

Place d'APL dans le monde actuel et à l'avenir

par Michel Dumontier

On voit souvent APL conjugué au futur (titre du congrès APL85 : APL and the future, titre du congrès APL90 : For the future...), je suis tout à fait d'accord pour dire qu'APL a été, est et sera toujours un langage d'avenir, mais que dire dès maintenant ?

Lors d'une réunion AFAPL à Paris, j'avais promis d'écrire un jour un article sur les 'victoires d'APL'. Le moment est venu d'en parler, car à la dernière réunion de la norme ISO pour l'extension d'APL à Toronto, le 15 août avant le congrès APL97, nous avons appris que nous allons devoir produire un rapport dans lequel nous devons donner les raisons d'être d'APL, sa place dans le marché, son implantation dans le monde et dans quels domaines. Au passage, pour concrétiser cela, nous pourrions énumérer les conquêtes d'APL, ses particularités (vitesse de réalisation, excellent pour le prototypage, puissance pour faire des essais rapidement pour la recherche etc. Je me suis engagé à chercher pour fournir de tels renseignements en disant qu'un article dans la revue AFAPL allait être susceptible de nous apporter les informations souhaitées.

Pour faire notre rapport, nous devons nous appuyer sur un texte intitulé 'market requirements' pour le COBOL. (J'en vois qui rient déjà attendez un peu!)

Finalement je livre le texte tel quel sans le traduire, car il faudra donner un texte en anglais : voyons donc si on peut remplacer dans le texte qui suit, COBOL par APL. Même si ce n'est pas passionnant, cela pourra nous donner l'occasion de nous détendre un peu!

COBOL continues to be widely used for new development and for enhancement and re-engineering of existing applications. (Pensez au syndrome de l'an 2000 : j'ai lu qu'aux Etats-Unis on recherche des programmeurs COBOL retraités, (y aurait-il pénurie ?!), pour gérer la crise de ce fameux syndrome bien connu qui va nous éclater dans les mains bientôt !). Many factors drive the market for COBOL standardization:

1. Deployment of applications to workstations and distributed environments and growth of COBOL in Unix environments has generated requirements for new features in the language. (Il en a bien besoin !). These needs have been met by implementor extensions to the language, in different ways by different implementors, leading to a need for post-implementation standardization. (Une chanson connue!).

2. Market pressure for new technology led COBOL vendors to cooperate on object-oriented design through the standardization process. Early implementations of the object oriented features in the draft are now available and users are designing them into new applications.
3. Growth of COBOL in the Unix market led X/Open to develop a Common Application Environment (CAE) providing a portable definition of features essential in a Unix environment but lacking in standard COBOL. The need for portability with non-Unix platforms has resulted in a requirement for inclusion of these features in the international standard.
4. While business-oriented features are a main-stay of COBOL, the many services and interfaces in today's environments demand a wider variety of data types and better interoperation of COBOL with other languages.
5. Technology advances and resulting spread of computers to end users makes it mandatory that computer systems adapt to the languages of users. This gives rise to a need in COBOL for support of large character sets and cultural adaptability.
6. The impact of year 2000 date handling (nous y voilà) generated a requirement for portable year-2000 features in the draft standard, which are now being widely implemented.

The current draft revision addresses a great many of the market requirements for COBOL, but not all of them. Even as the draft standard progresses, new requirements are evolving. With the current trend to web-based operation, users are beginning to reconfigure existing COBOL applications as servers integrated with web clients. COBOL vendors are starting to introduce language extensions to support this environment.

Thus, continued evolution of the international standard for COBOL is essential to provide the benefits of new technologies and new environments to COBOL users worldwide.

Si ce texte sort prochainement sur notre [http favori](#), peut-être y aura-t-il des lecteurs (franco-)anglophones qui pourront nous aider. Toute aide sera la bienvenue.

Venons-en maintenant aux conquêtes et particularités d'APL, d'hier et d'aujourd'hui. Pour cela, je vais reprendre une partie d'un rapport que j'ai fait en revenant d'Obninsk, plein de bonnes résolutions mais en butte aux sempiternelles inerties et incompréhensions habituelles. Il y a des marronniers bien sûr (sens journalistique) et vous m'en excuserez.

Il serait bon que quelque part, on trouve sous forme de tableur, les informations (épurées) qui suivent. Chacun pourrait contribuer à informer ce tableau (par

confirmations et infirmations) et cela permettrait à tous (pourquoi pas la communauté internationale) d'en profiter.

Je reprends le texte au moment où je faisais allusion à une question 'rituelle' que j'ai entendue pour la Nième fois : 'pourquoi un outil si puissant, sans commune mesure avec tout ce qu'on connaît, n'est-il pas plus connu et employé

J'avais la réponse, je me suis tû, elle n'a pas été donnée devinez-là

Cette question que l'on pose depuis plus de vingt ans, on l'entend régulièrement, et au prochain congrès APL92 à Saint Petersburg, l'une des conférences s'intitule: "If APL is so great, why isn't everybody using it?".

Gageons que cette question reviendra éternellement sur le tapis à moins soit d'un miracle digne de la Pentecôte!, soit qu'on arrive à trouver un heureux illuminé détenant des pouvoirs de décision: je ne désespère pas d'en trouver, d'autant plus que je me suis laissé dire que pour la première fois dans l'histoire d'APL, il y aura avec moi un "militaire" qui assistera au congrès APL92. Je souhaite que ce dernier puisse apprécier notre "monde" et qu'il ait des pouvoirs de décision permettant de faire sortir APL de son ghetto: le souhait éternel de tout APListe. Ce sera aussi pour la première fois dans l'histoire d'APL que son congrès international annuel aura lieu en Russie.

[Commentaire: En réalité, aucun militaire n'est venu au congrès APL92, (en fait, je ne me suis jamais fait d'illusion].)

Quelques informations sur APL pour ceux qui n'en ont jamais entendu parler: (cela existe)!

- Inventé par K. E. Iverson: (1er livre: A Programming Language en 1962)
- APL est à la fois une notation mathématique et un langage de programmation
- Chaque année a lieu le congrès international APL (alternativement aux Etats-Unis et en Europe) depuis 1972. Depuis quelques années il y a en même temps une exposition de produits et sociétés utilisant APL ainsi qu'un "software exchange" où on peut acquérir des logiciels APL gratuits (plus d'une centaine à l'heure actuelle).

[Commentaire: Ils ne sont plus gratuits maintenant, mais de prix abordable]

- Depuis 1982, dans la semaine qui précède ou qui suit le congrès APL, se tient la réunion ISO APL.

- APL a été normalisé (ISO 8485) en 1988 et c'est la première fois qu'un langage de programmation a une norme bilingue dont la version française a la même valeur normative que le document anglais. Ceci intéresse de nombreux pays dont la langue officielle dérive du latin. Depuis 1989, les réunions ISO concernent l'extension de

cette norme qui devrait aboutir dans les prochaines années, ce qui procurera à la communauté APL une norme de niveau APL2 d'IBM qui sera probablement trilingue, le russe étant l'une des 3 langues officielles de l'ISO, de nombreux pays de langues slaves pourront ainsi y trouver leur compte.

[Commentaire : Ceci était un vœu pieux, mais on sait que la norme ISO 8485 étendue, si elle sort un jour, ne sera qu'en anglais. Le document actuel fait 290 pages de format A4 en petits caractères, tassés. Je suis le seul rescapé du groupe APL AFNOR. Ce dernier a été dissous finalement, car on ne peut plus parler de groupe quand il n'y a plus qu'un seul individu, je ne me vois pas seul pour assurer la traduction, j'ai bien d'autres choses à faire, même en demi-retraite]

- Il vient d'être démontré (USA, Bernecky) qu'APL est le seul langage de programmation (avec son successeur J) qui soit mathématiquement complet: ce qui permet d'écrire n'importe quel algorithme).

- En plus de la liste (établie par moi-même à la suite d'une demande de l'AFNOR en 1988) de trente sociétés françaises vivant d'APL, on vient d'apprendre qu'il existe bien d'autres foyers et sociétés pratiquant APL en France.

- Quelques vieilles et récentes informations que personne ne connaît (sauf dans la communauté APL).

[Commentaire : vous me pardonneriez le ton que j'employais avec ce qui suit ! : ce n'est pas en prenant les gens à rebrousse poil qu'on les a dans la poche, au contraire, on s'en fait des ennemis, il y a d'autres exemples]..

[Autre commentaire : à confirmer ou infirmer, je ne citais pas ici mes sources

- Savez-vous que les images de synthèse du film "Tron" ont été faites avec APL?
- Savez-vous que la Banque de France utilise APL?
- Savez-vous que les "Mutuelles du Mans" utilisent APL?
- Savez-vous qu'APL est utilisé à la SNCF?
- Savez-vous que l'approvisionnement en pétrole de la France est géré en APL?
- Savez-vous qu'après l'accident de Tchernobyl les russes ont décidé d'utiliser APL pour contrôler leurs centrales nucléaires (voir Congrès APL91 à l'université de Stanford).

Il y a plus d'une dizaine d'années - et nous l'ignorions- l'Académie des Sciences de l'URSS décida de créer sa propre implantation d'APL sur compatible DEC et depuis, l'utilise dans tous ses laboratoires de recherche de pointe...(implantation réalisée par Andrei Kondrashev, Ingénieur système à l'Académie des Sciences de Moscou.

