemple :

$$0{,}483 \times 10^{-2} - 0{,}472 \times 10^{-2} = 0{,}11 \times 10^{-3},$$
$$\text{ou } (-2{,}483) - (-2{,}472) = (-3{,}11).$$

La virgule flottante est d'un usage plus commode, mais elle est plus lente, plus onéreuse de réalisation, et nécessite plus de bits pour un nombre donné de chiffres significatifs ; son usage tend cependant à se généraliser.

Les calculs d'erreur sont différents suivant le mode employé ; ce sont souvent eux qui guident le choix.

3° Schéma général :

Un calculateur comporte :

a) *Une unité centrale*, qui contrôle le déroulement du « programme » et exécute les instructions ; elle comprend un certain nombre de registres où sont envoyés les résultats des opérations, où sont signalées cer-

taines anomalies telles que dépassement de capacité (trop de chiffres dans un résultat), etc...

b) *Des unités de mémoire,* qui emmagasinent une très grande quantité d'informations ; le calculateur peut « écrire » dans une mémoire et la lire sans intervention d'un opérateur humain.

c) *Des unités d'entrée et de sortie,* qui permettent au calculateur de recevoir de l'information extérieure (programme, données numériques), ou inversement de fournir de l'information à l'extérieur (résultats, renseignements permettant de reprendre un calcul interrompu).

Pour permettre au calculateur l'utilisation de la mémoire, on divise celle-ci en *mots* d'un nombre fixe de bits (par exemple 40) et l'on affecte à chaque mot un nombre que l'on appelle son *adresse*. On fait tenir dans un mot un nombre ou une instruction.

Le schéma de fonctionnement est alors le suivant. Lorsque le calculateur vient d'exécuter l'instruction d'adresse α, il lui faut connaître l'adresse β de l'instruction suivante. On a adopté deux solutions :

a) β est indiqué dans l'instruction α.

b) $\beta = \alpha + 1$; on dit alors que la machine est *séquentielle.*

Mais, même dans une machine séquentielle, on doit pouvoir changer l'ordre d'exécution des instructions suivant le résultat de tests ; la solution *a* est employée alors pour certaines instructions (de *saut* ou de *transfert*).

Les premières instructions (à exécuter) d'un programme sont en général toujours les mêmes et sont conçues pour lire le programme (c'est-à-dire le mettre en mémoire) ; elles sont exécutées dès que l'on appuie sur le bouton « Départ ». A partir de ce moment, il n'y a plus besoin d'opérateur.

4° Description des instructions :

L'ensemble des bits codant une instruction est décomposé en deux :

a) une partie sert à désigner la fonction de l'instruction (il peut y avoir plus d'une centaine d'instructions différentes) ;

b) d'autres parties indiquent les adresses des facteurs sur lesquels opèrent l'instruction et (ou) l'instruction à exécuter immédiatement après.

Selon leur nature, les instructions se répartissent en :

a) *Instructions arithmétiques* (binaires ou décimales suivant le calculateur) ; ce sera, par exemple, l'addition en virgule flottante d'un registre et d'une mémoire (résultat dans le registre). Les temps d'exécution peuvent atteindre quelques microsecondes.

b) *Instructions logiques :* exécution d'opérations logiques bit à bit, telles que *et, ou, non,* $=>$, etc...

c) *Instructions de test :* leur rôle est de faire prendre des décisions au calculateur ; rechercher l'instruction suivante en α, β ou γ selon qu'un registre est négatif, nul, ou positif, etc...

d) *Instructions d'entrée-sortie :* ce sont celles qui relient le calculateur avec le monde extérieur : lecture d'une carte, etc...

5° Les mémoires :

D'une mémoire, il faut surtout retenir la *capacité* et le *temps d'accès*. Ce dernier est la durée qui sépare l'instant où un mot est demandé (par le calculateur) et celui où le mot est effectivement disponible pour le calcul.

a) *Mémoires à ferrites :* de petits anneaux de ferrites à deux états d'aimantation représentant un bit ; capacité de l'ordre de 10^6 bits ; temps d'accès de quelques microsecondes.

b) *Tambours magnétiques :* ce sont des tambours recouverts d'oxyde magnétique tournant à grande vitesse ; capacité de l'ordre de 2.10^5 bits ; temps d'accès variable dépendant de la position du tambour quand un mot est demandé ; temps moyen de quelques millisecondes.

c) *Bandes magnétiques :* ce sont des bandes similaires à celles utilisées en enregistrement sonore, capacité de 10^7 bits ; le temps d'accès est variable, peut aller jusqu'à 3 minutes. Pratiquement, on classe l'information sur une bande, de manière à éviter les sauts d'un bout à l'autre de celle-ci.

d) *Disques magnétiques :* ce sont des disques empilés recouverts d'oxyde magnétique. La capacité est de 10^7 bits et le temps d'accès moyen de 0,5 seconde.

