Contents

Quick reference diary		2
Glorious summer	Stephen Taylor	4
News		
Industry news: APL2000, Dyalog, Kx Systems, MicroAPL		7
Dyalog 2008 at Lo-skolen, Denmark	Adrian Smith	19
GSE & APL-Germany Fall Meeting	Adrian Smith	27
Discover		
Structured Storage and Monitor Expressions	David Liebtag	35
ISO 9000 Certified APL development	Chris Hogan	41
Unicode support for APL	Morten Kromberg	52
Learn		
Rain flips its q	Adrian Smith	71
The year 1997	Neville Holmes	82
Spice for beginners	Dan Baronet	89
Congratulations not in order?	Stephen Taylor	99
Suffer the little children…	Norman Thomson	100
Profit		
Cauchy curves	Bill Jones & Cliff Reiter	109
About polynomials	Gianluigi Quario	121
Quick calculation of Kendall's Rank Correlation Distribution	Gordon Sutcliffe	131
Consultants		137
Subscribing to Vector		144

Quick reference diary

15-16 Apr	Talinn, Estonia	Baltic APL Seminar
7-9 May	Stuttgart, Germany	Spring APL Meeting
11 May	Mountain View, CA	Bay Area Users' Group
8-9 Jun	Reading, Hants.	British APL Association 2009 conference
13-16 Sep	Princeton, NJ	Dyalog 2009

Dates for future issues

Vector articles are now published online as soon as they are ready. Issues go to the printers at the end of each quarter – as near as we can manage!

If you have an idea for an article, or would like to place an advertisement in the printed issue, please write to editor@vector.org.uk.

The British APL Association Announces

BAPLA 2009

'APL in the real world'

We are very pleased to be able to announce our 2 day APL conference in June 2009. Our aim is to show you where things are going and what you can look forward to plus what others are doing and how.

Interested? Want to know more? Then please visit

www.bapla09.com

Where all will be revealed.
We all look forward to our first conference in many years and hope that you will be able to join us.

A big thank you to all our sponsors;

[kx] bca DYALOG APL2000 the CARLISLE group Stylusnet optimasystems CATALYST

EDITORIAL

Glorious summer

The effects of the collapse of the international financial system this winter are still rumbling through the economy. With revenues in freefall, firms are slashing costs and shedding workers. Already Britain has two million unemployed.

Wintry times are always summer for the APLs. With budgets frozen, managers abandon IT doctrines for tools and people who deliver quickly and cheaply. The times are ripe again for the Direct Development methods in which the APLs have always performed well.

Go get 'em. In this vein we are holding a 2-day conference – BAPLA 2009 – in Reading in June. This is an opportunity to introduce a new generation of business managers to what Direct Development makes possible.

The BAA London chapter now meets every month. See the APL Wiki for their agenda; join their Google Group baa-london to keep in touch.

We have resurrected from the defunct Product Guide a listing of companies and individuals offering consulting services in the APLs – see the back pages. This listing is free to BAA members and non-members alike. Contact Ray Cannon for changes and additions.

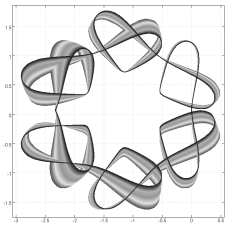
We know many *Vector* readers valued the Product Guide as a community 'white pages' for former colleagues, but a little study showed our listings were not up to date. Web applications such as Facebook and LinkedIn now do this job much better. For personal listings we recommend homepages on the J Wiki or the APL Wiki. And LinkedIn and Facebook both have Iversonians groups you can join. All of these can be found from the Community page at vector.org.uk.

Ian Clark, our projects officer, has brought to press the first title under the Vector Books imprint. Over the years, Eugene McDonnell wrote 41 articles for his *Vector* column on J. The first edition of *At Play With J* reproduces the articles as originally printed, and can be ordered from Lulu.com – see the Books page of our website. Because J has evolved since the first article appeared, volunteer editors are revising now for a second edition,

in which the examples match the current version of J. Contact Ian if you can lend a hand.

Catherine Lathwell is making a documentary film about the story of APL. Find her project blog *APL Diaries* through the Community page at vector.org.uk and see if you can offer her any help. Catherine will be speaking about her project at the APLBUG meeting on 11 May at the Computer History Museum in Mountain View, CA.

In this issue, Chris Hogan explains how to negotiate the ISO 9000 certification some clients might demand you have, without losing your Direct Development agility. Morten Kromberg shows Dyalog in the ecumenical world of Unicode and Windows IMEs. Inspired by q and Paul Mansour's flipdb, Adrian Smith whips up an in-memory DBMS. Dan Baronet introduces Spice, an addition to the Dyalog IDE, and David Liebtag describes new features of APL2.



The mathematicians are out in force. Bill Jones & Cliff Reiter marvel at Cauchy curves. In J-ottings 51, Norman Thomson has a go at the kids' homework. In Italy, Gianluigi Quario completes his meditations on polynomials. And Gordon Sutcliffe reports a fast way to calculate Kendall's rank correlation frequency distribution.

Stephen Taylor

N E W S

Industry News

from Sustaining Members
APL2000, Dyalog, Kx Systems, MicroAPL, Optima Systems

APL2000

2008 conference round-up

On November 10-11, 2008 APL enthusiasts from across the United States and seven other countries gathered at the Hyatt Regency Hotel in Bethesda, Maryland for the 2008 APL2000 User Conference.

The conference hotel was conveniently located with easy Metro access from the airport and to downtown Washington D.C. The nation's capital offered many opportunities for sightseeing, viewing exhibits at the Smithsonian Institution's museums, fine dining and shopping. Many attendees took advantage of special weekend rates and arrived early to spend a few extra days in the area.

Attendees represented industry leaders from a broad spectrum of business, engineering and science. APL2000 has a significant presence in the fields of finance, insurance, healthcare, aerospace, engineering, employee benefits, airline and automotive industries among many others.

Each conference attendee received a conference bag including an APL2000 conference shirt, flash drive with the conference proceedings and a copy of Ajay Askoolum's book, *System Building with APL+Win*.

For two exciting days the attendees immersed themselves in informative sessions and participated in stimulating discussions surrounding new developments in APL2000 products, particulary APL+Win, VisualAPL and APL WebServices.

APL EDUCATION

John Estep presented his *pro-bono* APL Workshop program, which trains high school students interested in programming using the APL language. Mathematics, software design and engineering are also emphasised. APL2000 and other APL

APL education

programming-language vendors have provided John with books and software for his students. John encouraged other APL programmers to use his course as a model for regional APL programmer training and development.

APL+WIN SESSIONS

John Walker's presentation, "What's New in APL+Win" highlighted the developments in the APL+Win product including:

- Enhanced Unicode support in APL+Win, for instance, the ability to convert between Unicode code points and various representations, such as UTF-8 and APL+Win ⎕AV.
- Localisation level specific referencing, the ability to reference functions and variables at specified localization levels.
- Enhanced printing and print preview in the APL Grid.
- Major overhaul and refactoring of the behaviour of tooltips in APL GUI objects.

There have been 10 releases from Version 6 in 2006 to Version 8.3 in 2008.

Ajay Askoolum's presentation, "C# As the GUI and APL+Win Supporting the Business Rules" illustrated how to interface APL+Win with mainstream technologies including:

- Creating a C# web service which relies on APL+Win to support multiple clients needing access to application-specific business rules.
- Exposing a .Net assembly as a COM for use with APL+Win.
- An introduction to the many examples of APL+Win interfaces in Ajay's book, which was distributed to conference attendees
- An HTA (HTML Application) using APL+Win as a COM Server.
- An additional, detailed example was provided by Ajay as part of the conference materials – a C# Windows Service using APL+Win as a COM Server – with SQL Server Express as the data tier, accessed by C# using ADO.NET and by ADO from APL+Win.

Eric Lescasse's presentation, "Interface APL+Win and .NET (C#)", also illustrated how to interface APL+Win with mainstream technologies including:

Eric Lescasse & Ajay Askoolum

- Consuming C# DLLs from APL+Win using the NetAccess product. It was announced at this conference that current APL+Win Subscribers will receive a licence for the NetAccess software, installer and documentation. NetAcess is a convenient tool to prepare C# DLLs that expose Microsoft .Net features for access by APL+Win. NetAccess opens the power of the .Net framework to APL+Win users, including the Microsoft .Net visual objects which can now be embedded in APL+Win forms.
- Porting your existing APL+Win application to the .Net environment with a C# WPF or Ajax GUI, an APL+Win calculation engine on the server and the project deployed over the Internet as a client-server .Net (C#) ClickOnce application.

Joe Blaze's presentation, "Improved Efficiency of Execution of APL Primitives", illustrated the new APL+Win interpreter enhancement, which a programmer can activate to utilise an alternate order of execution for APL+Win expressions involving multiple primitive operators. For large size, double data type arguments, significant performance increases can be achieved. The alternative order-of-execution option can data-fetch and store overhead.

Jeremy Main presented "Reverse Geocoding with APL". APL functions are used to access GPS position data and various public databases to establish the relative location of geographical elements, enterprises, parks, public places and other points of interest.

Jeremy Main

APL WEBSERVICES SESSIONS

Joe Blaze presented the APL WebServices sessions:

- Overview of the APL WebServices product from inception in 2002 to 2008 as a high performance, high-reliability tool to interface APL+Win functions and workspaces with web-based clients.

Joe Blaze

- Exposure to APL WebServices features for scalability, load balancing, request queueing, multi-threading and stateless/stateful client-server interaction were illustrated.
- Health, underwriting and financial examples were provided, illustrating APL WebServices on the client side supporting business rules and creating client-ready output. These examples also illustrated client-side GUI creation using HTML, PDF or C# WPF forms.

- An additional example was provided which uses APL WebServices to efficiently organize a grid of computers to perform stochastic analysis for a health-related application system. Extensions of this model to other monte carlo simulations is easily done.

VISUALAPL SESSION

Jairo Lopez presented the VisualAPL session:

- The VisualAPL version now available for Microsoft Visual Studio 2008 was demonstrated. VisualAPL also remains available for Microsoft Visual Studio 2005. VisualAPL remains the only APL implementation which is fully integrated with Visual Studio and produces fully-managed CLR output. The performance of VisualAPL is outstanding. VisualAPL programmers can take advantage of extreme memory (including 64-bit addressing), multi-threading and all .Net data types.
- The Cielo Explorer interactive session and scripting engine included with VisualAPL was demonstrated as unique among Visual Studio add-ins because it enables easy exploration of all of Microsoft .Net without the overhead of building and compiling an entire project. Line-by-line entry of APL or C# code in the Cielo Explorer is immediately processed and the results displayed within the Cielo Explorer window. In addition Cielo Explorer scripts can be incorporated into a .Net namespace and compiled into a .Net assembly, if desired by the programmer.
- A health and underwriting example was provided using a C# WPF GUI form and a VisualAPL .Net assembly to perform the analysis and prepare the output.