[Commentaire : On sait qu'Andrei travaille maintenant chez Lingo Allegro à Chicago, je l'ai encore vu cet été à Toronto]

- Savez-vous que l'Union des Ingénieurs et Techniciens utilisant la langue Française (UITF), fondée par Pierre Aigrain, ancien ministre, et soutenue par le Haut Conseil à la Francophonie utilise APL et en est très satisfaite?
- Savez-vous qu'une grande banque du Danemark est entièrement gérée en APL?
[Commentaire : Peu importe le nom de cette banque dont on nous a parlé lors du congrès APL90, mais aujourd'hui, de très nombreuses banques du nord de l'Europe utilisent couramment APL].
- Savez-vous que les approvisionnements de "l'Aéroport de Paris" (Roissy, Orly, Le Bourget, Toussu le Noble etc) sont faits en APL?
- Savez-vous que la rotation des avions, les équipages et le fret est entièrement géré en APL par la "British Airways" sur l'ensemble de son réseau mondial?
- Savez-vous que les 20 premières sociétés informatiques britanniques utilisent APL?
- Savez-vous que la société Morgan Stanley gère entièrement et en temps réel toutes les variations de la bourse de Wall Street avec sa propre implantation d'APL?
- Savez-vous que Reuter (principale agence de presse du monde) a racheté IP Sharp (la société où travaillait K. Iverson et qui utilise sa version d'APL) à Toronto (Canada) à cause de son réseau de télécommunications mondial entièrement construit en APL?
[Commentaire cela date un peu]
- Savez-vous qu'APL est utilisé à Tsukuba, principal centre de recherche de pointe du Japon?
- Savez-vous que le produit "Statgraphics" (mondialement connu) est entièrement en APL?
- Savez-vous que le produit "MAPLE V" (mondialement connu) a été écrit par une équipe d'APListes de l'Université de Waterloo?
- Savez-vous qu'APL est très développé dans les pays Scandinaves en particulier Finlande et Danemark
- Savez-vous qu'IBM en Allemagne a réalisé un système remarquable de tomographie tri-dimensionnelle du cerveau entièrement en APL? (En 1990, au congrès APL de Copenhague, IBM nous a affirmé que ces images époustouflantes qu'on nous a montré n'auraient jamais pu être réalisées (et surtout en si peu de temps) sans APL)
[Commentaire : récemment, un ami chercheur, m'a dédié son livre (dans lequel il fait une découverte fondamentale) en parlant de notre commune passion pour APL il ajoute entre parenthèse : 'sans lequel je n'aurais jamais découvert tout cela'. Bien sûr, il sait que c'est mon leitmotiv de dire qu'APL peut tout, qu'il fait tout vite, qu'il excelle dans la recherche, que c'est un outil puissant d'investigation scientifique etc, mais je ne pense pas qu'il disait cela seulement pour me faire

plaisir, je le crois, et il sait que je suis prêt à le croire, mais dites-moi, hors du monde APL, le croirait-on ?, je n'en suis pas sûr, vue mon expérience et celle des autres : je ne cite personne..]

- Andrei Kondrashev (cité plus haut) est également psychologue et il étudie en APL le comportement de la population suite à la perestroïka.

Il y en aurait bien d'autres à citer, mais après 23 ans de pratique d'APL, si ce papier parvient à des amis du monde APL, que l'on me pardonne les oublis nécessaires que j'ai pu faire, on ne peut pas ni tout retenir, ni tout savoir et on comprendra facilement que tout le monde ne rend pas compte de ce qu'il fait avec un produit aussi stratégique.

[Commentaire, aujourd'hui, ce texte parvient dans le monde APL. Les oublis dont je parlais (et j'ajouterais mes lacunes), il est temps de les combler

Conclusion:

Si les arguments précédents ne sont pas suffisants j'en ai de plus frappants mais ils risquent de faire mal, c'est une tonne de littérature APL depuis 25 ans qu'il existe: Le dernier numéro de la revue "IBM Systems Journal" (Vol 30 No 4) qui fête le 25ième anniversaire d'APL est consacré exclusivement à APL.

Selon la parole bien connue: nul n'est prophète en son pays, je l'avais vite constaté et me suis bien gardé de dépenser de l'énergie inutilement à faire du prosélytisme, on ne pourra pas me reprocher non plus de faire de la rétention d'information, j'ai clairement exprimé depuis longtemps mon opinion en déclarant que je considérais APL comme un produit stratégique non seulement de première envergure pour un mathématicien et la recherche scientifique (APL étant bien connu comme un excellent outil de prototypage, donc de recherche) mais aussi pour tous ceux qui ont besoin de puissance de recherche.. je ne fais pas de dessin.

Compte tenu de ce qui vient d'être dit clairement ou entre les lignes, on ne s'étonnera pas du rejet d'APL dans de nombreux milieux.

Je laisse au lecteur le soin de tirer toute conclusion utile et de prendre les mesures qui conviennent. Fin du texte

[Commentaire : ce dernier bout de texte, je le reproduis quand même, qu'on me pardonne, je l'ai écrit non seulement dans la fureur, mais aussi dans le désarroi, devant le mur d'incompréhension auquel je me suis heurté pendant bien des années: quand j'étais à l'ENS de Cachan, nous disions entre mathématiciens : 'nous sommes cernés par les cons' (sic). Aujourd'hui, sans citer personne mais comprenez qui le voudra bien : je citerai un dessin humoristique qui représente un homme fier comme Artaban, derrière son bureau avec son poing sur la table en s'exclamant : 'Dans mon école, on ne m'a pas appris à tout comprendre, mais à dénigrer ce que je ne comprends pas'.]

Je me retiens pour ne pas en rajouter...

Conclusion : Je reproduis ici un passage d'une interview avec Hubert Reeves dans l'Express du 7 novembre 1990 page 134

« Parmi les 100 milliards d'étoiles de notre galaxie, plusieurs milliards sont semblables au Soleil. On peut supposer que plusieurs d'entre elles ont des cortèges planétaires et que , dans ces cortèges , une planète pourrait être située à une distance raisonnable de son soleil. Dès que, sur une planète, les températures moyennes permettent à l'eau de rester liquide, vraisemblablement, la vie va se développer. Peut-être pas sous la forme précise que nous connaissons. Il existe probablement d'autres faunes, d'autres flores, mais sans doute pas très différentes.»

Nous pouvons donc espérer que des êtres pas très différents de nous existent quelque part dans l'univers, sur une planète plus clémente, moins polluée, propice au véritable développement d'APL (J'ai toujours rêvé qu'on l'enseigne à l'école, mais pour l'instant, notre planète n'y est pas favorable, c'est le moins qu'on puisse dire). Sur cette planète, donc, APL existe, tout le monde l'a appris à l'école, ce qui a permis aux recherches d'aboutir à des progrès fulgurants dans tous les domaines et qui leur a permis de vaincre les maladies, d'éliminer les guerres (je n'ajouterai pas les religions pour ne pas me faire incendier !) et que les habitants connaissent enfin une vie paisible et agréable dans une nature luxuriante et calme. Si, lorsque nous quittons cette terre, une partie de nous même peut se détacher et aller aussi vite que les tachyons, peut-être pourrions nous aller nous recycler dans ce nirvana moins hostile... c'est mon souhait.

Plutôt que de représenter ici un dessin de nature luxuriante, je ne résiste pas au désir de dire que l'ami dont je parlais plus haut est Jean-Claude Perez, son livre s'intitule l'ADN décrypté ou la découverte et les preuves du langage caché de l'ADN. Il a trouvé que dans les séquences d'acides aminés TCAG, règne une structure où on y trouve le nombre d'or et les nombres des suites de Fibonacci ou celles que je baptisais Fibonacciennes dans mon article sur 'codes à barre et Nombre d'or'. Ainsi le Nombre d'or se trouve partout dans la nature, pas seulement dans les plantes, (il y a longtemps qu'on le savait), mais en nous-même. En me souvenant que dans un examen de physique électronique qui m'avait beaucoup plu, il s'agissait de savoir ce que devenait notre univers en changeant la constante diélectrique du vide, cela me fait penser, que l'univers peut comporter des planètes où des formes de vie sont différentes : on change certaines structures (comme l'ADN) et certaines constantes physiques (Avogadro, Boltzmann etc) et on obtient d'autres formes de vie.

Ne nous plaignons pas, nous avons la chance de connaître une forme de vie et une époque où est apparu APL, quel bonheur ! et maintenant il y a J son successeur qui nous promet encore bien des joies.

Nota Bene : (message reçu par e-mail)

Suj : petite précision au sujet de ... la précision!

Date :17/11/97 23:09:33

De : Dumontier

A : Lemagnen

Chère Ludmila,

Ayant regardé après coup dans le dictionnaire le mot balustre, je m'aperçois qu'ils ne décrivent pas la fonction du balustre: ce n'est pas seulement que décoratif comme ils le laissent entendre, en fait il a une fonction bien précise:

C'est un anneau cylindrique en acier à ressort, il a pour fonction d'écarter constamment les branches du compas pour que les branches appuient sur un seul et même côté du filetage afin qu'il n'y ait aucun jeu, pour assurer la précision.

Qui n'a pas utilisé un compas sans balustre, pas assez serré et qui trace une spirale (en général divergente par l'effet centrifuge!) plutôt qu'un cercle!

Autre précision, je regrette ne pas avoir de logiciel de géométrie (j'en ai vu un super en 1994, mais je ne suis pas parvenu à m'en procurer un gratuitement), je t'aurais fait quelque chose d'ultra précis!. Paint suffit pour la démonstration...

Autre commentaire: Je cite la Pentecôte, car elle relate un fait où tous les gens (le contraire de la confusion des langues à Babel), comprenaient les langues étrangères (APL) dans leur propre langue!

L'époque que nous vivons est pire que celle de Babel. Elle n'est pas que celle du chaos, mais aussi celle de la confusion des valeurs, croyances, certitudes, informations (trop d'information = pas d'information) ou autres.

Pour mon ami J-C Perez, ce n'est pas par hasard que je le cite, en effet justement le fait que nous existons n'est pas un hasard, il y a une harmonie et des données mathématiques qui procurent à la nature une harmonie qui lui permet d'organiser la vie et cela rejoint Hubert Reeves qui dit que si les conditions le permettent, la vie apparaît, il le constate, c'est tout, mais j'y vois une explication, j'ajouterai même que je vois l'univers comme des îlots potentiels (et de ce fait peut-être réels, mais on n'a pas encore pu le vérifier) où en changeant localement certains paramètres judicieux, on fait apparaître toutes sortes de vies. Cela rejoint mes articles au sujet du nombre d'or, des suites Fibonacciennes (Lucas) ou quasi Fibonacciennes (celles que j'ai inventées dans mon article sur les codes à barre). Si il y a encore un peu de place, et si cela permet d'éclairer mon texte, tu peux le rajouter...

Bises,

Michel

Rôles d'APL, dans la surveillance par satellite.

**résumé en français de l'article de Jack G. Rudd
par Michel Dumontier**

Durant la guerre du Golfe, les USA et ses alliés ont acquis une expérience dans les domaines de la défense. Ces avantages ont aidé les forces de coalition en leur fournissant des données météorologiques, de l'assistance à la navigation, la position géographique des forces et d'autres renseignements.