Le mathématicien doit prévoir une organisation judicieuse de son calcul, de manière à éviter les temps d'attente d'information.

6° Unités d'entrée-sortie :

a) *Carte perforée :* une carte contient $12 \times 80 = 960$ positions perforables, donc 960 bits, en général utilisés avec grande redondance (une colonne de 12 bits pour un chiffre ou signe). La vitesse de lecture ou de perforation est de l'ordre de 4 000 bits/seconde.

b) *Ruban perforé :* vitesse de 2 400 bits/seconde.

c) *Bande magnétique :* outre leur fonction de mémoire, les bandes magnétiques peuvent servir d'entrée ou de sortie à une vitesse de 10^5 bits/seconde.

d) *Impression :* 500 lignes/minute ; directement consultable par l'homme, mais non exploitable postérieurement par une machine.

e) *Entrée-sortie analogique :* on a enfin imaginé des dispositifs permettant d'utiliser directement les résultats de mesure ou de fournir à l'utilisateur les résultats de calcul sous forme de courbe.

B. — Utilisation d'un calculateur

Je n'étudierai que l'utilisation en calcul scientifique, laissant de côté les applications considérables dans le domaine de la gestion.

1° L'organigramme :

Un calcul à effectuer se compose d'une suite (dans le temps) d'opérations et de tests ; avant de procéder au codage, il faut faire un schéma détaillé de ceux-ci. C'est l'organigramme. Par exemple, si l'on veut résoudre $ax^2 + bx + c = 0$, a, b et c étant lus par carte, l'organigramme sera le suivant :

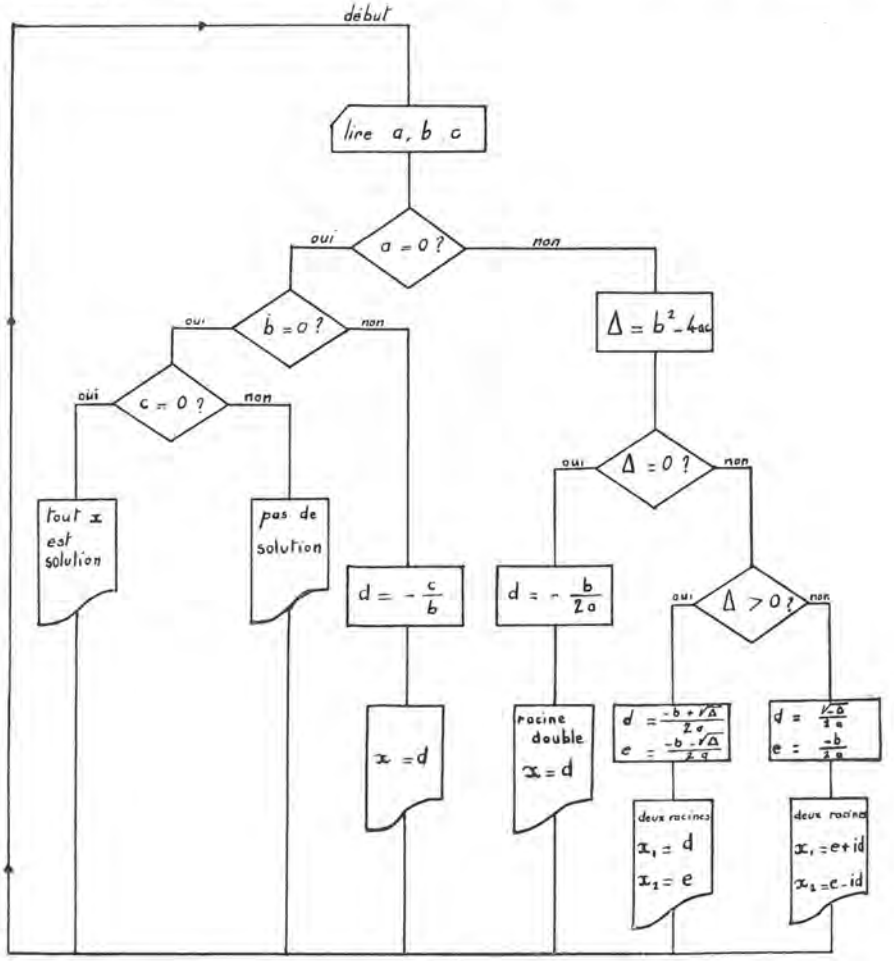

FIG. 2

Ici, la résolution d'une équation est suivie de la lecture de nouvelles données.