OVERVIEW OF APL2000 MISSION AND LICENSING OPTIONS

Sonia Beekman made this presentation.

APL2000's mission

APL2000 is a full-service software company that develops and markets APL products for the Win32, .Net, Unix and Linux operating system environments. APL2000 also provides consulting services to assist in the design, development, implementation, conversion, update, enhancement and deployment of APL-based application software systems.

APL+Win is the product for customers with an investment in Win-32 based programming. The APL+Win Subscription Program has been very well received by APL2000 customers, and APL2000 remains dedicated to the development and maintenance of this product.

optimasystems

APL Programmer position in the UK

Optima Systems Ltd in the UK is currently on the look-out for a full time APL programmer/analyst to join our current team. We offer an attractive package working in a friendly environment and can consider applications from all skill levels as training can be given. If you are interested and want to find out more then please e-mail your C.V. in the first instance to paul@optima-systems.co.uk or by post.

Contact details can be found on our web site.

01293 562700

info@optima-systems.co.uk
www.optima-systems.co.uk

VisualAPL is the product for customers interested in an APL implementation which can fully exploit the features of the Microsoft .Net framework and is integrated with the mainstream Microsoft Visual Studio 2005/2008 interactive development environment.

APL WebServices is the product which provides an easy and effective way to use APL+Win code to support the calculations and business rules for a web-based or .Net graphical user interface. APL WebServices is the high-performance, high reliability, .Net-based Web Services application to expose an APL+Win based software application to any Web-connected user.

APL2000 product licensing options

APL+Win subscriptions will now include Lescasse Consulting's NetAccess.

APL+Win subscriptions will continue to provide unlimited 'runtime' use. Seven subscription levels are available to best meet the APL+Win needs of the customer.

A no-cost education version of APL+Win is available for classroom use by instructors and students.

The Professional and Enterprise versions of VisualAPL will now include unlimited production use as part of the licence. 'Production' use in the Visual Studio environment means distribution of the compiled output of a VisualAPL product. This 'production' use is analogous to 'runtime' use in the Win32 environment. A lower-cost desktop version of VisualAPL remains available for those programmers not requiring distribution of their output beyond their desktop.

No-cost, full-featured, limited-time-period, demonstration and evaluation versions of VisualAPL and APL WebServices are available.

Further information on APL+Win, VisualAPL, APL WebServices and other APL2000 products can be obtained on the APL2000 web site, www.apl2000.com or by contacting sales@apl2000.com.

MONDAY EVENING DINNER CRUISE

Monday evening, conference attendees were given a bus tour of the Washington D.C. monuments on their way to the waterfront. There they boarded the "Capital Elite", a 70-foot yacht chartered by APL2000 for a private 3-hour dinner cruise down the Potomac River. The boat provided a cozy atmosphere to wine, dine, network and chat with friends. The main deck was set for dinner and offered hors d'oeuvres followed by a delicious buffet. The upper deck had a bar and lounge area with comfortable sofas and cocktail tables. Although the weather was a bit nippy, a few brave souls ventured onto the outer decks to take in the fresh air and enjoy the view. A great time was had by all.

Cruising on the Potomac

APL MEMORY LANE

The closing session provided a trip down memory lane with an APL Jeopardy Game prepared by Kevin Weaver.

NetAccess from Lescasse Consulting now included with all APL+Win subscriptions

NetAccess opens the power of the .Net Framework to APL+Win. All APL+Win Subscription licensees will receive NetAccess software, the installer and documentation, as part of their subscription benefits.

NetAccess, developed by Lescasse Consulting, is a convenient tool to prepare C# DLLs that expose Microsoft .Net features for access by APL+Win. NetAccess uses APL+Win's ActiveX interface to great advantage by exposing the user-selected methods, properties and events of the C# DLL as ActiveX elements.

With NetAccess, you can write Microsoft .Net C# DLLs to be used with ⎕wi from APL+Win. This enables you to tap the power of .Net. For example the Microsoft .Net visual objects can now be embeded in APL+Win forms.

This exciting new feature is provided to our APL+Win subscribers at no additional cost, and reflects APL2000's commitment to APL+Win and to our customers. To further support NetAccess users, a separate section on the APL Developer Network Forum has been established for NetAccess related discussions.

For further information, or to purchase APL2000 products, contact sales@apl2000.com or call Sonia Beekman at +1 (301) 208-7150.

Dyalog

Dyalog 2008 at a glance

As we're rapidly approaching the end of a *very* eventful year it seems there has been something new almost monthly. Let us take the opportunity to re-cap some of the highlights.

October 2008 – Dyalog '08 User Group Conference

This year Dyalog hosted its most successful user conference ever in Elsinore in Denmark. More than 100 delegates (plus guests) attended the 2½-day conference and many also took part in the 1½ days of training and workshops on offer. Dyalog is already in the process of planning the Dyalog '09 conference – currently scheduled to take place near New York in the US in the early autumn of 2009. If you were unable to attend, would like to refresh your memory, or perhaps see one of the sessions you were unable to catch, take a look at:

- Adrian Smith's conference review in this issue
- This year Dyalog taped most of the conference sessions and you can access the slideshows with full audio from www.dyalog.com.
- Andrea 'Tony' Grignani and Stefano 'Wildheart' Lanzavecchia have put together a photo Gallery show from the Conference featuring Tony's beautiful pictures supported by original music composed by Wildheart.

You can find their YouTube show at uk.youtube.com/watch?v=lW552GGVQxk.

- Do not miss the wonderful presentation John Scholes made at Dyalog '08 where he took a look at Dyalog through 25 years and made a plea for simplicity. You can find John's presentation at uk.youtube.com/watch?v=qSVR4Z3DA24.

September 2008 – LearnAPL

In the previous issue of *Vector* we featured the joint apprenticeship scheme launched by Dyalog Ltd and Optima Systems Ltd. This initiative is aimed at an individual looking to expand their knowledge into the exciting world of APL. The apprenticeship scheme will hopefully attract interesting candidates over the next couple of months and since this is part of Dyalog's ongoing activities with regards to attracting new programmers, we would like to highlight the initiative again. See more at www.dyalog.com/pdf/LearnAPL.pdf.

August 2008 – Version 12.0.3

Dyalog announced the commercial availability of Dyalog Version 12.0.3, including support for secure sockets. Version 12.0.3 is available 'off the shelf' for Windows 32 & 64, AIX and Linux. Solaris and other Unix versions are available on request.

Version 12 Manuals are available as free-of-charge PDF downloads or as print-on-demand from stores.lulu.com/dyalog.

Two new public workspaces (APL2IN and APL2PCIN) were also made available for download. You can get them from www.dyalog.com/version12.html.

June 2008 – New Dyalog team member – Richard Smith

In June the Dyalog Development Team welcomed Richard Smith. Richard landed firmly on his feet in Dyalog and he quickly started excellent – and independent! – work on Journalling Component Files. The first result of Richard's work was a presentation held at Dyalog '08. Hear and see it at winweb.dyalog.com/Dyalog08/RichardSmith_JournalingFiles.html.

May 2008 – New Board member, Michael Holmberg Andersen

Michael joined the Dyalog Board of Directors in May. He has worked in SimCorp A/S since 1991 and currently holds the position of Senior Vice President for the IMS Development Department, where he is responsible for SimCorp's software-development activities.

April 2008 – Dyalog@25

The Dyalog APL language celebrated its 25th anniversary in April, where Dyalog Ltd hosted an open-house day in the office in Bramley on Friday the 11th, followed by a more formal event on the 12th. The Dyalog story has further been published as a booklet which was distributed with the previous issue of *Vector* to BAA members, and to delegates at the Dyalog '08 Conference. You can get a PDF version from www.vector.org.uk/archive/v234b/d25.pdf.

February 2008 – Dyalog Version 12.0.1

At the end of February Dyalog announced the commercial availability of Dyalog Version 12.0 for 32-bit Windows, with Unicode Support. At the same time the company launched two new online libraries for Help and Documentation. You can peruse the new libraries at www.dyalog.com/documentation and www.dyalog.com/help. (Both these sites include the full version 12.0 and 11.0 Release Notes.)

Kx Systems

Kx Systems and Nagler & Company have announced a strategic partnership agreement covering the German-speaking areas of Europe, primarily Germany, Austria and Switzerland as well as Benelux and Poland. Nagler has formed a new company, Symagon, specifically to work with Kx and to provide market data solutions to European financial institutions. Symagon is a wholly owned, Germany-based subsidiary of Nagler & Company.

The main drivers for forming the partnership were the deep market knowledge of Nagler's consultants and their successful track record of working with Kx and implementing kdb+ at financial institutions, most notably in Germany. The partnership will see Symagon provide sales, consultancy and support services to firms in Benelux, Germany, Austria, Switzerland and Poland.

Says Jens Rick, managing director of Symagon: "Having worked with Kx on a number of market data projects for the last three years we decided that we wanted to formalise what has become a very productive working relationship. We found kdb+ to be far superior to other systems on the market in terms of the speed of implementation, performance and cost of ownership. We are very impressed with Kx's dedication to its customers, its excellent standard of client support and the continuing enhancement of kdb+."

Nagler & Company first started working with Kx in 2005; from the first joint project it was clear to both firms that together they offered a winning

combination of talent, experience and location. Symagon specialises in market data, providing consultancy that spans all phases of planning, development and implementation and covers quantitative analysis, structuring, algorithmic trading, market making and risk management.

Janet Lustgarten, CEO of Kx Systems, says: "Symagon consultants have a thorough understanding of our philosophy and of our products, kdb+ and q. They have many years' experience in our core markets and have worked on numerous projects with Kx. Symagon will help us to extend our presence in Europe and provide a high level of expertise to financial institutions in the region."

MicroAPL

At MicroAPL we have always worked hard on making new versions of APLX backwards-compatible, believing that new versions of the interpreter should not break existing APL applications if at all possible.

As an example, consider APL programs written the Apple Macintosh. Our original APL product, then called APL.68000, was released in 1986 and included a number of then-innovative workspaces for GUI programming. At that time, a typical Macintosh had an 8MHz 68000 processor and 512K of memory.

Over the years, Apple made the transition from 68000 to PowerPC processors, then changed the operating system to the modern OS X, and finally changed processors again to use the Intel x86 architecture.

Despite this – and despite the fact that the APL.68000 interpreter was originally written in 68000 assembler – you can still load an original APL.68000 workspace into APLX. All the original APL code will run, and even most of the original GUI functions still work (although we would not recommend them for new programs).

One of our APLX users recently demonstrated the importance of this approach in a very exciting way. Dr Glenn Schneider from the University of Arizona's Steward Observatory is an umbraphile – literally a 'Shadow Lover' – who travels all round the world taking spectacular photographs of solar eclipses.