Le programme qui nous intéresse se nomme DSP : Defense Support Program, sa mission est de fournir un système de surveillance par satellite, hautement disponible, résistant à la survie et fiable, qui donne ainsi aux USA et ses alliés une surveillance des missiles balistiques et d'autres informations relatives aux départs de missiles ou l'explosion d'armes nucléaires.

Durant la guerre du golfe, le DSP a détecté les départs des missiles SCUD irakiens, averti en temps opportun les populations civiles et les forces de coalition en Israël et en Arabie Saoudite.

Cette expérience récente (écrit pour APL93) confirme la pertinence et la flexibilité du DSP pour prévenir à temps les attaques de missiles contre les USA ou ses alliés. Dans un contexte plus large, le DSP continue à avoir une valeur dissuasive qui contribue à la stabilité et la paix dans une région, évite le conflit nucléaire et permet le contrôle de l'armement international.

Précision technique :, le DSP détecte les ICBM, les SLBM, les IRBM et les SRBM.

Les plus alarmants sont

Les ICBM: Missiles Balistiques Inter Continentaux.

Les SLBM : Missiles Balistiques Lancés (Launched : c'est la même lettre) à partir de sous-marins Soviétiques.

U.S. Space Command s'est servie également des informations du DSP pour alerter les forces de coalition des attaques de missiles, y compris les Batteries de Missiles Patriot.

Les algorithmes de traque et de détection de missiles qui ont permis la détection des SCUD et les avertissements en temps réel durant la guerre du golfe étaient essentiellement les mêmes, avec des modifications mineures, que ceux qui ont été développés et programmés au début des années 70 par la Division Systèmes Fédéraux d'IBM.

Ces algorithmes ont d'abord été développés, expérimentés (par prototypes) et testés en APL.

On pourrait s'arrêter après cette première partie nommée "Éléments de base," l'essentiel a été dit : à la fin on retiendra qu'APL a été toujours utilisé pour le DSP à partir de 1971. Ce fût l'année où APL a été utilisé intensément pour développer des algorithmes et faire du prototypage pour le DSP. A chaque fois qu'il y a eu urgence, APL était la seule porte de salut, car il permettait une expérimentation rapide et un développement rapide des algorithmes nécessaires pour atteindre les buts poursuivis, surtout dans le domaine du temps réel.

Avant de traduire la conclusion, voici un autre passage de ce que dit James Martin, conférencier de renommée mondiale, qui rejoint étroitement ce qui a été constaté avec l'expérience APL-DSP : Lorsqu'une équipe est amenée à intégrer une grande quantité de programmation de prototypage, si APL réduit le nombre de participants par un facteur de trois, il réduit aussi les échanges et par conséquent, les erreurs de communication et la reprogrammation par un facteur supérieur à trois. L'effet sur la productivité est cumulatif et synergique

Un autre encore : En 1987, un universitaire a coopéré au projet et ce nouvel intervenant n'avait eu aucune expérience d'APL auparavant : presque instantanément, il devint efficace pour « scalairiser » des fonctions APL2 en entrée d'un traducteur ADA. Ceci est encore un autre exemple de la rapidité avec laquelle une personne brillante qui a des inclinations naturelles en mathématiques et dans le domaine scientifique, peut devenir extrêmement productif en APL

Un autre ? oui : on ne va pas se priver pour celui-là ! un paragraphe qui s'intitule "utilisation militaire directe de programmes APL2

Le FSC d'IBM à Boulder a eu un contrat récent avec le (réputé) Laboratoire National de Lawrence Livermore pour fournir des nouveaux programmes d'analyse à l'usage des analystes de l'Air Force Commande de l'U.S. Air Force.

Que ce soit des programmes d'analyse, ou de prototypage, tous sont écrits en APL2. Jusqu'à présent, aucune pression n'a été exercée pour traduire ces programmes dans des langages plus familiers aux analystes. En réalité, les analystes ont montré un grand intérêt et une capacité à travailler directement sur les programmes APL et même à les modifier de manière non triviale.

Encore un autre, j'ai du mal à résister ce sera le dernier avant la conclusion

Le paragraphe s'intitule « Utilisation d'APL dans d'autres programmes de surveillance »

Au début de 1984, peu après l'annonce du Président Reagan de la SDI (Initiative de Défense Stratégique), notre organisation (là où travaille l'auteur, le FSC IBM de Boulder), eût la charge de faire le prototypage et concevoir le système du BSTS (Boost Surveillance Tracking System) ainsi que celui du SSTS (Surveillance and Tracking System) pour la SDI.

Plus tard, le BSTS évolua vers l'AWS (Advanced Warning System) et éventuellement vers le FEWS (Follow-on Early Warning System). Ce système est prévu pour remplacer le DSP et son développement est assuré par l'USAF.

Le FSC IBM de Boulder sous-traite à Lockheed pour ce programme.

Plus récemment, le FSC IBM sous-traite à Rockwell pour Brillant Eyes“qui est une partie du programme GPALS de la SDI.

APL a été utilisé comme langage de prototypage, pour tous ces programmes depuis le début. D'autres rôles pour APL ont encore à définir“

Ceci prouve une fois de plus (même après que les APListes l'ont proclamé depuis longtemps sur tous les toits), que les américains et plus récemment les russes , ont toujours su, savent et sauront tirer partie de l'immense avantage d'APL, surtout dans les domaines stratégiques qui je le rappelle ne sont pas que du domaine militaire: mais aussi celui du scientifique, médical etc.

Conclusion:

La conclusion reprend largement ce qui a été dit, je ne reprendrai que la dernière phrase :

Enfin, nous avons fourni des programmes d'analyse sur des stations de travail pour l'utilisation directe des analystes militaires. Nous utilisons APL2 pour étudier et prototyper“des solutions massivement parallèles pour des applications militaires qui requièrent trop de calculs numériques pour des approches traditionnelles (entendez bien sûr non-APListes) (je sens qu'il y en a qui ne comprennent pas, ou qui ne veulent pas comprendre...)

L'auteur donne ensuite une liste (peut-être incomplète et il s'en excuse) de 74 programmeurs qui ont contribué au programme du FSC d'IBM.



Dessin de Bernard Legrand, APListe connu de nous tous, mais le dessinateur...? ? ? alors que c'est lui qui a fait tous les dessins de ses livres.

Roles of APL in Satellite Surveillance

par Jack G. Rudd

Background

Following are excerpts from a statement approved for public release by the U.S. government:

"During Desert Shield/Storm, the United States and its coalition allies capitalized upon the use of space-based systems in support of U.S. and other coalition objectives. These space-based assets assisted coalition forces by providing weather data, navigational assistance in desert terrain, the geographic disposition of forces and other related intelligence.

"Integral to this allied effort was the Defense Support Program. The DSP mission is to provide a highly available, survivable and reliable satellite-borne surveillance system which provides the United States and its allies ballistic missile early warning and other information related to missile launches, surveillance and the detonation of nuclear weapons.

"During Desert Storm, DSP detected the launch of Iraqi SCUD missiles, provided timely warning to civilian populations and coalition forces in Israel and Saudi Arabia, and helped safeguard property. The United Nations Coalition Forces who were protected can be justifiably proud of the system's outstanding defensive performance. DSP's superb performance conclusively proved that the system provides significant early warning of enemy tactical missile attack.

"This recent experience confirms the continued relevance and flexibility of DSP, a system intended primarily to provide early warning of strategic missile attack against the United States and its friends and allies around the world. In this larger strategic context, the deterrent value of DSP provides a continuing contribution to regional peace and stability, the avoidance of nuclear conflict, and international arms control.

"Questions and answers to potential news media questions on missile warning

capability:

News reports have said the system uses infrared devices to detect missiles. Is that correct? How does that work?

Answer: Yes. Missiles generate intense heat and infrared returns. The sensors on the satellite look for those infrared returns and transmit the data back to earth.

It has been reported that information from the satellite system was routed through Space Command to coalition forces in Saudi Arabia. Is that true?

Answer: United States Space Command in Colorado Springs provided information to forces engaged in the Persian Gulf conflict.

As part of the DSP system, did the Joint Defense Facility Nurrungar in Australia support Desert Storm?

Answer: Yes.

Did the Joint Defense Facility Nurrungar report SCUD launches?

Answer: Yes.

Were all SCUDs launched by Iraq detected by the satellite system and warning given?

Answer: Warning was given in every SCUD attack. The system was very effective.

What types of ballistic missiles can the DSP system detect?

Answer: DSP can detect ICBMs, SLBMs, IRBMs, and SRBMs.

How else was DSP information utilized?

Answer: DSP information was used by U.S. Space Command to alert coalition forces, including Patriot Missile Batteries, of missile attack."

The missile detection and tracking algorithms that enabled SCUD detections and warnings in real time during Desert Storm were essentially the same algorithms, with only minor modifications, which IBM's Federal Systems Division originally developed and programmed in the early 1970's. Those algorithms were first developed, prototyped and tested in APL.

IBM's Involvement in the DSP Program

The IBM Federal Systems Company at Boulder, Colorado has three decades of experience on the DSP program and its precursor program. A feasibility demonstration program which preceded DSP began in the late 1950's, just after the Soviets launched the first Sputnik. IBM's involvement on that program began in 1962, with a small contingent of people based in Sunnyvale, California.

The DSP system went into production in the late 1960's in Azusa, California, where IBM participated as the primary subcontractor for engineering and development of ground station real-time software. The system was then designed primarily for detecting intercontinental ballistic missiles (ICBMs) in the eastern hemisphere.

In 1971 IBM's Federal Systems Division in Westlake Village, California became an associate prime contractor on the DSP program by winning a major engineering and development contract to upgrade the ground station software to detect Soviet submarine-launched ballistic missiles (SLBMs) in the western hemisphere. This was a major challenge because it was not then known whether the system could detect and report these relatively dim targets without a significant false report problem, or whether the new algorithms and software to accomplish this could be made efficient enough to process the satellite downlink data within the severely limited computational resources of those years (an IBM System/360 model 75 with 0.8 MIPS, 3 megabytes of RAM, a few hundred megabytes of DASD, and special purpose hardware for front-end processing).

1971 was also the year that APL was first used intensively to develop and prototype algorithms for the DSP program.

The DSP Development Environment in the 1970's

The following paragraphs are not intended simply to describe the primitive conditions in those exciting yesteryears of project development. Rather, this material lays the foundation for describing the prototyping role of APL in the early years of the DSP program. And it provides an opportunity to record observations about some specific engineering and software development methods which worked well prior to the era of Modern Software Engineering Practices, the Ada language, Object-Oriented Design, and other popular trends in project development for the federal government.