Cet organigramme n'est d'ailleurs pas le meilleur ; par exemple, il y aurait intérêt à calculer $-\frac{b}{2a}$ avant de faire le test $\Delta > 0$, ceci afin de diminuer le nombre d'instructions.

Je crois que, même au niveau scolaire, la confection d'organigramme est un exercice excellent dans toutes les discussions de calcul ; il oblige

à étudier tous les cas et ne permet pas de laisser dans l'ombre le point délicat.

Remarquons enfin que l'organigramme doit prévoir également l'utilisation rationnelle des mémoires ; on peut très bien ainsi affecter de la même adresse des quantités différentes, à la condition que l'on n'ait plus besoin de la première quantité au moment où la seconde est introduite. C'est là une différence avec le calcul à main où l'on n'est pas limité par le nombre des symboles.

2° Les sous-programmes :

Dans le programme ci-dessus, nous voyons apparaître une racine carrée, qui n'est pas généralement une instruction. Il faut donc écrire une suite d'instructions effectuant la racine carrée ; il faudra d'ailleurs écrire cette suite en tous les points où l'on calcule cette racine carrée. Pour économiser du temps et des mémoires, on réalisera une fois pour toutes un *sous-programme* de racine carrée étudié avec soin et qui ne se trouvera qu'en un seul emplacement du programme principal (ici, résolution de l'équation du second degré). Le programme se branchera sur ce sous-programme toutes les fois qu'une racine devra être calculée ; le retour se fera (par des techniques appropriées) au point où l'on aura quitté le programme principal.

Les divers sous-programmes mis au point sur un calculateur sont réunis en une bibliothèque (sur bande ou sur carte perforée) ; celle-ci contient des sous-programmes mathématiques (calcul de fonction, méthodes d'intégration, etc...), ou non-mathématiques (tri, conversion binaire-décimal, lecture de cartes, etc...).

Enfin, au moyen de sous-programmes appropriés, le programmeur peut réaliser toute opération non prévue par le constructeur (calcul sur le corps des complexes, sur des nombres comportant plus de décimales que le nombre standard, etc...). L'évolution des machines conduit souvent à réaliser comme nouvelles instructions des sous-programmes d'usage courant ; par exemple, la virgule flottante est apparue d'abord sous forme de sous-programme.

3° La programmation automatique :

L'organigramme étant écrit, il faut alors coder le problème, c'est-à-dire remplacer chaque case de l'organigramme par la séquence d'instructions convenable, affecter aux instructions et aux variables des adresses et, par exemple, perforer des cartes pour l'introduction dans la machine.

Cette partie du travail est très fastidieuse, d'où l'idée de la faire exécuter par le calculateur. Sous sa forme la plus évoluée, la programmation automatique prend la forme suivante :

a) Le « programmeur » écrit le programme sous forme d'une suite d'instructions écrites suivant un code se rapprochant de l'écriture mathématique (ou usuelle pour les décisions logiques) ; par exemple :

X = (AXC I − A I XC) / (AXB I − A I XB)
ALLER EN I SI (X > 0 .

Ces instructions sont représentées par une suite linéaire de symboles (nécessité technique).

b) Les instructions sont transcrites caractère à caractère sur un support lisible par la machine (carte, bande, etc...).

c) La machine, ayant lu le programme ainsi écrit, utilisera le « programme à programme » (programme d'assemblage) et créera un nouveau programme, dans le langage propre de la machine, qu'il suffira alors d'utiliser. Ces programmes d'assemblage sont, on le conçoit, d'une grande complexité.

Je signale enfin que les constructeurs tentent de se mettre d'accord sur une symbolisation unique appelée ALGOL (au niveau *a*). Ainsi, un programme donné serait utilisable par toutes les machines (après assemblage convenable).

C. — Les Mathématiques numériques

Je vais maintenant sur quelques exemples essayer de faire ressortir les particularités du calcul sur machine.