The August 2008 solar eclipse was visible from the Arctic Ocean, with the path of totality close to the North Pole. Dr Schneider viewed the eclipse from an Airbus A330-200 flying high above Svalbard, with his cameras pointing out of the aircraft windows. The software he used to control the cameras and help navigate the aircraft is written in APLX running on a Macintosh. It was originally

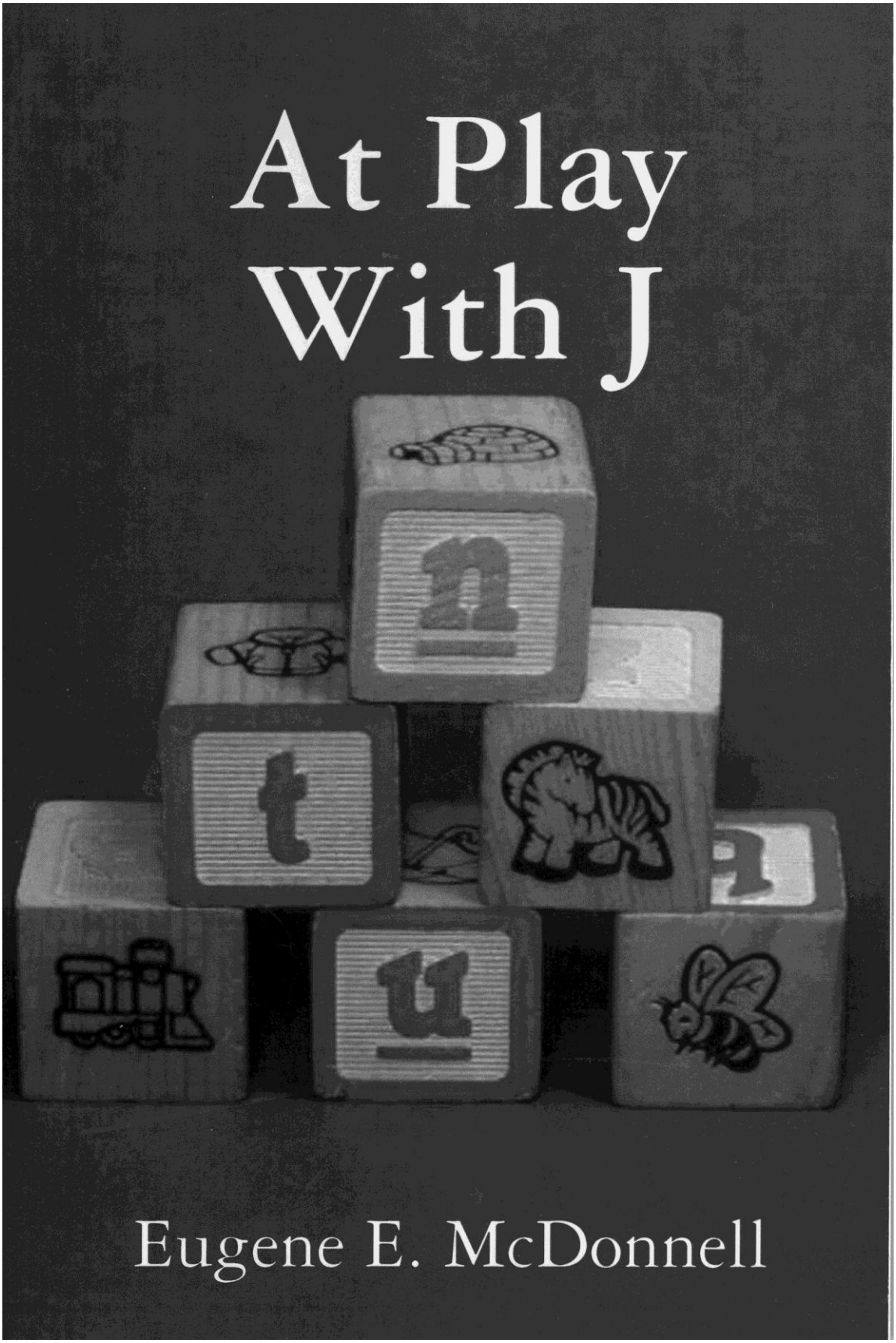
implemented in APL.68000 and still makes use of GUI features we introduced back in 1986.

As a testament to the enduring strength of APL as a language, readers might like to note that some of the core algorithms were originally implemented in 1974 in APL on a Xerox Sigma 9 computer under the UTS (and later CP-V) operating system.

To see some pictures of the eclipse and scenes from inside the Airbus, visit nicmosis.as.arizona.edu:8000/ECLIPSE_WEB/ECLIPSE_08/TSE2008_FLIGHT_FD /TSE2008_FLIGHT_DECK_SETUP.html

Looking to the future, the development team at MicroAPL is currently working hard on APLX Version 5 which will be released in 2009. Although we already have a long list of new features, it's not too late to suggest anything you would like to see added to APLX.

Please send your ideas to aplxv5@microapl.co.uk. We can't promise to implement everything in this release but we're always interested to hear your ideas.



From Vector Books: first edition available now at Lulu.com

MEETING

Dyalog 2008 at Lo-skolen, Denmark

reported by Adrian Smith
adrian@apl385.com

Setting the scene

Back to Denmark, for what everyone seemed to agree was the best Dyalog gathering yet. The delegate list numbered over 100 for the first time, and the sun shone for the Viking challenge. The quality of the food was almost too good – I for one had to go for a very minimal lunch to keep my wits about me for the afternoon sessions.

On target for the 12th century

The presentations covered a wide range of topics, and the feedback forms showed there had been something good for everyone there, from the talk by Charles Brenner (APL content zero) to the talk by Roger Hui (APL content close to 100%) – both were highly rated, and by very different groups of attendees. As always, the slack time was well used, and I walked past lots of small groups clustered around laptops in the various coffee areas.

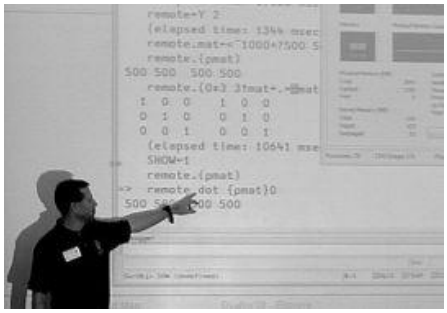
The recreation opportunities were welcome, although I mourned the 9-pin bowling. But there were woods to walk in, a tough little par-3 golf circuit (50kr including hire of equipment), table-tennis, skittles/pool, table football and an easy walk down to the beach if you wanted to smell the sea air from a little closer. All told, it is hard to beat Lo-skolen as a venue for a group of this size.

Major sessions

Opening keynote

This was all about usability, consolidation and gradually expoiting the new multi-core hardware that is coming our way. Morten showed us how to use the new logging capability of the interpreter, and asked everyone to give it a run, and

send in the logfiles for Dyalog to analyse. This looks like a great way to find the remaining bottlenecks in the interpreter, and will point Nicolas and Roger Hui at the most productive speedups.


Morten Kromberg's keynote address

Here is Morten, illustrating a simple Conga server which gives the impression of APL running a genuine *parallel each* operator. The CPU buckets very satisfactorily both fill up and both empty at the same time. The ability to spread load across cores like this will become increasingly useful as we go from dual to quad cores and on upwards.

Morten briefly showed the Unicode chessboard again, and stressed the benefits of moving across to the new interpreter family as soon as possible. He also outlined a true 'managed code' project which will one day allow a new build of Dyalog to run anywhere a .Net process is allowed to run. It will be lean, mean, and will tick all the required boxes for implementation in highly secure environments. It will definitely not carry forward any baggage from the current Dyalog, so is unlikely ever to replace it completely as an application development tool.

Apart from these announcements, it will all be about getting the bug-list fully under control, and making it possible to work with really big classes in a productive way. Morten pointed us to John Daintree's talk on the IDE as well as to Nicolas Delcros and Roger Hui who have both been working to improve the efficiency of the current interpreter, and at Richard Smith who has been helping John Scholes to make great strides in eliminating `FILE DAMAGED` by adding journalling to component files.

Charles Brenner on "Forensic Mathematics"

Dr Brenner has almost single-handedly established a whole new discipline of 'Forensic Mathematics' and (naturally) is a world-respected expert in the field. Having seen a prior run of this talk at Naples a few years back, I knew a fair bit of the background, but this was still a fascinating insight into the world of DNA matching, as well as Bayesian likelihood and all the other stuff you need to know to make a reasoned judgement based on DNA 'evidence'.

Charles Brenner on DNA-View™

I think the key element I had missed from his previous talk was that all DNA matches are made between rather few highly specific *loci* in the junk DNA. Obvious when you think about it – we are highly likely to match exactly in the DNA that actually does something useful, as anyone who didn't would be unlikely to make it into the world at all! Look out for his paper in a forthcoming *Vector*.

John Daintree on the IDE for Dyalog 12.1

This was also rated 'most useful' on many of the feedback sheets. Classes are a great new capability, but (as many of us have quickly noticed) they could use some help for serious development – the editing environment is nowhere near as comfortable as the traditional namespace full of functions. Typically (see flipdb, for example) you have a skeleton class and a companion namespace with all the code in it, which the class `⎕IMPORT`s whenever it gets instanced. Alternatively you go the whole hog and generate/fix/instance an entire script from a your namespace on the fly.

Both of these are *kludges* and should be wiped from the map, which is what John has set out to do by making the editor respect the content of a class in a much better way. He showed the 'obvious' things working again – you can just say `)ed myclass.foo` in the session and get straight to the function – and there may be some extras like the ability to roll up sections of code the way Visual Studio does. For now, it would be good just to have the basics – if I can go `)ed myclass.newfn` and have it show up in the script, add stuff, double-click on a new name within it, get another new function, and so on, then I'm a 95% happy bunny.

Roger Hui, Nicolas Delcros and Richard Smith on APL internals

These three talks were closely related, and all promised simple incremental improvements in performance. John Scholes has a watching brief on all of these areas, but is was good to see the 'new boys' let loose in the core of the interpreter.

- **Roger** has been working on speedups to common boolean operations like `≠\boolmat` and has discovered that by playing with the byte alignments, he can quite easily write simple APL expressions that beat the raw primitives for speed. Read his script (on the key disk) or await the full paper to see a very neat use of the new LCM capability that came in with Dyalog 12.

- **Nic** has been investigating *matrix indexing* which was written in the days when the entire interpreter had to fit in a 64k segment, so takes very few

optimisation choices indeed. In the case where one or more axes are elided, it builds the entire index set in memory, before iterating around it. Clearly it can do much better with high-rank arrays by taking the elided ranks 'as read' in an extra layer of iteration. Nicolas showed some good speed improvements, and even the pathological case with a 15-dimensional array was only a little slower.