Our development environment consisted of an IBM 360/75 with an OS/MVT

operating system. Batch processing capability was provided for Fortran and assembler users. Although interactive capability was provided to the users of the real-time operational system, no interactive capability was provided for engineering and development. Input was via card decks and tapes, and output consisted of hard copy listings. Disk data sets were used at one's own risk, since they could be overwritten at any time by other users.

Software engineers, software developers, software testers, algorithm developers and data analysts all shared this single environment. Users of the real-time operational system and the spacecraft sensor simulator had priority over offline batch users. Consequently, engineers and analysts generally experienced 24-hour turnaround time on their batch jobs. Even the simplest error would not usually be discovered for at least a day.

Juxtaposed against this environment, the defining reality of the project was that of urgency. In strategic weaponry the Soviets were in the ascendancy. The pace of their ICBM buildup and the technological sophistication of their new ICBMs and SLBMs were alarming. Their actions gave every appearance of attempting to create a first-strike capability against U.S. land-based missiles and bombers.

More specifically to the point, Soviet ballistic missile submarines were beginning to patrol the areas off U.S. coasts earlier than previously expected. And at that time U.S. military forces were unable to detect Soviet SLBM launches over some critical broad ocean areas.

:p.

Our schedules reflected this sense of urgency. We had to deliver our initial systems in months. And from the very beginning we had to place a very high value on the efficiency of our real-time code, sacrificing other values when necessary.

Unlike most commercial systems, large real-time processing systems for military use often require many sophisticated mathematical algorithms, and the programming which implements these algorithms (in terms of source lines of code, or SLOC) can be as much as 20 percent of the entire system. DSP needed algorithms for satellite attitude determination, computation of ephemerides, clutter rejection, missile tracking and detection, geopolitical region determination, tactical parameter estimation, target classification, and many other functions.

In building or extensively upgrading a large system of this kind, algorithm

development is often the first activity to experience intense schedule pressure. Our group needed as rapidly as possible to get our algorithms working well enough to deliver initial versions of them to the software engineering and development organizations.

Thus we regarded those 24-hour delays (described above) as an outrage; a crime against the project, against our customers, and against Western civilization. Therefore, some of us began to use APL intensively to speed up the process.

Productivity of Development in APL

The conciseness of APL has contributed directly to its productivity, which is legendary. In Reference 3 James Martin cites productivity factors of between four and ten when comparing APL with traditional languages with regard to speed of application development.

For prototyping complex mathematical algorithms, our experience has been that, compared with traditional languages, APL2 improves the speed of development time by at least a factor of three, for one individual. This is also the approximate factor by which APL2 decreases the number of lines of source code (excluding comments).

However, when a team is required to integrate a large amount of prototype code, James Martin's figures are closer to the mark. If using APL reduces the number of participants by roughly a factor of three, then it reduces the pairwise human communication required, and consequent miscommunication and rework, by much more than a factor of three. The effect on productivity for the project is cumulative and synergistic.

APL'S Role in Minimizing Cycle Time

We had no APL system in our Westlake facility in the 1970's. Instead we connected to commercial IBM time-sharing services in the San Jose/Palo Alto area via phone lines and modems. Our input/output devices were IBM 2741 Selectric typewriters with individually owned APL typeballs. Our workspaces were limited to 125 kilobytes each. And the data rate was 134.5 baud. In such an environment, the dreaded message "WS FULL" was commonplace. Also common but far more frustrating was receiving a "RESEND" message after every depression of the enter key.

These constraints and frustrations were very significant detriments to

productivity. Nevertheless, for most algorithm development purposes, this environment was vastly superior to that of 24-hour turnaround times on Fortran batch runs.

However, because of these limitations, we were unable to write and integrate large algorithms entirely in APL or to test an APL function against a large amount of observed sensor data. To make the best of this situation, we wrote small algorithms or algorithm fragments in APL, tested them as extensively as was practical in a small workspace, translated them manually to Fortran, integrated them into our existing Fortran prototype, keypunched the cards, inspected them thoroughly, inserted them into the card decks, submitted the batch run at our Westlake facility, and went home.

This sequence of activities being a daily exercise, we were usually defining today's code additions in Fortran before we saw the results of executing yesterday's work. In such a situation, the best way to keep from getting behind (and thus confused from a configuration management viewpoint) was to avoid as many errors as possible. We got much practice at this. After a time our proficiency was such that on most days our newly integrated Fortran code (which was translated from APL) would execute without error and would accomplish the intended objective of the new algorithm.

As our Fortran prototypes matured, we delivered them to our DSP software developers, who integrated them into the deliverable real-time product. This happened quite early in the development cycle, with the very important benefit that these programmers became very practiced in writing and integrating real-time code. They became very productive very early.

This was a major difference with modern practices. Today's developers generally do not commence programming until after the design phase is complete, which can be years after the project begins. It is forbidden to assume that the designers are many of the same people as the programmers. And finally, designing without having performed enough trial coding to locate resource or performance bottlenecks is guaranteed to induce later redesign. Thus software development productivity is reduced for all these reasons.

Evolution of our APL Environment

In 1977 we obtained a Tektronix 4015 terminal with a 300 baud connection, which was later increased to 1200 baud. And in 1982 we installed VSAPL

under a VM operating system on an IBM 3033 mainframe in our Westlake facility.

For users who touch-typed the APL characters, each of these new increases in bandwidth had surprising effects. At least for the first two new plateaus, the human thought processes and keyboard inputs seemed to speed up automatically, without extra effort, to match the new level of bandwidth. For the third plateau the effects of the improved bandwidth were less clear.

In 1982 it was a standard goal for interactive systems within IBM to achieve subsecond response time to trivial keyboard entries. However, late at night when there was only one user, our system achieved a response time of approximately 0.15 seconds. At this level it seemed that the system almost always kept up with human interactions.

However, with the later addition of a terminal concentrator, our system response time late at night increased to about 0.30 seconds; and for one accustomed to response times of 0.15 second, this interfered quite significantly with human interactions using APL. So it became far from obvious that APL users cannot realize additional practical benefits from system response times that are below 0.15 seconds.

It was not until 1992, using APL2 on IBM RISC System/6000 workstations, that we regained that liberating experience of not having to slow down deliberately to give the system enough time to respond; plus the joys of 2-gigabyte DASD and workspaces of 100 megabytes or more.

A Single Global DSP System

In 1974 our USAF customer studied the performance of its new western hemisphere DSP system on eastern hemisphere observations of ICBMs and space launches. The customer's conclusion was that this new system, designed primarily to detect and track relatively dim SLBM launches, actually outperformed the older DSP ground station processing software for events observed in the eastern hemisphere (such as ICBMs and space launches).

Thus, although we had not planned for this contingency, our algorithms and software were sufficiently robust that our customer adopted them for global operational use. APLers will not be surprised that our attention to generality and edge conditions was rewarded.

APL Prototyping for Fixed Point Microcode

Also in 1974 we won a contract to design, build and deliver a transportable version of the DSP ground-based processing system, including communications, processing hardware and software. The processing platform selected for our most mathematically sophisticated and numerically intensive applications (clutter rejection, missile detection and tracking, multiple event resolution, and missile classification) was a very special-purpose signal processor that was specifically designed for rapid processing of sonar signals.

This processor had an instruction set that supported fixed point add, multiply, sine and cosine instructions (mostly only 16-bit), but had no instruction at all for reciprocal, division or square root, which had to be simulated. Shift instructions were supported, but with significant limitations. One couldn't shift an arbitrary number of bits with a single instruction. Branches were allowed, but were very restricted. At most 16 branch points were allowed within a given microcoded program. Also, the instructions were executed in pipeline fashion. And there were many other limitations.

Our missile detection and tracking algorithms, on the other hand, were originally developed for general purpose computers, preferably in a language and on a platform that conveniently supported 64-bit floating point arithmetic. So our software development process for this project consisted of trying to force-fit our surveillance applications onto a machine and into a language that were optimized for fast fourier transforms.

Realizing that only the most detailed prototyping would enable the project to succeed, we decided to prototype in two stages. For most algorithms we first created APL prototype code that minimized the number of branch points, reciprocals, divides and square roots. Then we would translate this into fixed-point Fortran prototype code which mimicked the effects of the microcode instructions bit for bit, except for the pipelined aspects of the processing. Finally, the microcoding was performed as a translation of the fixed point Fortran code.

One algorithm, a three-state batch filter, was extremely difficult to microcode without segmenting it into pieces, looping over the calls to the segments, and thus unacceptably increasing the overhead time required to load the subprograms into memory. So we even wrote a fixed-point APL prototype of this routine, as well.

There was a significant payoff from this detailed prototyping. For example, the microcode program which implements this three-state filter has never required maintenance of any kind (nor has the algorithm), even after 14 years in the field. By contrast, the microcode program which performed missile classification was not so prototyped. It later had to be scrapped, and the design effort begun again from scratch.

In 1981 an analysis of data from our Quality Assurance organization revealed that the number of incident reports (similar to APARs) and design change requests per thousand lines of code were lower by about a factor of six for code which had been prototyped in detail than for similar code which was developed in a more traditional way, via English text, mathematical equations and flow diagrams.

APL Analysis Programs and Algorithm Improvements

The 1982 installation of VSAPL in our facility gave us access via APL to large files of historical DSP data. DSP satellites had observed thousands of rocket and missile launches and other phenomena, constituting hundreds of megabytes of data. In addition, the complete historical ephemerides of each DSP satellite were available on tape.

However, in 1982 we had not yet produced data analysis programs in any language which would give analysts the ability to call up this information interactively. This was estimated to be a major undertaking, requiring its own budget, its own work packages, and corresponding internal and external approvals.

But with the local installation of VSAPL, one person accomplished all these things using APL in his spare time. Examples of these display and data analysis programs are shown in Reference 4. In addition, it became convenient to exercise a new algorithm against thousands of historically observed events, along with the corresponding satellite ephemerides.

These programs were later converted to APL2 and used by algorithm developers to make very significant improvements in our satellite surveillance algorithms. Our newer algorithms which perform missile classification and tactical parameter estimation perform dramatically better than do the algorithms that are currently in operational use.