1° Calcul d'une fonction :

Lorsqu'un programme utilise une fonction f, le sous-programme de calcul de f peut utiliser :

a) une table de f calculée par la machine ou introduite directement (à partir de cartes par exemple), sur laquelle une interpolation est faite si les valeurs ne sont pas assez rapprochées, ou

b) une formule que l'on sait représenter assez bien f dans l'intervalle considéré.

On envisage souvent des solutions intermédiaires ; dans le cas *b* par exemple, on peut utiliser des formules différentes suivant l'intervalle où se trouve x.

La solution *a* est plus rapide, mais nécessite une plus grande place en mémoire. Dans un programme très important, on penchera pour la solution *b*.

Soit à calculer par exemple e^x :

$$e^x = 2^p (e^z)^q$$

$$z = \frac{\operatorname{Log} 2}{2}\left\{\frac{x}{\operatorname{Log} 2}\right\};\ p = \left[\frac{x}{\operatorname{Log} 2}\right],\ \text{si}\ 0 \leqslant \left\{\frac{x}{\operatorname{Log} 2}\right\} \leqslant \frac{1}{2}$$

$$z = \frac{\operatorname{Log} 2}{2}\left(\left\{\frac{x}{\operatorname{Log} 2}\right\} - 1\right);\ p = \left[\frac{x}{\operatorname{Log} 2}\right] + 1,\ \text{si}\ \frac{1}{2} \leqslant \left\{\frac{x}{\operatorname{Log} 2}\right\} < 1$$

Remarquons que $|z| \leqslant \dfrac{\operatorname{Log} 2}{4}$. Pour calculer e^z, on utilise :

$$e^z \sim \sum_0^n \frac{z^k}{k!},$$

le programme déterminant le nombre de termes à utiliser.

Remarquons que, sur une machine ne connaissant que les quatre opérations, on ne peut calculer que des fractions rationnelles. C'est ce qui conduit à rechercher des approximations rationnelles ou polynômiales des fonctions. Mais, plutôt que d'utiliser, ainsi que nous venons de le faire, des formules d'origine théorique, on préfère rechercher numériquement les coefficients inconnus de la fonction approchée.

Le problème s'énonce ainsi dans le cas de l'approximation polynômiale : trouver N, $a_0, a_1, \ldots, a_N$ tels que :

$$|f(x) - \sum_0^N a_k x^k| \leqslant \varepsilon \qquad \text{pour } x \in I$$

et que N soit le plus petit possible (de manière que le calcul soit le plus rapide possible) ; accessoirement, on peut imposer que, pour le N ainsi déterminé, le maximum de l'erreur soit le plus petit possible. Ce problème d'optimisation se traite par des méthodes comparables à celles utilisées pour minimiser des dépenses dans les problèmes économiques. On peut utiliser la théorie de la programmation linéaire.

Il existe des programmes dont l'objet est précisément la recherche du polynôme optimum dans un certain intervalle. L'établissement d'un sous-programme donnant f se fait alors en deux temps :

a) faire un programme précis au besoin assez compliqué, calculant f (par exemple pour e^x le programme indiqué ci-dessus) ;

b) utiliser ce programme pour trouver le « meilleur » polynôme (ou fraction rationnelle) donnant f.

Ceci peut être long, mais est fait une fois pour toutes ; la suppression d'une seule opération dans un sous-programme justifie un calcul préalable, même long, sur machine.

2° Méthodes de Monte-Carlo :

Si l'on utilise la méthode des trapèzes avec 10 intervalles pour chaque variable, le calcul d'une intégrale multiple d'ordre p exige le calcul de 10^p valeurs de la fonction. Même si le calculateur est très rapide, le temps exigé devient facilement supérieur à un siècle. Nombreux sont d'ailleurs les problèmes d'apparence bénigne, surtout si des dénombrements interviennent, qu'il est vain actuellement d'essayer de résoudre numériquement.

Soit par exemple à calculer l'intégrale : $I = \int \ldots \int_V f(P)dv$.

Les méthodes classiques reviennent à calculer $\sum_1^N \alpha_i f(P_i)$, les α_i et les P_i étant choisis suivant une loi bien déterminée (dans les trapèzes, les P_i sont les sommets d'un réseau de points).