- **Richard** (there were two Richard Smiths at the conference – this is the one who recently joined Dyalog) has been taking over existing work on component file reliability. By first writing a journal of a pending update, he allows the file to recover from nearly all the bad things that can happen when data is written. By adding an optional level of explicit cache flushing, he even overcomes the pathological cases where the operating system has actioned the file writes in a random order, so you never know what failed when the power was pulled.

This block of talks was concluded by Jonathan Manktelow with a quick demo of the latest .Net toys (called *Windows Presentation Foundation* or WPF for short) which implement screen-management in a much more web-oriented way. Interestingly the world is moving towards a *flow-layout* model where you arrange your fields and buttons using simple nested groups ({*this* above *that*} beside *these*) and the system is responsible for the detailed layout, depending on the space available, the fonts chosen, and so on. The convergence with Gary Bergquist's `ZarkWin` framework is very close (it was good to see Gary at a Dyalog conference for the first time) and I am seriously wondering about a modern edition of CPro which merges Gary's screen-layout tools with the 'observer pattern' engine that I think is the key ingredient of CausewayPro.

25 years of Dyalog with John Scholes

The usual mix of deadpan humour, interesting history, and jokes for programmers like the carefully timed demo of `⎕DL ¯5`, which he must have been practising. The *hammer sketch* was all his own work, and will stick in lots of memories, for the quality of the acting as well as the originality of the script.

John Scholes on display

25 years is actually a very long time in the software industry, and it was interesting to see how quickly the key moments get blurred into history. When did *namespaces* first appear?

Which APLnn meeting saw a Windows GUI for the first time? John had left plenty of gaps in the chronology, and the audience found it hard to fill in all of them. The relationship between the known 'hard' dates (when people joined and left, when the company address changed) and the key moments in the life of the interpreter proved very hard to pin down. Even the special Vector supplement (a good read, by the way) leaves quite a lot of detail to be filled in by some enterprising local historian.

Day 2 – User presentations, the Viking challenge and the Banquet

Most of these talks are on the conference keydisk, and will very likely appear in future issues of *Vector*, so better not to attempt a summary here, I think. The feedback forms mentioned most of them as 'useful' so I think the programme must have worked well. I was rather sad that my RainPro session collided with Morten on Conga, as this was something I would have definitely attended, given the chance.

Anne's airborne axe

As you can see from the picture, axes were thrown again this year. There were some rather less physical challenges too, mostly to see how well the groups could function as a team. One task involved everyone holding a rope and pulling as hard as they could in opposite directions. Now *there* is a true metaphor for APL development!

And so to the banquet, with the usual great food and quality entertainment (this time from a champion accordion player). Here are a couple of shots from the rogues' gallery to give you a flavour:

Chala from the Carlisle Group, with a serious camera!

Vibeke chatting with Gary Bergquist

The challenge is on to find a venue as good as this one anywhere else in the world, starting with the Eastern USA for next year. It will be tough, even in the more relaxed parts of Europe, and (apart from Minnowbrook) I don't think I can pick out any stateside venue that comes close. If anyone can think of one, let Vibeke know!

Day 3 – closing sessions and afternoon courses

Tommy Johannessen and his lunch server

Tommy writes software the way he throws axes – see opportunity … *charge!!* … which may get a few people hurt along the way but sometimes comes up with something really great. This talk showed us how to order our school meal on the internet, along with over 25,000 regular users of his ASPX-based system. Along the way he has bounced off all the nasty hidden features of the interface between the web server and Dyalog, and he has bullied all of them into submission. Wonderful stuff.

Gert Møller and his logic engine

This is another great use of APL, originally for prototyping but increasingly as an implementation tool for crashing straight through huge (and I mean *really* huge) constraint networks. This is based on some core work that was done many years back in APL, and Gert has been patiently waiting for some key patents to expire so that he can start over with a new company ready to exploit some very neat compression algorithms in APL. I get the feeling that Dyalog is beating C++ for speed on many of the crucial operations – it would be great to throw some of his code at our C# translator to see if there is a 'best of both worlds' solution out there.

Romilly Cocking returns to APL

Romilly was one of the founders of the BAA and co-owned the most successful APL consultancy business in the UK for many years. Then he quit the APL world for Smalltalk and made a pretty good living out of that, too, before exploiting his experience in developing with dynamic languages to win a reputation as a coach for 'agile' Java teams. Now he's back to APL and says it's better than ever before.

His talk was a very clear and simple introduction to genetic algorithms (GAs for short) and to some of the uses they can be put to. The core code was about 25 DFns and fitted very nicely on one slide. So that made two talks in one conference on DNA, and they complemented each other beautifully.

Wrap up

Morten closed with a slide of all the things they would love to shake out of the interpreter, like the underscored alphabet, ⎕SM, and external variables. As you would expect, almost every oddball feature was used by at least one person in the audience, so this is going to be a tough call, but signalling the intention can't be a

bad thing. Then it was off to one last lunch, and some final training sessions for the true stalwarts.

The morning light over Lo-skolen

If you missed any of it, many of the sessions were fed through 1024×768 screen-capture and should come with high-quality audio, so you can watch them again on the Dyalog web site.

GSE and APL-Germany Fall Meeting, Augsburg, November 24-25 2008

reported by Adrian Smith
adrian@apl385.com

Around the City

Augsburg dates from Roman times, and where ancient buildings have survived (mostly by virtue of being below ground-level) it is very impressive. The cathedral had a quiet simplicity most unusual in a region generally associated with the extremes of baroque. As always, the centre of town is for pedestrians only (and the occasional tram) so it was a real pleasure to walk back from the restaurant in the gently falling snow after a typically warming Bavarian meal.

The undercroft of the cathedral

The joint meeting was really well attended, so much so that we slightly overflowed the seminar room on the second day. Thanks to Dittrich and partners for setting it all up in their home town, and for paying for the magnificent flagons of locally-brewed *dunkelbier* on the second evening. Much good work was done, and all of the talks had interesting material to keep us awake.

The GSE requirements day

In the past, I have tried to come to these joint meetings for the second day only, as Day 1 is to some extent a private conversation between IBM and its most loyal customers. However there was a most interesting talk from Sandvik which I really wanted to hear, and the summary of "what's new in APL2" is intended for public consumption, so I tagged along for the morning session.

The Sandvik CAPP system, shown by Gunnar Jörtsö

This was the first time anyone from Sandvik has been allowed to talk about the system outside the company, so we were very privileged to see it. The 20-page handout would make an excellent paper in its own right, but I doubt we will get permission to publish more than some very brief notes here!

The system is implemented on the mainframe with GDDM drawing, and overlayed text. Engineers create the specifications for components in a visual flowchart, attaching 'COBOL-like' logic at each process stage. The specifications are (very carefully) checked, 'compiled' into APL2 functions which generate the CAD drawings to drive the machine tools directly. This can take the time from specification to manufacture down to a few hours, rather than the many days which would be involved in processing the drawings manually.

It is *big* in any terms, with around 1,500 users, who have built some 25,000 flowcharts which compile into over 2.5 million lines of APL2 code. The engineers still like the mainframe interface, but for the sales users (who configure new components while talking to customers) find a web front-end gives that modern look and feel. There are currently 10 servers running with a load-balancing master server handling the 20,000 daily browser requests from the web interface.

A few years back in Florida, we were shown a wonderful APL2000 system which simulated mineral processing, down to the level of detail that you need if you are thinking of opening a copper mine. Twice now, we have been given a walkthrough of the petroleum-chemistry simulator that Mobil research wrote in Dyalog APL to configure oil refineries in the most profitable way. Well, now there is a new companion for both these applications – the Sandvik system is at least as wide in scope, at least as business-critical, and rooted just as firmly in APL. Long may it stay that way!

David Liebtag on new things in APL2

David has been quietly working away at making the APL2 session more comfortable for developers, to some extent in response to previous GSE days, and

sometimes just because he likes a new idea, and it is easy to do. Some of the things that have been bugging his users are quite interesting:

- Commercial rounding has been speeded up by quite a large factor (×11 on integer data, ×3 on float) which could be beaten easily on Intel hardware by using the on-chip instructions if this were still not adequate.
- Reading LF-delimited files just got nearly 1000 times faster for big files. (A simple change of approach can work wonders).
- The toolbar with the APL symbols opens a stand-alone documentation window (rather than just showing a transient tooltip) from a right-click. I think I would like this better than the Dyalog tip – you can copy/paste the examples and maybe park the window somewhere for easy reading later!
- Hitting Ctl+Enter on an expression throws the output into a simple editor window, rather than into the session. This is a very nice feature, as you can easily stuff your session with a huge array you really didn't want to see all of!
- You can set a trace or stop on all labels in one keystroke, and F7 in the editor throws out unused locals.
- You can quickly save a selected part of the session log to a named file, handy for sending off for debugging support.

We also got a very quick overview of the structured storage, which has been slightly enhanced to allow you to move any array (so untyped data) into the external storage, for reasons which would become obvious when David showed us the new *monitor* capability on Day 2

Day 2 – User presentations at the APL Germany Day

Adrian Smith shows 'Dyalog for students'

This talk will get a full paper in a forthcoming *Vector*, so for now just a quick summary:

- Typing stuff is becoming *cool* again. The top 5% of students who may get to like APL are the rebels who relish the chance to get some real power over the machine. One of the coolest developer tools around is Microsoft PowerShell, which just makes the command prompt into a full-blown hackers toolkit. We need to get after these guys with APL!

- There was a time when APL *was* cool. It was the only place where it was easy to type in some numbers, `)SAVE` what you typed, make some simple analysis, and print out the results neatly (good old ⍕) with the minimum of fuss. 'Dyalog for students' is an attempt to recapture some of that old simplicity.

- The biggest problem is Microsoft Excel, which makes it so easy to type in the numbers, then makes it almost impossible to do anything useful with them. Microsoft Access does let you do useful things with data, but is impossible for mortals to get it moving. Adrian proposes a step back to something that looks like a command prompt, lets you easily enter and save data (so it has a well-hidden but powerful database engine), positively encourages you to analyse it with real he-man statistical tools (thanks to Alan Sykes and Ellis Morgan), charts it (now this really is quite cool) and helps you make your report if you find Word too boring and LaTeX too geeky.

I think this made a hit with the meeting, although Morten qualified my estimate of 5% down to 1%. This is still enough to be worth chasing – there was a time when almost all the students from a German Technical University had met APL, and this makes the job of recruiting new blood into the industry that much easier. I think I will get some good support from several directions if I can make this stuff work seamlessly with Dyalog.

David Liebtag on Monitor expressions with Processor 15

David Leibtag on new features in APL2

AP15 started as a way of enforcing typed arrays, mainly so APL2 can interface cleanly to external routines which expect strict data typing. However it has a few extra capabilities, which David showed us around.

- Once notified to P15, a variable has some attributes, for example you can find (or set) its *address* in memory, which gives you an easy way to define little Union structures by overlaying two arrays at the same address.
- New to this release is the Monitor attribute. This may be set to any executable expression (for example `⎕SI ⍳0` which will run whenever the variable is assigned. This has obvious debugging uses, but David did point out that there is quite an overhead in

moving data to the external storage, so you should not think of using this in production code for large arrays.