Evolution of our Prototyping Practices

For many years we performed rapid prototyping only for critical algorithms. In 1984 we began more comprehensive rapid prototyping in preparation for a major upgrade of the DSP ground station software. And in 1985 we converted from VSAPL to APL2. In subsequent years we prototyped and integrated the vast majority of the mathematical algorithms that this project required. Including the necessary architectural infrastructures and superstructures, we wrote tens of thousands of lines of APL2 code for the project.

For our prototyping and integration purposes, the biggest single benefit of APL2 over classical APL is that it allows nested vectors to serve the purpose of "parameter lists" for input and output variables. This enabled a major reduction in the use of global variables, and thus removed a major source of potential errors.

However, APL2 also provides many other valuable benefits that either classical APL did not provide or that we had never before had occasion to try. One is the ability, through the Name Association facility, to call Fortran subroutines conveniently. As our APL2 prototypes became larger and more encompassing, we began isolating the hot spots in the code (the performance bottlenecks) and translating them to Fortran (often by way of Ada, as we shall see below). And we often created Fortran and Ada versions of algorithms for benchmarking purposes, even for many algorithms that were not bottlenecks in APL2.

Before upgrading to VM/XA and subsequent operating systems, we were limited to 16 megabyte virtual machines and 15 megabyte APL2 workspaces. This was barely adequate for prototyping the processing of sensor data from one satellite downlink. However, we needed to prototype the coordinated and simultaneous processing of data from several satellites simultaneously. And so another facility we found necessary to use was APL2's Global Shared Variable Processor. This enabled us to coordinate the simultaneous processing and communication of several APL2 workspaces at once.

An instructive aspect of this advanced and esoteric design work was that it was accomplished entirely and flawlessly by an experienced programmer who was nevertheless an APL novice. He had had neither any previous work experience using APL nor any classroom instruction in APL. Aside from asking some questions of local APL experts, he relied entirely on APL2 manuals and the classic book, Reference 1.

Graceful and Ungraceful Transition to Ada

In early 1985, in preparation for a major project development effort using new military software engineering standards including the Ada language, our Westlake organization began four small trial coding projects using Ada. One of these projects consisted of transforming an existing algorithm written in 120 lines of uncommented APL into a functionally equivalent Ada version, including extensive comments. The same procedures were to be used in this small effort as were envisaged for the large project.

Thus, for example, this tiny project had a requirements phase, a design phase and a programming phase, with three corresponding labor-intensive inspections of unexecutable hard copy listings.

The project took five months to complete. The resulting Ada program consisted of 1561 source lines of Ada (1005 if only semicolons are counted) plus 1961 lines of comments or blanks. The bulkiness of the Ada code and the low productivity of this process in creating it were alarming to those of us who were engaged in rapid prototyping.

Coincidentally we had already begun to use APL2 to develop a semi-automated APL2-to-Ada translator (described in Reference 5) to assist us in transforming our prototype code into readable Ada prototype code, which our Ada developers could adapt and migrate into the real-time product, just as their predecessors had successfully migrated our Fortran prototype code in previous years.

When this translator was nearly completed, it was used to transform this same 120 lines of APL code into Ada. This required twenty hours of human effort in "scalarizing" the original APL code, and a few minutes of computer time. The result was 550 lines of Ada source code, no errors, and noticeably improved computational throughput. For this relatively uncommented code, the improvement in development cycle time over that of the corresponding manual effort was somewhere between 20:1 and 40:1, depending on the amount of effort that was expended to produce the commentary for the Ada code.

For completeness it must also be added that this schedule factor is somewhat inflated by the fact that the person who scalarized the APL code for input to the translator was also the person who developed the original algorithm.

In the automatically translated Ada code, the original variable names

in APL are preserved, the Ada code is properly structured and indented, and original comments are preserved intact. Through the scalarizing process the user has complete control over the structure and appearance of the translated Ada code. Since efficiency of execution was valued highly, and since the translator was intended mainly for highly algorithmic code, the translator was designed to support only integer and long float variable types. Abstract data types were deliberately not supported. Thus the character of this resulting Ada code is very similar to that of structured and indented Fortran.

This translator was written by the author and a university co-op who had had very little previous APL experience. Since we worked different but somewhat overlapping shifts, our modus operandi was simply to work within the same workspace. Each of us would send the newly modified workspace to the other at the end of his working day, and in a state suitable for the other to pick up and continue. Both of us kept backup workspaces, and we had no configuration control problems worth mentioning.

Two months into the effort, the novice was so accomplished that it was difficult to discern from the APL code which of us had written a given function. A while later, when he became confident enough to design and program directly from the keyboard, without writing anything down, his productivity was approximately the same as mine.

In 1987 another university co-op and the author used this translator to create 4200 lines of working Ada prototype code in 2.5 weeks. This co-op had had no previous APL experience or instruction at all, yet almost instantly he became proficient at scalarizing existing APL2 functions for input to the translator. This was yet another example of the speed with which a bright person who is scientifically or mathematically inclined can become extremely productive with APL.

These 4200 lines of Ada code implemented our track formation and missile detection algorithms using observations from two satellites at once. Thus this program was quite representative of our numerically intensive algorithms. We delivered this program to our government customer, and it was later used to verify the correctness of execution of candidate Ada compilers running on candidate hardware platforms.

It turns out in retrospect that this program should also have been used to benchmark the efficiency of execution of these compilers and platforms. The measured efficiency in executing this program on the platform that

was finally selected (in terms of MAIPS, millions of Ada instructions per second) was within 20 percent of that for the entire system. However, since the customer used much simpler code sequences for benchmarking purposes, the actual extrapolation performed was considerably less accurate. The result was that the total number of platforms required was significantly underestimated.

So another lesson learned from our prototyping experience is always to benchmark with a significant amount of actual prototype code, in the target language, on the target platform. Benchmarking experiments which fall short of this standard should not be used as a basis for serious decisions.

Development of our APL-to-Ada translator was funded by our Air Force customer, with the specific intention of using it on our contract. In 1988, after the translator was evaluated favorably by an independent group, our software development organization even stated formally its intention to use the translator wherever appropriate.

But this was not to be. The primary reason was that the translator did not support abstract data types, which were then widely believed to be a sine qua non of good Ada design even for highly mathematical functions.

In the context of project development for the federal government, standard practice requires that the engineering design specifications be communicated to the software development organization via documents containing English text, mathematical equations, design flow diagrams, and such like. However, when a system design is sufficiently complex, the ambiguity and incompleteness that are inherent in these vehicles render them inadequate for the job. Indeed, that is one of the primary reasons to perform extensive prototyping in the first instance.

And so, on this very complex system, because of practical experience with these inadequacies, eventually our software development organization became eager to accept working prototype code in both APL2 and Fortran, especially for highly mathematical functions. However, because abstract data types pervaded all aspects of the real-time Ada code, mechanically translated Ada code was not used as a communication vehicle. Instead, the translations from APL2 and Fortran into production Ada code were performed manually.

However, the vast majority of our Fortran prototype code had been originally translated from APL2 prototype code. And much of that translation had been accomplished by scalarizing the original APL2 code, passing it through

our APL2-to-Ada translator, and manually translating the resulting Ada prototype code to Fortran. That procedure, although it seems cumbersome to the uninitiated, was found often to be faster overall and less prone to error than direct manual translation from APL2 to Fortran.

Thus the final result of this overall process was that significant sections of the real-time Ada production software on this project have lived in an earlier incarnation as output from our APL2-to-Ada translator.

Uses of APL on other Surveillance Programs

Beginning in 1984, not long after President Reagan's announcement of the Strategic Defense Initiative (SDI), our organization in Westlake Village began preliminary system design work, including algorithm prototyping, for SDI's proposed Boost Surveillance and Tracking System (BSTS), along with the Space Surveillance and Tracking System (SSTS). BSTS later evolved into the Advanced Warning System (AWS), and eventually into the Follow-on Early Warning System (FEWS). This latter system is now intended primarily to replace DSP, and its development is managed by USAF. IBM FSC at Boulder is a subcontractor to Lockheed for this program, and has overall responsibility for ground processing.

More recently IBM FSC at Boulder, as a subcontractor to Rockwell, has begun system design work, including algorithm prototyping, for Brilliant Eyes, which is part of SDI's GPALS program.

APL has been used as an algorithm prototyping language on all these programs from the beginning. Other potential roles for APL on these programs, e.g., in the software engineering and development processes, remain to be defined.

APL2 Programs for Direct Military Use

IBM FSC at Boulder recently began a contractual effort with Lawrence Livermore National Laboratory to provide new analysis programs for use by USAF Space Command analysts. Some of these programs are or will be written in APL2. Some are derived from existing APL2 analysis programs already in use within IBM FSC at Boulder. Others are being written and delivered in APL2 because they are initially regarded as experimental prototypes.

So far there has been no pressure to translate these programs to a language which is more familiar to their analysts. Indeed their analysts have

shown considerable interest and ability to work directly with the APL2 programs, and even to modify or enhance the programs in nontrivial ways.

Prototyping for an MPP Application

USAF Space Command continuously maintains a catalog of several thousand space objects, from highly valued satellites to very small and inconsequential objects such as space debris. Currently it requires many thousands of observations per day to maintain this catalog and keep it current.

Because of computational resource limitations, it has not been possible to maintain this catalog with Space Command's most accurate orbit propagation algorithms. Therefore, the orbits of only highly valued satellites are maintained with the greatest accuracy. Orbits of other objects are maintained using faster but more simplified algorithms.

Significant advantages would accrue from maintaining all such orbits very accurately. One advantage is that fewer daily observations would be required, likely enabling USAF to reduce significantly the manpower and expenses necessary to collect and manage the lower volumes of sensor data. Another advantage is that for valuable or critical satellites, potential collisions with other space objects could be predicted more precisely and further in advance as a matter of course.

As originally suggested by the Chief Scientist of USAF Space Command, this application is very well suited for a massively parallel processing (MPP) approach. In the limit, one might even apply one processor to each space object, independently propagating the orbits of all space orbits and simultaneously keeping all the processors productive. In principle either a SIMD or a MIMD approach could be made to work efficiently and well.

Accordingly, the IBM Federal Systems Company has been working with space propagation experts within USAF Space Command to parallelize their more accurate orbit propagation algorithms and to estimate MPP resources necessary to accomplish this goal.