Dans une méthode de Monte-Carlo, on considérera la variable aléatoire $f(P)$, P étant un point aléatoire de distribution uniforme dans V, son espérance E est $\dfrac{I}{\int \ldots \int_V dv}$; si l'on connaît le volume de V, on fait un nombre suffisant de « tirages » de P, la moyenne des $f(P)$ obtenus est, avec une certaine probabilité p, comprise entre $E + \varepsilon$ et $E - \varepsilon$; si la probabilité est assez grande, on admet que E est égale à cette moyenne à ε près.

D'une manière générale, lorsque l'on désire un nombre a, l'on recherche une variable aléatoire X dont l'espérance est égale à a, et l'on fait un nombre suffisant de tirages pour l'estimation de a.

Souvent avec dix mille tirages l'on peut espérer trois chiffres significatifs ; on a donc réduit le calcul d'intégrale indiqué au début à une taille raisonnable.

Remarquons que l'on doit disposer en machine d'un programme à tirer des échantillons d'une variable aléatoire. Deux solutions ont été retenues :

a) un phénomène physique, type bruit de fond, fournit des « nombres au hasard » qui sont enregistrés et fournis le cas échéant à la machine ;

b) un sous-programme calcule des pseudo-nombres au hasard, utilisables, comme des séries de tests le montrent, pour ce type de calcul. Une loi fréquemment utilisée est :

$$u_{n+1} = ku_n \quad \text{mod } p.$$

Un des premiers calculs de Monte-Carlo (avant la lettre) a été l'estimation de π par le jet d'une aiguille.

L'idée est toujours de trouver un modèle probabiliste à un problème mathématique et de simuler ledit modèle. Quelquefois même, ce mode de résolution revient à simuler le phénomène physique de nature probabiliste (mouvement de particules, etc...), ou un phénomène simplifié, qui a donné naissance aux équations à résoudre. Ici, la mise en équation du physicien n'aura été d'aucun secours pour la résolution numérique.

Remarquons que la confiance dans le résultat obtenu dépend de la variance de la variable considérée. Faute de mieux, cette variance est souvent elle-même estimée expérimentalement.

2° Résolution d'équations linéaires :

Pour résoudre un système d'équations linéaires d'ordre n, les formules utilisant des déterminants sont à proscrire (un déterminant d'ordre n est la somme de n !, termes eux-mêmes produits de n facteurs) ; la méthode la plus rapide en général, pour une précision donnée, est l'élimination dont de nombreuses variantes sont utilisées. On voit que les constantes occupent sensiblement n^2 mémoires, le temps de calcul varie comme n^3. Des considérations d'encombrement font utiliser d'autres méthodes, qui peuvent faire gagner du temps dans le cas où de nombreux coefficients sont nuls.

a) *Méthode de Monte-Carlo :* problème du parieur.

Soit deux parieurs A et B de richesses a et b, faisant une suite de paris de mise h ; à chaque pari, A a la probabilité p de gagner, et B la probabilité $q = 1 - p$. On recherche la probabilité que A ruine B (événement où le jeu s'arrête, de même que si B ruine A) ; nous supposons a et b multiples entiers de h. Nous représentons l'état du jeu à un certain instant par le schéma :

OE représentant la fortune actuelle de A et EF celle de B,

$$(OF = a + b = C^{te}).$$

On peut imaginer que E est une particule parcourant $+h$ avec la probabilité p et $-h$ avec la probabilité q ; on recherche alors la probabilité qu'a la particule d'atteindre F.

Soit $v_m(x)$ la probabilité que A ruine B en au plus m coups, la fortune de A étant x au départ ; ou A gagne le premier pari et ruine ensuite B en au plus $m-1$ coups, ou A le perd et ruine ensuite B en au plus $m-1$ coups, ce qui se traduit par :

$$(1) \qquad v_m(x) = pv_{m-1}(x+h) + qv_{m-1}(x-h) \qquad \begin{matrix} 0 < x < a+b \\ m \geqslant 1 \end{matrix}$$

d'autre part : $v_m(0) = 0 \qquad v_m(a+b) = 1.$

$$\begin{cases} v_0(x) = 0 & \text{si } 0 \leqslant x < a+b \\ v_0(x) = 1 & \text{si } x = a+b. \end{cases}$$

La relation (1) et $v_1(x) \geqslant v_0(x)$ entraînent par récurrence que

$$v_m(x) \geqslant v_{m-1}(x) \ ;$$

d'autre part $v_m(x) \leqslant 1$, donc $v_m(x)$ admet une limite $v(x)$ lorsque $m \to \infty$.