- The monitor expression runs in the same way as code executed by ⎕EC so it cannot change the variable which triggered it. Seems sensible enough!

I think this will be very popular – when you are trying to track an obscure bug in a big mainframe application (written 20 years ago by a team of programmers who left the company long ago) then having a handle on all the assignment points to some rogue variable can be an enormous help. The drumming on the tables was exceptionally enthusiastic for this one!

Actually, you could abuse this quite nicely by using little scalars (say a version number) to track updates, and having a monitor on this variable trigger updates elsewhere in the system. It is probably really easy to tell the programmer "when you are done, just increment ∆version" rather than having to call some particular follow-on code. Just a small step in the direction of event-based programming.

Bernd with the image album

Dr Reiner Nussbaum

Emulating ⎕WI in APL2 by Bernd Geißelhardt

This was a nice little gallery-maker, but the impressive thing for me was that it was converted from an APL2000 original using cover functions to mimic ⎕WI in APL2. Bernd reckoned that with about 900 lines of (very tedious) code he can do a good percentage of the basic Windows toolkit. Obviously a full-blown generic conversion kit would be a lot, lot harder.

Unix for grown-ups by Dr Reiner Nussbaum

The code behind this talk is all being shared on SourceForge, so search for unix4aplers if you want more details. Basically it is an attempt to transfer a bit of APL thinking to the tasks system programmers do on big Unix boxes. Jobs like load balancing can be handled very well by the APL ? if all the tasks are roughly the same weight. Reports showing the disk space taken by files more than *x* days (or years or whatever) old can benefit from the typical APL calendar routines which convert from ⎕TS form (as reported by the

Unix system time) to decimal Julian days.

Remarkably, Reiner could not find any good (i.e. working) examples on the Net to do this job, so he carefully hand-translated an existing APL calendar library to Perl to do the work.

Trying to manage huge arrays in a limited workspace by Holger Walliser

How do you program a recursive summation on a 5-dimensional matrix when the size gets above 250Mb? For the lucky few on 64-bit APL systems on workstations, the answer is obvious – just buy enough memory! On the mainframe the job is much harder, and Holger took us through a few approaches, none of which worked particularly well.

At the end, David Liebtag suggested a more radical approach, using simple sequential files (the data is a homogeneous floating-point matrix) where it should be possible to make a fairly simple cover function to 'map' the file in bite-size pieces back into the workspace very efficiently. Maybe at the next meeting, we will see if the problem can be solved this way!

Themes for 2009 by Morten Kromberg

Dyalog left the Elsinore conference with the best of intentions (stop working on new stuff, get the reliability up, document what we have) with the 12.1 release expected quite early in 2009. It sounds as if the big customers have just come strongly back behind APL, are recruiting new APL staff, and (of course) have a list of new features which they really, really need. Oh bother, here we go again, said Morten, looking pleased, but slightly hassled. So look forward to 12.1 a little later than planned, but definitely with a much shorter list of outstanding bugs.

In the few spare hours available to him, Morten has been working on a simplified version of Stefano's WildServer (known for now as the *MildServer*) which allows APLers to interact really easily with data coming in from web forms. This looks as if it is midway between Pete Donnelly's original 'demo' server (which has been running Starmap and a few related Causeway sites like CUSP very quietly for a little over 10 years now) and the serious object madness offered by the WildServer technology. For simple-minded APL folk who just want to get on the Ajax bandwagon, it has a lot to offer, and it will be offered as an open-source project on the Dyalog website. Incidentally one of the good things about saving your APL code outside the workspace as Unicode text files is that such collaborative development becomes practical.

Wrap up

This whole meeting had quite an optimistic feel to it. APL on the mainframe is alive and well in Germany, and seems likely to stay that way for many years yet. But the APLers of the old school are well aware of new ideas, and (see for example the Sandvik application) are able to skip a generation very easily by adding HTTP support to the big old iron kit and coming out smelling of browser.

A well-attended session on the second day

The APL2 language remains almost unchanged over the decades since its inception, but the interfaces and environment are moving steadily ahead, and as long as David is around and willing to work on it, steady and reliable progress will continue. These meetings are always worth the trip, and I hope to see the same old faces (and a few more new ones) again next year.

D I S C O V E R

Structured Storage and Monitor Expressions

by David Liebtag
liebtag@us.ibm.com

This is a version of a presentation made to the APL Germany 2008 Spring Meeting, since updated to reflect the release of APL2 Service Level 13. *Ed*.

Introduction

APL2's name association facility ⎕NA allows APL programs to cause references to names to be processed by associated processors rather than the APL2 interpreter. Associated processors are supplied which provide access to files and programs written in other languages such as C, Java, and Rexx. APL2 Service Level 12 included a new Associated Processor 15, which provides access to structured storage. Service Level 13 added support for Processor 15 monitor expressions which can be used to perform data change tracing and validation.

APL is traditionally a loosely-typed language. Arrays simply contain numbers and characters and the structure and type of their data can be changed at any time. APLers usually view this as one of the language's benefits: they don't have to worry about how data is represented internally; they can focus on their problems. However, many users of other languages view strong typing as a benefit: type checking helps programmers avoid domain, rank, and length errors. Associated Processor 15 supports strong data typing and enables APL2 programmers to gain the benefits of type checking that are usually only enjoyed by programmers of compiled languages. In addition, Associated Processor 15 lets APL2 programs access data that is outside and larger than the workspace and share data with programs written in other languages. This article introduces Associated Processor 15 and demonstrates some of the ways that it can be used.

Strong data typing

Have you ever encountered an error in an application and discovered that a variable's array had an unexpected type, rank, length, or depth? How would you go about finding the cause of such an error? You might have to look at every line

that the application might have executed that assigned a value to the variable. Needless to say, in a large application, this might be tedious. Wouldn't it be nice if you could force your application to suspend at the point at which the incorrect value was assigned? That is exactly what you can do with Associated Processor 15.

Consider this expression:

```
      'I4 1 3' 15 ⎕NA 'Integers'
1
```

The expression associates the name `Integers` with a rank one array that contains three 4 byte integers. (`⎕NA` returns 1 to indicate the association succeeded.)

You can use the name just like normal:

```
      Integers←4 6 8
      Integers
4 6 8
```

But look what happens when you try to assign a value which does not match the pattern:

```
      Integers←12 4.56 17
DOMAIN ERROR
      Integers←12 4.56 17
      ^^
```

The new second element is a floating point number rather than an integer. Because the new array does not match the pattern, Associated Processor 15 immediately signals an error.

If you reference the variable, it has the last valid value:

```
      Integers
4 6 8
```

Patterns

Associated Processor 15 supports simple, nonhomogeneous, and nested arrays. The arrays may contain a wide variety of types of data including 1-, 2-, and 4-byte characters, 1-bit and 1-, 2-, and 4-byte integers, and 4- and 8-byte floating-point numbers. Character vectors may be fixed-length or variable-length null-terminated strings. Here are some sample patterns:

Integer scalar: `'I4 0'`

Character matrix: `'C1 2 3 4'`

String: `'S1 1 64'`

Nested array: `'G0 1 3 I4 1 3 E8 2 3 4 C1 1 32'`

In short, Associated Processor 15 supports the all same data types as APL2's Associated Processor 11 and the `ATR`, `PFA`, and `RTA` external functions.

Accessing data by address

Sometimes programs written in other languages refer to data by address rather than by value. Associated Processor 15 can be used to share data with these programs.

Suppose you used a variable named `ADDRESS` to hold the result of a program which returned the address of a three-integer vector. Consider this expression:

```
      ('I4 1 3' ADDRESS) 15 ⎕NA 'NotMine'
1
```

In this case, the name `NotMine` refers to the storage owned by the other program. You can use it just like before:

```
      NotMine←14 16 18
      NotMine
14 16 18
```

You can also retrieve the address of an APL2 array and pass it to programs written in other languages. Here's another association expression:

```
      ('ADDRESS' 'Integers') 15 ⎕NA 'ArrayAddress'
1
```

This expression associates the name `ArrayAddress` with the address of the `Integers` array. You could use `ArrayAddress` as an argument to a non-APL program and that program could access the array directly.

For simple arrays with types corresponding exactly to the APL2 internal data types, Associated Processor 15 can access arrays that are larger than the workspace. This enables APL2 programs to access very large arrays that are allocated by other programs.

Naming array elements

Because you can acquire the address of arrays, and you can associate names with arbitrary addresses, you can use Associated Processor 15 to name elements of arrays. For example:

```
      ('I4 0' (ArrayAddress+4)) 15 ⎕NA 'SecondInteger'
1
      Integers←3 4 5
      SecondInteger
4
      SecondInteger←67
      Integers
3 67 5
```

The expression adds 4 to `ArrayAddress` to adjust the address to the second four-byte integer.

Accessing exported variables

Sometimes programs written in other languages share data by placing it in shared libraries (or DLLs on Windows) and exporting it by name. This enables multiple programs on the same machine to all use the same storage. Associated Processor 15 supports accessing exported variables.

For example, APL2 on Windows includes a library named `aplwin.dll` that exports a 4-byte Boolean variable named `DisplayLogo`. The following expressions demonstrate how to access this variable:

```
      ⍝ Associate name
      ('I4 0' 'aplwin') 15 ⎕NA 'DisplayLogo'
1
      ⍝ Reference value
      DisplayLogo
0
```

DisplayLogo controls whether the Session Manager displays the product information dialog during invocation.

Monitor expressions

Each variable associated with Processor 15 has a monitor expression. The monitor expression is executed each time the variable is changed. Here's another association expression:

```
      ('MONITOR' 'Integers') 15 ⎕NA 'Monitor'
1
```

This expression associates the name `Monitor` with the `Integers` variable's monitor expression. You can use monitor expressions to trace variable specifications. For example:

```
      3 11 ⎕NA 'DISPLAY'
1
      Monitor←'DISPLAY Integers'
      Integers←3 4 5
┌→────┐
│3 4 5│
└~────┘
```

You can also use monitor expressions to validate values. For example:

```
      ⍝ Ensure the first element is 1, 2, or 3.
      Monitor←'⎕ES(~(↑Integers)∊1 2 3)/5 4'
      Integers←17 2 3
DOMAIN ERROR
      Integers←17 2 3
      ^^
```

A variable's monitor expression is executed after the variable's value has been changed.

Arbitrary arrays

Sometimes you want to use Associated Processor 15's support for monitor expressions, but you do not want to impose strong data typing. For these cases, you can supply an empty pattern:

```
      '' 15 ⎕NA 'Array'
1
```

Any arbitrary array can be assigned to `Array`.

Summary

Associated Processor 15 provides efficient ways to detect data errors and share data with programs written in languages other than APL2.