Using USAF Space Command's existing Fortran prototype code as a base, we have extracted the core of the processing, transformed it so as to achieve better intellectual control, translated it into APL2, parallelized it for a SIMD architecture, and exercised it on an IBM RISC System/6000 workstation

against a catalog of several thousand space objects. And we have estimated the resources required to maintain the entire space catalog using the most accurate orbit propagation methods. A description of this work is contained in Reference 2.

Conclusions

We have used APL on the Defense Support Program and other military surveillance programs for data analysis, for algorithm prototyping, for display prototyping, for translation of APL prototype code to software production languages, and for financial analysis and scheduling. APL has enabled our algorithm development organizations to overcome intense schedule pressures while maintaining high quality. And APL has enabled precise communication of complex algorithmic and engineering designs in cases where documents containing English text and mathematical equations have been inadequate for the job.

Finally, we have begun to provide APL2 analysis programs on workstations for direct use by military analysts. And we are using APL2 to study and prototype massively parallel processing solutions to potential military applications that are too numerically intensive for traditional serial approaches.

Acknowledgements

The APL operation described above is probably one of the largest and most sustained in history. Following are current and former employees of the IBM Federal Systems Company at Boulder, CO (and previously at Westlake Village, CA) who have contributed to our surveillance programs through the use of APL, translation from APL to production languages, and/or technical support of APL. The author apologizes for any omissions.

Fred Badik
Howard Bergstrom
Mike Bowen
Dave Brown
Don Butler
John Curiale
Joe Donegan
Linda Feigel
Dick Gormley
Neils Hansen
Bruce Haake

Mike Bailey
Ilsa Bohn
Chuck Brans
Margarita Bruther
Bob Campbell
Roy Dent
Erich Feigel
Howard Gantz
Bruce Griffith
Ron Harrison
Charlie Hart

Sheila Hershey
Doug Hunt
Gene Kennedy
Eric Klementis
Mark Kressin
Ron Lass
Peter Lee
Jim McBride
Milt Marasch
Bill Martin
Bob Miles
Marc Munger
Ken Myers
Carmen Nater
Marlin Nielson
Mike Peper
Brent Robbins
Jim Roecker
Van Romine
John Simoncic
Steve Sipocz
Don Tadehara
Don Tork
Jane Wall
Barry Wible
Paula Young
Bruce Zuver

Ray Hoppes
Dave Kelley
Roger Kiwimagi
Ken Klementis
Joanne Lapierre
Dave Lattimore
Brad Lindquist
John Mantey
Rick Marsh
Mick Mickelson
Dan Most
Eric Myers
Mary Myers
Ryan Nguyen
Joe Pachuta
Gary Phillis
Howard Robbins
Emily Rudd
Jim Russell
Karen Singkofer
Jerry Smith
Sue Thomas
Mike Trafton
Dick White
Sam Van Winkle
Andy Yao

References:

1. Gilman, Leonard and Rose, Allen J. 1974,
APL An Interactive Approach, Wiley, New York, NY
2. Marsh, R.A., Rudd, J.G. and Thomas, S.G.,
Tracking of Space Objects using Massively Parallel Processing,
SPIE's Signal and Data Processing of Small Targets,
12-16 April 1993
3. Martin, James 1982,
Application Development Without Programmers,
Prentice-Hall, Englewood, NJ

4. Rudd, Jack G. 1986,
APL Graphics Representation and Analysis of Space-Based Observations,
APL Quote Quad, APL86 Conference Proceedings,
Vol. 16, No. 4, pp. 86-95.

5. Rudd, Jack G. and Klementis, Eric M. 1987, APL to Ada Translator,
APL Quote Quad, APL87 Conference Proceedings,
Vol. 17, No. 4, pp. 269-283.

6. Rudd, Jack G. and Brown, James A. 1990, Toward a Common Prototyping
Language, APL Quote Quad, APL90 Conference Proceedings,
Vol. 20, No. 4, pp. 322-330.

APL , une approche matricielle pour la détermination des taux zéros-coupons

par **H. RALISON**
Consultant financier & Informaticien

H. RALISON est actuellement consultant financier et informaticien chez CENTAUROD. Il intervient dans les banques et assurances. Il pratique APL depuis une dizaine d'années. Il est responsable de l'équipe en charge du développement des composantes objets réutilisables en APL dans le domaine des mathématiques financières

Résumé

Au -delà de la présentation des principes de détermination des taux zéro-coupons, l'objectif de cet article est de mettre en exergue la puissance d'APL à traiter des calculs actuariels. Une des méthodes pour évaluer un actif financier est la méthode «zéro-coupon» Elle repose sur la décomposition des cash flows futurs en flux zéro-coupons distincts, actualisés à un taux spécifique selon l'échéance. Pour déterminer les taux zéro-coupons, plusieurs techniques existent: méthode itérative, méthode matricielle. C'est avec cette dernière méthode que nous allons illustrer la supériorité d'APL à résoudre les problèmes complexes et ses différents aspects à savoir : programmation vectorielle, programmation matricielle.

MotsClés :

APL, langage, actuariat , marché financier, assurance, méthode, programmation vectorielle, programmation matricielle, informatique, mathématique, finance, banque, assurance.

Généralités

Nous rappelons que dans les marchés financiers, les opérateurs ont pour habitude d'élaborer la courbe de rendement nécessitant le calcul des taux zéro-coupons à partir des taux d'intérêts ou prix observés sur le marché, afin d'évaluer les prix et les sensibilités des produits de taux.

L'intérêt de disposer d'une gamme des prix zéro-coupons $\{B_i\}_{i \in I}$ est de pouvoir évaluer à l'instant présent n'importe quelle chronique de cash-flows $\{Cf_i\}_{i \in I}$ de la façon suivante :

$$NPV = \sum_{i \in I} B_i \cdot Cf_i$$

NPV sera appelée valeur présente de la chronique de flux $\{Cf_i\}_{i \in I}$

avec $B_i = (1+i_n)^{-n}$ facteur d'actualisation pour la maturité n
 i_n taux zéro-coupon

Dans les calculs financiers, on préfère souvent utiliser les facteurs d'actualisation plutôt que les taux zéro-coupons. Etant donné la dualité existante, déterminer l'un ou l'autre est indifférent.

Introduction au calcul des taux zéro-coupons

Par type de produit, le calcul de la courbe zéro-coupons impose la connaissance des taux pour toutes les maturités. L'opérateur financier se trouve donc dans l'obligation de définir lui-même les taux manquants. Habituellement, on procède par interpolation linéaire.

CALCUL D'UN TAUX ZÉRO-COUPON SUR LE COURT TERME

En général, les produits négociés sur le Court Terme sont des instruments zéro-coupons. Un prêt monétaire est en effet remboursé en même temps que les intérêts à la date d'échéance. Le calcul du taux zéro-coupon est alors un simple problème de conversion de base. Il suffit de ramener le taux en une base actuarielle.

Dans le cas d'un taux monétaire, la formule habituelle de conversion en base actuarielle nous fournit le taux zéro-coupon:

$$z = \left(1 + \frac{i_m \cdot m}{360}\right)^{\frac{1}{n}} - 1$$

avec i_m taux monétaire
 m nombre de jours exact du placement
 z taux zéro-coupon ou actuariel
 n fraction d'année du placement en base Exact/Exact

On a de façon équivalente, le facteur d'actualisation :

$$r = \frac{1}{1 + \frac{m \cdot i_m}{360}}$$

CALCUL DES TAUX ZÉRO-COUPONS SUR LE LONG TERME

La plupart des instruments sur le long terme comportent en général des coupons intermédiaires, l'opérateur financier est donc amené à déterminer la valeur actuelle nette de ces coupons intermédiaires et à les soustraire du prix actuel du placement. L'actualisation de ces coupons se fait grâce aux taux zéro-coupons des années précédentes. Pour déterminer le taux zéro-coupon d'une maturité n , il faudra donc connaître auparavant les taux zéro-coupons des maturités $n-1$, $n-2$,

Pour résoudre ce problème récursif, on peut utiliser deux techniques:

- la méthode itérative
- la méthode matricielle

Nous étudierons surtout cette dernière méthode en supposant que nous disposons d'une série de N instruments actuariels détachant de coupons annuels C_i pendant un nombre d'années i . Il s'agit par exemple de N obligations émises aujourd'hui au pair. L'objectif est de calculer les N taux zéro-coupons et les N facteurs d'actualisation r_n (n de 1 à N).

Le taux zéro-coupon en fonction des $(n-1)$ taux zéro-coupons précédents peut être déterminé par la formule suivante

$$i_n = \left[\frac{1 + C_n}{1 - C_n \sum_{k=1}^{n-1} \frac{1}{(1 + i_k)^k}} \right]^{\frac{1}{n}} - 1$$

En pratique, on raisonne directement à partir des facteurs d'actualisation, sans calculer les taux zéro-coupons.

Sachant que $r_i = \frac{1}{(1 + i_k)^k}$

La formule ci-dessus permet de calculer directement les facteurs d'actualisation sur l'ensemble de la courbe:

$$r_n = \frac{1 - C_n \sum_{k=1}^{n-1} r_k}{1 + C_n}$$

METHODE MATRICIELLE

Il est intéressant de présenter les calculs précédents sous la forme de matrices et de vecteurs. Reprenons la forme ci-dessus. On peut l'écrire sous la forme

$$C_n \sum r_k + (1 + C_n) \cdot r_n = 1 \quad \text{pour } n \in [1 \text{ à } N]$$

Cet ensemble d'équations forme un système de N équations linéaires à N inconnues r_n

Il peut s'écrire sous la forme matricielle suivante :

$$[A][r] = [1]$$

avec $[A]$ matrice des flux des instruments couponnés

$[r]$ vecteur des facteurs d'actualisation

$[1]$ vecteur unité

et $[A]$ est définie par

$$A_{ik} = 0 \quad \text{si } i > k$$

$$A_{ik} = C_i \quad \text{si } i < k$$

$$A_{ik} = 1 + C_i$$

Sous forme détaillée, nous avons

$$\begin{bmatrix} 1 + C_1 & 0 & 0 & 0 & \dots & 0 \\ C_2 & 1 + C_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ C_n & C_n & C_n & \dots & 1 + C_n \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \\ \cdot \\ r_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \cdot \\ 1 \end{bmatrix}$$

Les facteurs d'actualisation se résolvent matriciellement par

$$[r] = [1][A]^{-1}$$

Résolution du système d'équations ci-dessus par APL

Nous allons illustrer avec APL la résolution du système de N équations à n inconnues ci-dessus ainsi que la formule qui permet de déterminer les taux zéro-coupons sur le Court Terme.