C'est cette limite que nous définirons comme probabilité que A ruine B en un nombre fini de paris ; $v(x)$ sera solution de :

$$\begin{cases} v(x) = pv(x+h) + qv(x-h) \\ v(0) = 0 \\ v(1) = 1. \end{cases}$$

Si l'on remarque que x prend un nombre fini de valeurs, on voit que nous avons un système, cramérien d'ailleurs, d'un nombre fini d'équations linéaires pour déterminer $v(x)$. On peut trouver la valeur explicite de $v(x)$ en utilisant les méthodes de résolution d'équations aux différences :

$$v(x) = \frac{1 - \left(\frac{p}{q}\right)^{-\frac{x}{h}}}{1 - \left(\frac{p}{q}\right)^{-\frac{a+b}{h}}}.$$

On peut voir également que $v(x) + u(x) = 1$, $u(x)$ étant la probabilité que B ruine A en un nombre fini de paris ; autrement dit, le jeu se termine presque sûrement en un nombre fini de paris.

On peut donc simuler le jeu sur machine et estimer par la méthode de Monte-Carlo ; on est « presque » assuré que chaque partie demande un nombre fini de coups, donc que le calcul lui-même est réalisable en un temps fini. Dans le cas particulier, le calcul ne présenterait pas d'intérêt pratique. En revanche, en remplaçant certaines équations différentielles (ou aux dérivées partielles) par des équations aux différences, on tombe sur des équations d'un type voisin de (2) en si grand nombre qu'il est raisonnable d'appliquer une méthode de Monte-Carlo.

b) *Méthodes itératives.*

Utilisant la notation matricielle, un système d'équations linéaires, que nous supposerons cramérien, peut se mettre d'un grand nombre de manières sous la forme :

$$(3) \qquad x = Ax + b.$$

Considérons la suite : x_0 quelconque,

$$x_{k+1} = Ax_k + b.$$

Si elle tend vers une limite α, α est la solution du système proposé. On montre que, pour que cette suite ait une limite, il faut et il suffit que la plus grande valeur propre de A soit en module inférieure à 1 et que la convergence est d'autant plus rapide que le module maximum est plus petit.

Cette méthode de résolution, bien que plus longue que l'élimination si A est une matrice pleine (pas de coefficient nul), est de programmation très simple et surtout nécessite beaucoup moins de mémoires si la matrice A est très creuse. Quand on l'utilise, on cherche à écrire l'équation (3) de manière que la convergence soit la plus rapide possible.

4° Conclusion :

Je n'ai pas voulu développer ici tous les domaines d'application du calcul numérique ; pratiquement, dès qu'un problème mathématique a pour solution un nombre fini de nombres, il n'est pas vain d'essayer d'en rechercher une solution approchée numérique. Remarquons que l'on se contente, dans un problème où la solution est un ensemble infini de nombres, de ne donner qu'une partie de la solution (par exemple, une fonction de deux variables sera calculée aux sommets d'un réseau).

En revanche, je voudrais, en conclusion de ces quelques exemples, dégager ces deux points fondamentaux :

1° Une solution explicite d'un problème est rarement à utiliser ; elle ramène le problème à d'autres (calcul d'intégrale, de fonctions spéciales, sommation d'une série), qui sont souvent, numériquement, d'un traitement plus difficile que le problème initial, ou beaucoup plus long. Ainsi, pour résoudre une équation du second degré, une méthode itérative peut souvent être préférée aux formules classiques. L'étude théorique d'un problème ne sert souvent qu'à interpréter certains accidents de calcul ; par exemple, si l'équation du second degré n'a pas de racines, on conçoit qu'une méthode itérative ne converge pas.

2° Les méthodes employées sur calculateurs rapides ne sont pas généralement une adaptation de ce qui est fait à la main. Lorsqu'un problème déjà résolu à la main est à résoudre sur machine, mais à une échelle beaucoup plus grande (exemple : système de mille équations linéaires à mille inconnues), le temps de calcul, ou l'encombrement, devient souvent prohibitif. On doit alors se tourner vers des méthodes dont le temps de calcul, ou l'encombrement, croît moins vite avec la taille du problème, et qui seront par conséquent, à partir d'une certaine taille, à préférer à la méthode initiale. Il faut également remarquer qu'avec des mémoires à bandes et, dans une certaine mesure, des mémoires à tambours, il y a intérêt à utiliser l'information en séquence sur la mémoire, ce qui peut donner la préférence à une méthode utilisant des nombres toujours dans le même ordre.