Further information about APL2 is available at http://www.ibm.com/software/awdtools/apl. Detailed information about APL2, Associated Processor 15, and Service Level 13's other new facilities can be found in the *APL2 User's Guide* through the Library link.

ISO 9000 Certified APL development

How to achieve Quality Assurance in a small company

or How to do the boring bits and get on with the fun stuff

Chris Hogan
chris.hogan@4xtra.com

That light-hearted subtitle is to point out from the start what this paper isn't about: it's not about making you change the way you develop APL software.

What is it that makes APL so good for software development?

Well, that is a sufficiently large and contentious topic for a raft of papers. I'll limit my remarks to a few obvious points. I'm not trying for a rigorous set of definitions, just a simple list we can all agree with:

- APL's functional richness gives compact code and high productivity:
- Which in turn leads to small teams as there is no point chopping work up if a few people can do it, not when larger teams lead to longer communication lines or adding extra developers results in information overload by rushing the development.
- The immediacy of function results makes it easy to incorporate non-technical business experts into the team to interpret and guide the developers. Not pair programming as advocated by XP perhaps, but certainly what we might call *pair development* is a common feature of APL projects. This simplifies communications at a vital stage of project evolution.
- The fact that defined functions and operators are a normal part of the invocation structure enables the developer to build rapidly from small units. When combined with the rapidity of results this means prototyping and incremental delivery are a natural way forward.
- Looking at that list we can see why APLers were pioneering modern Agile methods long before they were even called 'RAD'. My point is that

whatever our Quality Management System (QMS) does it cannot interfere with the advantages of developing in APL.

So what is ISO 9000?

ISO 9000 started life with the British Ministry of Defence: during a war they wanted to say to any factory in the land "stop what you're doing and make tanks/guns/bombs" and be reasonably assured that said tanks would run, the guns would fire and the bombs wouldn't blow up while still in the factory.

In 1979 this military standard was published by the British Standards Institute as BS 5750, with the intention of applying it to peacetime manufacturing, to improve quality control in industries such as aerospace, where variations in quality can be disastrous.

BS 5750 became European standard EN 29000, then in 1987 went global as ISO 9000 – ISO is the International Organization for Standardization in English and the Organisation Internationale de Normalisation in French: neither gives us "ISO", it's from the Greek for equal, so a link with APL already…

The way ISO 9000 worked was to require the factory, or software company, or call centre, to document, not what they made or did, but how the manufacturing / coding / telephone-answering procedures were managed, and to prove by record-keeping that the procedures were being followed.

This first version of ISO 9000 was essentially BS5750 with a different name. It had three main variants, numbered 9001, 9002 and 9003, depending on whether you originated new products, maintained products or just handled products with no concern for how they were made.

I use the word *products*, but the 9003 version proved popular with service companies, because you can define, say, a complaints procedure even though you don't actually manufacture a physical product. Call centres use this to show they are handling customers in a consistent way, even if it is badly.

The emphasis was still on conformance with procedures, so a new version was published in 1994 concentrating on quality assurance via prevention, instead of just checking final outputs, but even this didn't change the main product of ISO 9000: shed-loads of work instructions, records and an army of bureaucrats. Probably most frequent criticism of ISO 9000 was the amount of money, time and paperwork required to achieve certification. You're probably thinking "Not good news for APLers so far", but please bear with me,.

The current version, which appeared in 2000, combines the three variants into one, also called 9001, so an organization claiming to be ISO 9000 registered probably means ISO 9001:2000 as ISO 9000:2000 is actually the reference guide. The 2000 version has two main differences which should interest us.

Firstly, it makes 'continuous improvement' of the company's performance the central purpose of a QMS. Think of it as being more proactive, rather than just reactive to reviews of the final output. From the APL point of view, we can treat all business processes as an evolving prototype.

Secondly, the standard became 'parameterised': instead of different standards according for the different activities companies perform, ISO 9001 is intended to include the lot, but if you can show you don't need bits, you can leave them out and if you show you can combine bits, then combine them you can – as long as the auditor is able to find out where you've put them.

I said that everything has got simpler, but ISO 9000 has spawned a slew of specialised standards, some of which don't even use the '9000': ISO 10007 is for configuration management, which can be included in any ISO 9000 QMS for software development; 14000 is for environmental management; and to add to the confusion, health systems managers often refer to EN 29000 where there is a specific medical standard. I'm not going to go into the rest of the range, but I should mention TickIT, which is an ISO 9000 scheme to "suit information technology companies, especially software development". This paper is not about TickIT, which is a lot more prescriptive and better suited to large companies with time to waste sufficient personnel to implement it. A software company can have a certified ISO 9000 QMS for its products and services without having to use TickIT.

The earlier versions of the standard were very declarative: "the company shall establish a process to…", with little guidance on how, and even less idea of why, you were to do it. The new wording is a bit clearer and more process-driven. I would argue that developing software is a process, but equally the QMS itself can be seen as a collection of objects which change state according to a set of rules applied to them. Nothing in the standard dictates a procedural rather than object oriented approach, it is up to the developers to describe what they do and stick to it, not to change they way they work to fit the QMS. Nor is ISO 9000 hostile to the APL prototyping approach, in fact continuous improvement implies incremental delivery and refers to the QMS itself as well as the product or service quality. It is a symmetrical equivalent to the incremental development of an APL project.

Certification

An ISO certificate is not a once-off, but has to be renewed to prove to the outside world you are sticking to the procedures you've defined. How can a non-IT professional comment on your work? Well:

He or she doesn't, they just testify to all and sundry that the processes you said you would implement have been implemented, are being following and (just maybe) are having an effect.

If the auditor can follow it, then so can your clients and it should prove reassuring to them that you can prove the actions you've taken on their behalf to fix their problems or develop their code.

Contents of ISO 9001

ISO 9001:2000 is only about 30 pages long, but like anything important it's how you use it that counts. Having said you can leave out what you don't need, a QMS must have the following:

Quality Policy

A formal statement by management, usually about one page long, to commit to the QMS

Quality manual

The guidebook to all other documents, so the auditor can see what you've left out or combined

Control of Documents

How you hand out instructions and details of the QMS to your employees

Control of Records

How you record what you did

Control of Nonconforming Product or Service

What do you do when it goes wrong

Corrective Action

How you fix the problem

Preventive Action

How you stop it happening again

Internal Audits

Are you following your own standards?

Audits

A problem for the typical APL development team is that there are two types of auditing: external by a certification body and internal by company staff who are not part of the team, but how do you audit when you are only a one-man band? Or even a ten-man band, if you don't have a dedicated QA department? Like so many things today, you use the Internet.

In HMW's case, the Professional Contractors Group (PCG) offers a scheme where the QMS is kept locally, but backed up via Subversion, commonly used as a code repository by multi-site development teams. The Internal Auditor is a member of the PCG staff who can scan the records you've published and keep you on track. The external auditor can also see these records, though they often prefer it if you visit them for a brief chat. As we'll see HMW have gone beyond Subversion to produce an interactive QMS directly available to the auditors.

The PCG scheme

There is one unique feature of the PCG scheme, which is not a part of the general ISO 9000, but is mandatory for any QMS which is certified to PCG ISO 9000. This is the Code of Ethics, which is in effect a manifesto of how the company works. It has to be sent to every client and referred to in any contract. HMW also posts our copy of the code of ethics on our web site.

The PCG scheme supplies standard documents, such as the "Control of Documents" and "Control of Records" and templates for the Quality Policy and Manual. The scheme defines just two main work flows to cover the rest of the compulsory sections: finding work and doing work. A bit obvious isn't it?

Finding work

The idea is to show that you are trying to find work you are competent to do. This just keeps all your records in one place where the auditor can find them, so you:

List your clients, agencies, web sites, wherever you source your potential work from. Show that the code of ethics was included in the materials sent. Perform a simple risk assessment – -does the company have the skills and resources to do the required work? Keep copies of any quotations sent.

Doing work

Doesn't really add a lot more which is specific to this stage:

Keep copies of all contracts and other policies agreed, for example health and safety, any confidentiality agreements, etc. Any instructions give about the work you are to perform, basically log what you do, if you're not given formal specifications, a daily work log will do. Much of what a developer does is technical and therefore is outside the remit of the QMS. Invoices show you billed for work you did and any timesheets, but only if the contract requires billing by signed timesheet.

Some items, such as the confidentiality agreement, can be stored under either work flow. There is, of course a bit more about QMS reviews, training (increasing the skills of staff is as important to the idea of continuous improvement as better quality in products and service) and so on, but as far as direct work for a client goes, that's it. You should have all this information to hand anyway or do you really work with no idea of what the customer wants?

All ISO 9000 asks is that you record what it was you were asked to do and show that you did it, or at least worked on it Customer feedback is the only proof you need. If they are happy then ISO doesn't care how you made them happy, just that you record their state of happiness.

The scheme extended

The standard PCG scheme assumes the company is a service provider and therefore leaves out some aspects of the standard. This isn't a problem for developers working closely with users or to a set of requirements, that is those providing a development service, but it does preclude developing software as a distinct product.

Fear not! Procedures for these steps have been added into the PCG scheme by several companies including HMW. There are only really two missing elements: a specific design and development cycle; and a process to publish and maintain separate versions of the products.

For design the PCG requirements documents are easily expanded. ISO 9000 does not dictate what the technical content of any of these documents has to be. If you like patterns, then your procedure might be to state which pattern you are using and that it is verified by someone in your company. The auditor just has to see that one was specified and that someone bothered to verify it. You don't fail for

using the wrong pattern, that's your choice as an expert. You can fail only if you boast to the world you are using patterns and then either you don't use them or can't prove to your customer and auditor that you did.

An ISO 9000 Quality System for APL developers

HMW Computing has developed a Wiki based system using TWiki, which follows the PCG scheme structure, with optional extensions for product development, but is directly available via the world wide web, raher than Subversion. I'm not intending this section as a sales pitch, but I do have to explain some features of our QMS Wiki to show how APL fits with ISO 9000 – but if anybody wishes to talk to me later…

What do we have to do?

Not as much as you might think, as long as we do it consistently. I'm not the sort of person to add unnecessary bureaucracy to anyone's burden, especially my own. Our solution is based on the premise that if a development team has documented its work properly, then most of the effort to comply with ISO 9000 has already been done. I stress that this is a structure for a team to record everything about any type of project. So for a particular project, much of this will be unpopulated. Only a few items are mandatory to satisfy ISO 9000.

You don't have to document payroll procedures for example. Nothing stops you from doing so, but this is where companies often get bogged down, putting things into the QMS which only have peripheral impact on the work they do for the end customer. In XP terms YAGNI – "You Ain't Gonna Need It". Just stick to what's necessary to get the APL project done.

Another mistake made by many companies is that they attempt to get the QMS right first time. An auditor at a working party meeting of the PCG the other day stated that this was a very bad idea: get your QMS working and go for the audit. If nothing else it leaves room for some "continuous improvement"!