Nous commençons par ce dernier car elle se traduit telle quelle en APL.
Zéro-coupons Court Terme :

$$z = \left(1 + \frac{i_m \cdot m}{360}\right)^{\frac{1}{n}} - 1 \quad \text{formule actuarielle}$$

$$r = \frac{1}{1 + \frac{m \cdot i_m}{360}} \quad \text{facteur d'actualisation}$$

```
[0]TauxZéro,,ZéroCouponsMonétaireAd;TauxMonétaire;NbreJours;Base;FacteurActu
alisation
[1] © Auteur    Actuariat Financier
[2] © Fonction  Détermination des taux zéro-coupons Court Terme
[3] TauxMonétaire NbreJours Base,,Ad
[4] FacteurActualisation,,÷1+(0.01×TauxMonétaire×NbreJours)÷360
[5]TauxZéro,,100×, 1+(1+(0.01×TauxMonétaire×NbreJours)÷360)*Base÷NbreJours
[6] TauxZéro,,TauxZéro,TauxZéro,[1.5]FacteurActualisation
```

Commentaires:

La ligne 4 du programme traduit la formule définissant le facteur d'actualisation

La ligne 5 les taux zéro-coupons monétaires

Nota Bene:

Toutes les variables utilisées dans ce programme peuvent être de toute structure : scalaire, vecteur ou matrice

Exemple d'exécution du programme

```
ZéroCouponsMonétaire (12 12 12 11.5 10.75 10) (1 28 89 181 273
365) 365
```

```
12    12.93546839    0.9996667777
```

12	12.87404863	0.9907529723
12	12.73805699	0.9711880868
11.5	12.00249644	0.9453409136
10.75	11.04640998	0.92462389
10	10.13888889	0.9079445145

Zéro-coupons Long Terme

$$C_n \sum r_k + (1 + C_n) \cdot r_n = 1 \quad \text{pour } n \in [1 \text{ à } N]$$

$$[A][r] = [1]$$

Sous forme détaillée, nous avons

$$\begin{bmatrix} 1+C_1 & 0 & 0 & 0 & \dots & 0 \\ C_2 & 1+C_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ C_n & C_n & C_n & \dots & 1+C_n \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \cdot \\ r_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \cdot \\ 1 \end{bmatrix}$$

- [1] TauxZeros,,ZeroCouponsLongTerme tauxcouponnés;MatTaux;vRo
- [2] © Auteur Actuariat Financier
- [3] © Fonction Calcul des taux zérocoupons et des []facteurs d'actualisation
- [4] © correspondants sur le long terme
- [5] © Technique de Calcul ... Méthode Matricielle
- [6] © pour n¹[1;N] on Cn^x(x/¹2i)+(1+Cn)x¹2n=1
- [7] © sous la forme matricielle [A].[¹2]=[1]
- [8] © avec [A] = matrice des flux des [9]instruments couponnés
- [10] © [¹2] = vecteur des facteurs [11]d'actualisation
- [12] © [1] = vecteur unité
- [13] © et [A] est définie par :
- [14] © Aik = 0 si i > k
- [15] © Aik = Ci si i < k
- [16] © Aii = 1+Ci
- [17] © Maturité TauxCouponnés
- [18] © 1 an 10.14
- [19] © 2 ans 9
- [20] © 3 ans 8.5
- [21] © 4 ans 8.35
- [22] © 5 ans 8.3
- [23] © 6 ans 8.28
- [24] © 7 ans 8.25
- [25] © 8 ans 8.27
- [26] © 9 ans 8.28
- [27] © 10 ans 8.3
- [28] © Usage ZeroCoupons 10.14 9 8.5 8.35 8.3 8.28 8.25 8.27 8.28 8.3
- [29] vRo,¹2,tauxcouponnés
- [30] MatTaux,,((¹4vRo)^o.%o¹4vRo)x³(2/vRo)¹20.01xtauxcouponnés © construction de A
- [31] (1 1³MatTaux),,1+1 1³MatTaux ©
- [32] FacteurActualisation,,(vRo¹21)Ž MatTaux © calcul de ¹2

[33] TauxZeros,,(100x⁻¹+(÷(vRo^{1/2}1)Ž MatTaux)*1÷¼vRo) © calcul tauxzéros
 [34] TauxZeros,,tauxcouponnés,TauxZeros,[1.5]FacteurActualisation

Commentaires

Ligne 1 à ligne 28 du commentaire
 Ligne 29 la variable vRo reçoit le nombre des taux couponnés en entrée
 Lignes 30 & 31 détermination de la matrice [A] Les intructions utilisées dans ces 2 lignes sont :
 Apl Opérateur externe ArgG °.f ArD
 Transposition ³ArD
 Dimension ArG ½ ArD
 Transposition dyadic ArG ³ ArD
 Ligne 32 calcul des facteurs 'actualisations ArG Ž ArD cette instruction Apl permet de résoudre un système linéaire ou d'une manière monadique (avec un seul argument) de déterminer l'inversion d'une matrice
 Ligne 33 calcul des taux zéro-coupons même instruction que ci-dessus

ZeroCouponsLongTerme 10.14 9 8.5 8.35 8.3 8.28 8.25 8.27 8.28
 8.3

10.14	10.14	0.907935355
9	8.949266421	0.8424640533
8.5	8.425136626	0.7845309219
8.35	8.275390452	0.7275803576
8.3	8.23182363	0.673325589
8.28	8.218775845	0.6225644221
8.25	8.189318115	0.5763805472
8.27	8.223522049	0.5314062907
8.28	8.242625902	0.4902472035
8.3	8.275586828	0.4515382424

Conclusions

A travers ces deux exemples, nous avons montré que Apl est un des très rares langages qui permettent de traduire des traitements complexes de manière simple et efficace en un très peu de lignes d'instructions. Ceci est dû au fait que APL possède des instructions de base permettant des transformations complexes

sur les données (inverser une matrice, transposer une matrice, créer un tableau à N dimensions, faire un tri , cumuler des données dans un tableau à 0, 1, 2...N dimensions,). L'une des caractéristiques importantes d'APL est de traiter les données en bloc (vecteurs, matrices, tableau à N dimensions..). Ainsi l'utilisateur d'APL peut-il , mieux qu'avec tout autre langage, traiter un problème comme il le pense.

De par sa nature, Apl est un langage à notation mathématique qui permet de résoudre d'une manière efficace les problèmes actuariels tant dans le domaine financier que dans le domaine de l'assurance et également dans d'autres domaines (Gestion, PAO, DAO, Systèmes Experts,.....).

Apl est un des outils très prisés dans les banques & assurances anglosaxones dans le domaine actuariel (J.P Morgan, MerryLynch, Barclays Bank, OCDE, REUTER.....)

La quadrature du cercle? ...c'est 'possible' : plus précis que le palmer!

par Michel J. Dumontier

J'ai longtemps cru comme tout un chacun, que la quadrature du cercle à la règle et au compas était 'impossible'. D'autre part, j'ai eu un professeur de physique qui nous interdisait de mettre plus de 2 décimales à π dans les calculs, disant à juste titre que les appareils de physique sont précis en général à 10^{-1} et que ceux précis à 10^{-2} sont très chers. Même si les appareils de mesure sont maintenant plus précis, il n'en est pas moins vrai que même le compas à pointe sèche et vis sans fin, avec ou sans balustre, ne dépasse pas la précision du palmer⁽¹⁾ ! alors pourquoi priver les élèves que nous étions et encore plus ceux de maintenant, de faire cette construction magique ?, d'autant plus que l'on sait maintenant que l'univers est discret, on peut, à partir d'un certain rang, mettre allègrement les décimales à la poubelle, surtout lorsque, de plus, la théorie du chaos, les fractales et tutti quanti, font rage depuis un certain temps

⁽¹⁾ le palmer travaille à 1 micron près ($1/1000^{\text{ème}}$ de millimètre) sur des distances ne dépassant pas 10 cm: d'où une erreur relative de 10^5 .

1) La formule magique!

Les anciens, qui n'avaient pas de calculette ou d'ordinateurs, croyaient que π était exactement égal à $6/5\Phi^2$, Φ étant le nombre d'or. On les comprendra (Voir article revue AFAPL N° 16 page 87 sur le nombre d'or et sa relation avec la suite de Fibonacci).

Je pense que cette formule était connue dans l'Egypte ancienne : ils avaient même baptisé le rapport $6/5$, le nombre d'Osiris (1.2).

Puisque nous avons une calculette à 17 chiffres, profitons-en

Valeur de $\sqrt{\pi}$:

CEPP_{,17}
PI_{,±1} ^a CE_{,RPI},PI*0.5
1.7724538509055158

Valeur de $(\sqrt{6/5})\times\Phi$

$\Phi_{RPI} = 0.5 \times (1 + 5 \times 0.5)$
 1.618033988749895
 $\Phi_{RPIA} = \Phi_{RPI} \times (1.2 \times 0.5)$
 1.772467428896755

Quelle est la précision relative du calcul

$(RPIA - RPI) \div RPI$
 0.0000076605612226216272
 $7.66E^{-6}$: c'est mieux que la précision du palmer

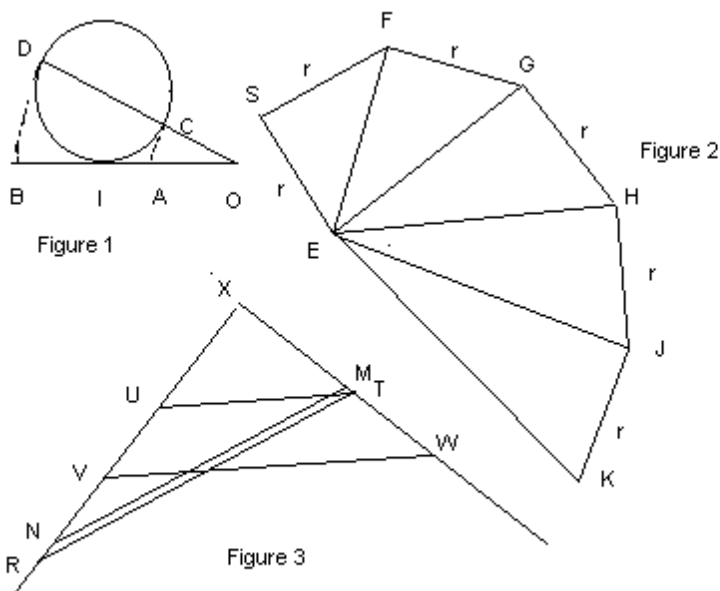
2) La quadrature du cercle, en terme de surface, à la règle et au compas

Si on trace un cercle de rayon r , sa surface est πr^2

Il s'agit donc de tracer un carré de côté $\sqrt{\pi} r$

Or, la formule approchée de $\sqrt{\pi}$ est $(\sqrt{6}/\sqrt{5}) \times \Phi$

Il suffit maintenant de se souvenir de son cours de troisième (d'il y a cinquante ans ou quarante mais pas moins hélas !, les choses fondamentales ne s'enseignent plus !)



a) Construction du nombre d'or

On se souvient (j'en faisais allusion dans la revue AFAPL N° 16 page 92) : rappel de la classe de troisième : partage d'un segment en moyenne extrême raison. Dans la figure 1, si on trace $OI=r$, et un cercle de diamètre r (et non $2r$) tangent en I à OB . On trace OCD passant par le centre du cercle, on a $OI^2=OC \times OD$. Les cercles de centre O passant par C et D coupent respectivement OI en A et B . On a alors $OA=OC$ et $OB=OD$ et $AB=CD=r$.