The Wiki facilities

The Wiki has pages for our client entities, which can proceed through the different states of being a prospect, a client, and sadly sometimes an ex-client. Various topics may be associated with them at different stages. Also we keep all

our suppliers in the system. The clients' staff get their own pages, so we can mange contacts with them and their responsibilities within projects.

The fact that all the project-related events are held in the Wiki made a team-wide calendar an obvious extension to the QMS. We have pages for meeting minutes and site visits, with a debrief form for that vital feedback, so we can tell if they are satisfied, well, at least as far as the QMS is concerned. All events, together with scheduled development actions and purely private events can be displayed on the calendar. We have the usual options for repeating events, appointments and public holidays. In short, the Wiki has all the features of a Customer Relationship Management System, without going over the top.

At an early stage the entity is only potentially a Client. Recording at this level is simple: a Wiki Page to record our understanding of the requirements; proof that we sent the code of ethics and (we hope) some acknowledgement that they read it (most clients don't bother); a form for risk assessment (mainly can we do the job and do we want to); and container pages to which are attached any Quotations, CVs or Confidentiality agreements exchanged.

Quotations can be as brief as an email saying something like: "It'll cost you £5,000" to do the work you describe in the attached Word document", which you send to the quotations page (there is a email to Wiki gateway)- you have to be able to prove you did quote and that the customer agreed for you to go ahead on that basis. If on the other hand, you use a method to estimate the effort, then say so in the Quality Manual and prove you use it every time you issue a quotation to your customers

The prospect becomes a client when we start some work for them, so it doesn't add much more, except give us places to hold more documents, such as any Health and Safety policies or other general instructions, to initiate projects and a set of searches to aggregate information for the project level Wiki pages, which is really were the main activities take place.

Though I said earlier that we could leave out the ISO 9000 section on handling and storage of products if we were just in a consulting project, we do have to allow for the situation where we are lent, or placed in charge of, some property belonging to our client. This might be a piece of software installed on our equipment to access a database, or the loan of a laptop. In all cases we must demonstrate to the auditor we are taking care of our client's assets. If we go as far as to provide facilities to a client (for example web hosting), or if we are given the responsibility to run some facility on behalf of the client we must once more list what it we are doing and do and fill in a risk assessment form. All of these are

in pre-defined Wiki pages, so the burden of completing them is light. They are to record what it is we are up to for the auditor to have a clearer understanding and to ensure we keep an eye on what we agreed to do.

HMW uses Process and Data Definitions to gather user requirements into outline system specifications. They are based on a number of sources and are compatible with IEEE 830 (which I won't go into here), we've successfully used them for years, but this particular layout is not mandatory for ISO 9000. We turn the definitions into Development Schedules and Modules, which act both as a part of the code repository. And as a data item for the Incident and Change Request tracking parts of the QMS.

There is a page to record the project environment, which describes the hardware and operation system the project is to use. The daily notes log is a bit like a project Blog. There are a number of other supporting pages, but I won't go into them here. The fact that all this information is all kept in a Wiki enhances the project documentation over anything kept on paper. For example, by automatically cross indexing Process and Data Definitions we can see which processes create and access particular data items.

This supports the whole software development lifecycle. It's all ISO 12207 compatible, which is yet another standard (but don't get me started on this one), so we are killing two birds with one stone. Of course, other documents could still conform to ISO 12207 if they satisfy the QMS requirements.

ISO 9000 says a lot about checking output for defects, which I think is called testing in software developments. We use our development schedule pages to assign testing activities, but as long as you can prove you've tested the software to an auditor, then you can use any layout you wish.

HMW uses Maya and Inca our in-house code and change management tools. They are separate from the QMS, of course, but we use the Wiki to publish change files to clients and as the repository for completed products, which covers the preservation, handling and dispatch requirements of ISO 9000. Other teams could use workspaces or some other mechanism. TWiki handles all file types as attachments, with the added advantage of built in version control on each attachment. I stress that we use the Wiki as a repository of official releases, not as the development platform.

The Wiki has a sub-system to allow recording of hours worked against any project defined in the QMS. The result is searchable time sheets, which are used by our in-house accounts to generate invoices based on the time recorded.

We monitor the QMS performance with a series of searches and the work flows provide sensible points for reviews, with appropriate links to on-line forms which activate when needed. The data model is fully documented, but I've not included it for the sake of brevity, as this article is long enough as it is.

Why would you want ISO 9000?

For some business sectors it is a pre-requisite: defence; engineering; and some NHS work. Many other organizations have ISO 9000 certification for at least their customer services and feel more comfortable with suppliers who can demonstrate the same standard of service. I would think that all customers care about the quality of work that is done for them and ISO 9000 is one way of advertising that you are the ones to provide it.

It shows you have a process in place and so increases customer confidence and satisfaction, improving the likelihood of closing deals or renewing contracts.

Using a standard document can increase your efficiency, obviously by making sure you don't forget something, but also by stopping you don't from putting down too much, wasting time and confusing the client. After all aren't things like "patterns" development templates? Aren't they just a reminder of what to include and exclude in any process or object?

Conclusion

Our experience shows that, admittedly once we put in a lot of effort to build the QMS, it doesn't have to be painful to use keep it going.

APL projects using the type of development approach we take for granted can achieve international standards and not be seen as some bastard child of the "mainstream" techniques.

ISO 9000 can help you to make your company more productive, efficient and therefore more profitable.

Further reading

1. PCG ISO 9000 official site http://iso9001.pcg.org.uk/cms/index.php
2. List of ISO 9000 standards
   http://www.iso.org/iso/iso_catalogue/catalogue_ics/
   catalogue_ics_browse.htm?ICS1=3&ICS2=120&ICS3=10

3. Twiki Open Source Enterprise Wiki http://twiki.org
4. HMW's Code of Ethics http://www.hmwcomputing.co.uk/CodeOfEthics.html
5. IEEE 830 Recommended Practice for Software Requirements http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html
6. ISO/IEC 12207 Framework for Software Lifecycle Processes http://www.iso.org/iso/catalogue_detail?csnumber=43447

Unicode Support for APL

by Morten Kromberg
mkrom@dyalog.com

An earlier version of this article was presented as a paper at APL07 and published in APL Quote Quad as part of the Proceedings. Ed.

Unicode offers users of APL the same benefits as users of other programming languages, namely the ability to consistently represent and manipulate text expressed in any of the world's writing systems. For APL users, there is a significant bonus, which is that Unicode includes the APL character set, and therefore allows APL programs to be represented and manipulated using industry standards which are finally becoming widely supported ('only' 17 years after the first version of the Unicode standard). The paper presents the design of Unicode support for an APL interpreter which fully supports Unicode in all phases of system development and deployment. The paper discusses the internal representations used for Unicode characters in Dyalog version 12.0, and how the design is intended to provide a smooth upgrade path of users of earlier versions of the system.

Introduction to Unicode

Unicode is an industry standard allowing computers to consistently represent and manipulate text expressed in any of the world's writing systems. It assigns a number, or *code point*, to each of approximately 100,000 characters, including the APL character set. The first version of the standard appeared in 1991, but it is only in the last few years that support for Unicode has become common in operating systems and 'mainstream' applications.

The adoption of Unicode provides APL users with three important benefits:

- It becomes possible to write applications that fully support not just North American and Western European character sets, but all of the world's languages and writing systems – including the APL character set itself.

- Character data no longer needs to be translated as it enters or leaves the APL system during interoperation with other components like database systems or code libraries written in other languages.
- APL source code can now be handled by the off-the-shelf source code and project management tools.

Adding support for Unicode requires changes to the way character data is entered, displayed and stored in an application. Much of the work involved consists of ripping out special APL mechanisms for handling keyboards and translating data, in favour of using standard tools provided by the operating system. Although work is required to change the way the interpreter and session manager works, the resulting system is easier both to maintain and to explain to future generations of APL users.

Character encodings

Although the Unicode standard assigns a unique number to each character, these *Code Point Numbers* can be stored using a number of different encodings. There is a set of fixed-width encodings named UCS-*n*, where UCS stands for Universal Character Set and *n* is the number of bytes used to represent each character, and another set of variable-length encodings named UTF-*n*, where UTF stands for Unicode Transformation Format and *n* is the smallest number of bits (not bytes) used per character.

The four most commonly occurring representations are probably UCS-4, UCS-2 (which has become obsolete since Unicode broke the 2-byte boundary), UTF-8 and UTF-16:

UCS-4

(also known as **UTF-32**) is a fixed-width encoding which uses 4 bytes per Unicode character, and is able to represent all characters that will ever be defined by the standard. The Unicode standard guarantees that the highest code point which will ever be allocated is x10FFFF, a number which requires 21 bits to represent, so 4 bytes (32 bits) is ample to represent all of Unicode. This format is common in some Unix environments.

UCS-2

is an obsolete representation which was popular in the early days, when the standard only defined characters in what is now known as the Basic Multilingual Plane. Until Windows 2000, it was the default encoding on the

Windows platform. It can only represent Unicode code points up to xFFFF (65,535 - which does still cover most of the commonly-used characters).

UTF-8

is the most commonly used encoding for data in files (especially web pages), under both Unix and Windows. The reason for the popularity of UTF-8 is that it is backwards compatible with 7-bit ASCII (using one byte to store each character in the range 0-127), and also that – unlike the other encodings - it is not affected by the endianness of the platform (see the following discussion of the Byte Order Mark). Two bytes are required to represent code points from 128 to 2,047 (x7FF), three bytes per character from 2,048 to 65,535 (xFFFF), and 4 bytes per UTF-8 encoded character outside the Basic Multilingual Plane.

UTF-16

is an encoding which is identical to UCS-2 for all characters in the range 0-xFFFF, except for 2,048 so-called surrogate code points (D800-DFFF). Some bit patterns in that range are used in UTF-16 to indicate that second 16-bit word is required. UTF-16 is the default encoding on the Windows platform, from Windows 2000 onwards.

The Byte Order Mark

The final thing that should be mentioned in a five-minute introduction to Unicode is the 'BOM'. Because microprocessors differ in whether the most or least significant byte is written to storage first, it is necessary to know which type of system wrote a text file in order to properly decode it. Unicode defines two code points, xFEFF and xFFFE, as the big- and little-endian Byte Order Marks, respectively. Some (but not all) applications will write byte order marks, usually at the beginning of a text file, to make it possible to detect the encoding used:

File starts with…	**Encoding is**
EF BB BF	UTF-8 (these 3 bytes are the UTF-8 encoding of FEFF)
FF FE	UTF-16 (or UCS-2), little endian
FE FF	UTF-16 (or UCS-2), big endian
FF FE 00 00	UTF-32 / UCS-4, little endian

When writing text files, especially in a Windows environment, it *can* be a good idea to prefix the file with the above byte sequences (but some applications still ignore or get confused by BOM sequences).