En posant $x=OA$, on a : $OI^2=OA \times (OA+AB)$ ou $r^2=x \times (x+r)$ ou $x^2+rx-r^2=0$ et $OA=x=r \times (-1+\sqrt{5})/2 = r\varphi$ et $OB=x+r = r(1+\sqrt{5})/2$, donc $OB = r\Phi$.

NB : pour $r=1$, $x=\varphi$ est solution de l'équation $x^2+x-1=0$, donc φ est lié par la relation : $1-\varphi=\varphi^2$.

De même que A partage le segment OI en moyenne extrême raison, I partage le segment AB dans les mêmes proportions. A et I partagent aussi le segment OB dans les mêmes proportions. Le segment OA est moyen entre les segments AI et OI : $\Phi=OA/IA=OI/OA$; $OA^2=OI \times IA$, car en les remplaçant par leur valeur on a $OA^2=(r\varphi)^2=r^2\varphi^2$, $OI \times IA = r \times (r-r\varphi) = r^2 \times (1-\varphi)$, or on sait (5 lignes plus haut) que $1-\varphi=\varphi^2$, donc $OI \times IA = r^2\varphi^2 = OA^2$.

Pour ne rien oublier, rappelons les relations entre φ et Φ :

$$\Phi-1=1/\Phi=\varphi ; \varphi^2=1-\varphi ; \Phi^2=1+\Phi$$

La figure 2, est celle bien connue de la construction (en escargot) de \sqrt{N} .

Les triangles ESF , EFG , EGH , EHJ , EJK , ont un angle droit aux sommets S, F, G, H, J respectivement et on a $EF=r\sqrt{2}$, $EG=r\sqrt{3}$, $EH=r\sqrt{4}$, $EJ=r\sqrt{5}$ et $EK=r\sqrt{6}$.

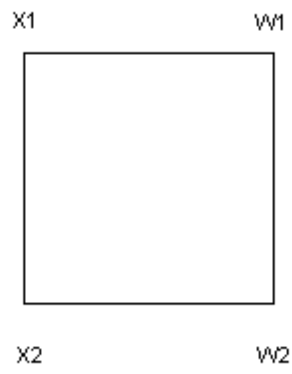
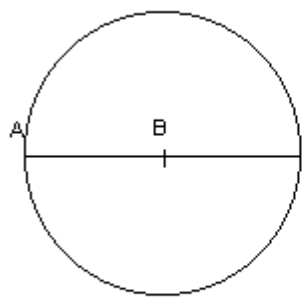
b) Construction de $r(\sqrt{6}/\sqrt{5}) \times \Phi$

Dans la figure 3 on a $XM=r$, $XN=EJ=r\sqrt{5}$, $XR=EK=r\sqrt{6}$

En traçant RT parallèle à NM connue, on obtient le point T tel que $XT=r \times (\sqrt{6}/\sqrt{5})$

De même si $XU=r$ et $XV=OB=r\Phi$, en traçant une parallèle VW à UT , on aura $XW/XV=XT/XU$, donc $XW=XT \times XV/XU = r \times (\sqrt{6}/\sqrt{5}) \times r\Phi/r = r \times (\sqrt{6}/\sqrt{5}) \times \Phi$, ce qui est le côté du carré recherché.

Dessignons la solution



—

3) La quadrature du cercle, en terme de longueur, à la règle et au compas

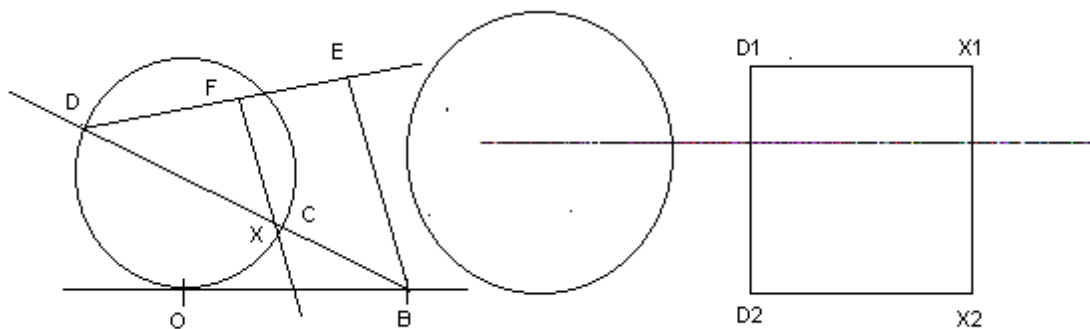
Calcul de l'erreur relative du calcul

$$\begin{aligned} &PIA_{,}1.2 \times PHI \times PHI \\ &CE_{,}ER_{,}(PIA-PI) \div PI \\ &0.000015321181129503586 \end{aligned}$$

Cette fois l'erreur relative, au lieu d'être égale à $7.66E^{-6}$, elle est égale à $1.53E^{-5}$. La précision est de l'ordre de celle du palmer.

La longueur du cercle étant de $2 \times r \times \pi$, il s'agit donc de tracer un carré de côté $r \times \pi / 2$ soit de côté $(3/5) \times r \times \Phi^2$

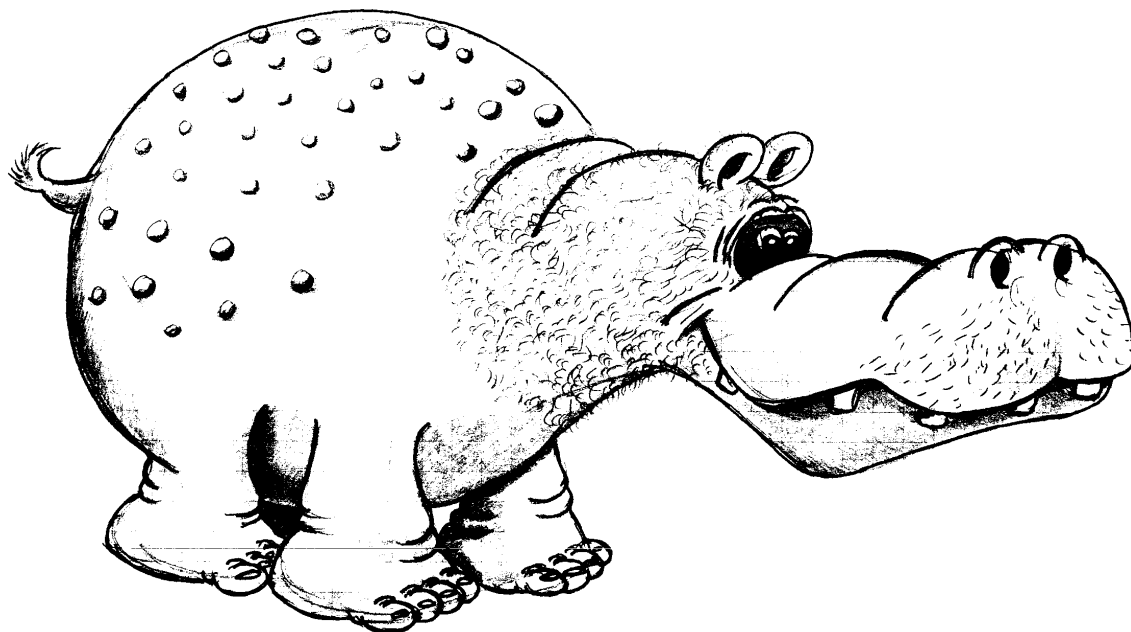
On reporte donc la valeur $OB = r \times \Phi$ et on trace un cercle de diamètre $r \times \Phi$ tangent en O à OB. On trace BD passant par le centre du cercle. BD vaut donc $r \times \Phi^2$. Sur une droite passant par D, on trace une valeur DE proportionnelle à 5 (exemple 5cm) et une valeur DF proportionnelle à 3. Il suffit maintenant de tracer FX parallèle à EB et nous aurons donc $DX = 3 \times DB / 5 = (3/5) \times r \times \Phi^2$. Il suffit alors de tracer un carré de côté $D1X1 = DX$. On a tracé à gauche de ce carré, un cercle de rayon r pour la comparaison.



NB. le rapport théorique des 2 carrés (linéaire/quadratique) est $(r \times \pi / 2) / r \times \sqrt{\pi} = (\sqrt{\pi}) / 2 = 0.88622692545275792$. Le rapport pratique est $((3/5) \times r \times \Phi^2) / r \times (\sqrt{6} / \sqrt{5}) \times \Phi = 0.8862337144483776$

L'erreur relative sur ce rapport est 0.0000076605025389982064

Même les Egyptiens n'avaient pas, en architecture, une précision de 7 microns sur une distance de 10 mètres (voir erreur relative sur le côté du carré pour la quadrature en terme de surface) alors, en ce qui concerne nos règles et nos compas, ne soyons pas plus royalistes que le roi ! et donnons à nos élèves des problèmes concrets et des solutions concrètes, plutôt que des impasses irréelles et irréalistes.



Un autre dessin de Bernard Legrand. Selon les superstitions russes les « béguémotik », c'est le nom affectueux de cet animal en russe, apportent argent et bonheur...