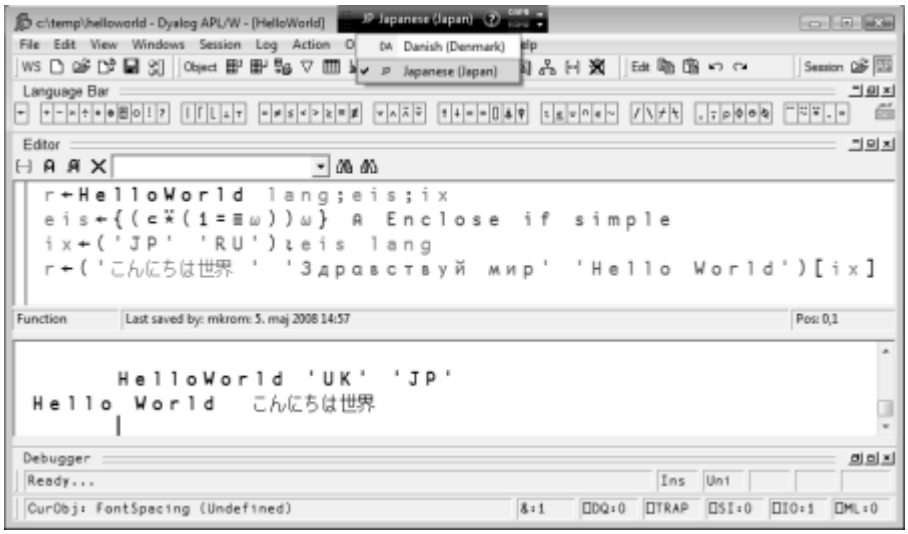
Wikipedia is a good source of information about encodings: take a look at Comparison of Unicode encodings.

Using a Unicode APL system

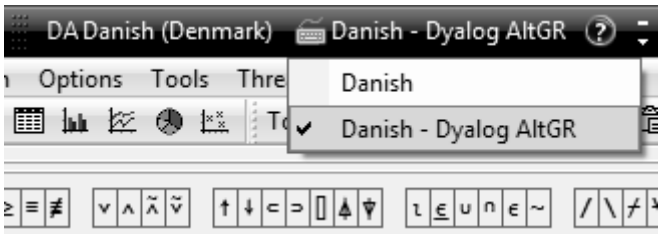
The APL system with Unicode support is designed to look, feel – and work – just like the non-Unicode systems that preceded it. With a very small number of exceptions, APL code that does not either exchange data with external components, or use the system function ⎕DR to explicitly work with the internal representation of character data, should run unchanged.

The obvious difference (which is hard to detect if your main language is English), is that you can enter and display all the characters on your keyboard – and use 'Input Mode Editors' for eastern languages – not only to enter data into applications, but also into the APL development environment itself.

The following example shows the definition and execution of a simple multi-lingual application:



The black rectangle at the top of the screen, which has dropped down a menu, is not part of the APL session – it is the *Windows Language Bar*. The language bar allows multi-lingual users to switch keyboard layouts – in the above example Danish and Japanese are available. For entry of APL characters, an alternative layout for the main language is typically used:



Danish – Dyalog AltGR (selected above) is an APL keyboard based on the standard Danish keyboard, with the addition of AltGR (which can also be keyed as Ctrl+Alt) key combinations to enter APL symbols. For example, ⎕ is entered using AltGr-L. 'Classic' Dyalog Keyboards based on Ctrl key combinations are also provided, but these are harder to use in most Windows applications, which tend to define a number of Ctrl combinations as 'hotkeys'. The bad news is that more and more applications are starting to use AltGR key combinations as keyboard accelerators, which means that some APL characters remain hard to enter in those applications (some applications allow you to disable these hotkeys).

The APL keyboards are not part of the standard operating system language support, but they have been created using standard tools (in the case of Microsoft Windows, we use a tool called the Microsoft Keyboard Layout Creator). One immediate advantage of using standard tools is that we have been able to involve more people in keyboard creation, so version 12.0 is shipped with significantly more national APL keyboards than earlier versions.

Hotkeys (typically Ctrl+Left Shift) can be set up for quick switching between input modes without using the mouse to click on the Language Bar.

Most of the behavioural differences apparent in the Unicode edition consist of strange behaviour having disappeared, for example:

- You can copy and paste between the APL development environment and other applications, and all that is required for APL symbols to appear correctly in a word processor is that a Unicode font containing APL glyphs is selected. Both Windows and Unix systems come with at least one standard Unicode font which can be used to display APL.

- The same is generally true when you write data to files or database systems: No translation of the data is required (although you *might* be required to encode the data, depending on the interface used).

The code changes that are required are natural consequences of the new internal representation of characters in the Unicode edition of Dyalog APL.

Internal representation

Historically, APL systems have used a single byte (8 bits) to represent each character in an array. The list of all the 256 possible characters [1] is known to APL users as the Atomic Vector, a system variable named ⎕AV. Atomic vectors differ from one vendor to the next, sometimes even between products from the same vendor, and some vendors (including Dyalog) allow the user to redefine sections of this system constant, in order to support different national languages.

The highest currently assigned Unicode code point is currently around 175,000, and the standard specifies a maximum value of x10FFFF (a number slightly larger than one million – an order of magnitude higher than the total number of characters currently defined). Three bytes (24 bits) would be sufficient to represent any Unicode character. In practice, it is more common to use four bytes (UCS-4/UTF-32) for a 'complete' fixed-width representation, simply because three bytes is an impractical group size for most computer hardware.

Obviously, a single byte per character is no longer sufficient to represent character arrays in a Unicode APL system.

Unicode representations in other array languages

At least two array language vendors (IBM and Jsoftware) had implemented Unicode support before Dyalog, so we had the benefit of examining the solutions that our colleagues had chosen before deciding what to do. I am grateful to David Liebtag at IBM and Chris Burke of Jsoftware for helping me to understand the details of – and the reasoning behind – the choices that were made:

IBM added multi-byte character support to APL2 *before* the first version of the Unicode standard saw the light of day. From Version 1 Release 3 (dated 1987), APL2 uses 1 and 4 byte representations for character data. The 1-byte representation is used for arrays containing characters that are all elements of the APL2 atomic vector. The 4-byte representation is used for other character arrays. In the 4 byte representation, 2 bytes contain the data's codepage and 2 bytes contain the data's code point. Support for Unicode was added in Version 2 Release 2 (dated 1994), where APL2 supports codepage values of zero and 1200 which both indicate the data uses the UCS-2 representation. These APL2 representations have the advantages that existing applications can run without change and applications which only use characters in the APL2 atomic vector have minimal storage requirements.

J introduced a double-byte character type containing UCS-2 encoded characters in Version 4.06, which was released in 2001. From Version 6.01 (September 2006), J switched from using ANSI to UTF-8 as the default encoding of (single-byte) character constants entering the J system from the keyboard and through the execution of J script files. This change was made in order to make it easy to work with text files, and in particular to support J scripts containing Unicode characters, as UTF-8 had emerged as the de facto standard for such files. If J applications wish to view Unicode strings as arrays where each element of the array is exactly one Unicode character, conversion to the double-byte (UCS-2) representation is required.

Initially, we considered following the APL2 model and having two encodings – one based on our atomic vector and a 'wide character' type containing Unicode characters not in ⎕AV. Upwards compatibility is important to our users, and this solution would allow existing applications to run completely unchanged, without changing the storage requirements. Eventually, we decided against doing this for two reasons:

- Simplicity: the number of people who will learn to use APL in the future vastly exceeds the number of people using it today. Maintaining two representations, and trying to continue to explain to future of generations of APL programmers how the single-byte representation came about, was not felt to be in the long-term interests of our users.
- The above was compounded by the fact that users of Dyalog APL have been able to customize their atomic vectors for different languages (in Eastern Europe in particular), so not all atomic vectors are the same.

Exposing the variable-length encoding to the end user in the way that J has done did not feel like an attractive solution. This gives a system in which character arrays containing about 50 European characters in the ANSI set (but outside ASCII) cannot easily be compared – or easily extracted from an array – without first being converted to a double-byte representation, where they consume twice as much space. For example, the following expression gives a length error in J:

```
'ä'='Seppälä'
```

This is because the array on the right has 9 elements, and the one on the left has 2. It is very easy to work around this in J, all you have to do is convert the strings to double-byte characters first using the public utility function `ucp` – but this just didn't 'feel right' to us.

Finally, Unicode had broken the 2-byte barrier in the years following the APL2 and J implementations, and we did not feel that we should restrict our users to the Basic Multilingual Plane.

One code point, one character

After much debate, we decided that the new Unicode product should only have a single type of character as seen from the user's point of view. Therefore, all character arrays contain one Unicode code point number for each character in the array. The system picks the smallest possible internal representation (1, 2 or 4-bytes), depending on the range of elements in the array – in the same way that Dyalog APL has always stored integers. This means that most existing customer data will continue to be represented using one byte per character. Arrays which contain APL symbols require two bytes per element, and a very small number of characters require four bytes (but no pre-existing data created by an earlier version of the system will require more than two). The character array `'Seppälä'` requires 7 bytes to store the contents of the array, and the system behaves as any Finnish or Japanese [2] person would expect:

```
      'ä' = 'Seppälä says: "こんにちは世界"'
0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

In the above example, the array on the right is stored using two bytes per character, because of the Japanese characters, which have code point numbers above 255:

```
      ⎕UCS'こんにちは世界'
12371 12435 12395 12385 12399 19990 30028
```

`⎕UCS` is a new system function for converting to and from integer code point numbers to the corresponding characters. The monadic function is self-inverse, and has the same definition as in APL2 implementation: a right argument of code points gives you a character array of the same shape:

```
      ⎕UCS 123 9077 91 9035 9077 93 125
{ω[⍋ω]}
```

In Dyalog APL, `⎕UCS` also accepts a left argument which must be the name of a UTF encoding. In this case, if the right argument is a character array, the numeric result contains the encoded data stream, and vice versa:

```
      'UTF-8' ⎕UCS '界'
231 149 140
      'UTF-8' ⎕UCS 65 195 132
AÄ
```

The dyadic extension makes it straightforward to work with variable-length encodings. Writing any character string to a UTF-8 encoded text file requires a slightly more complex expression, along the lines of:

```
      (⎕UCS 'UTF-8' ⎕UCS text) ⎕NAPPEND tn
```

The leftmost ⎕UCS converts the integers returned by dyadic ⎕UCS back into characters before writing them to file. This is necessary because numbers in the range 128-255 would cause data to be written using two bytes per element. In effect, the last ⎕UCS is used to turn the numbers into 8-bit unsigned integers. We debated adding an 'unsigned' type number to native file functions, or even a type number to select UTF-8 encoding as part of the native file interface, but decided to be conservative in the first Unicode release, as the above expression is quite straightforward and easily embedded in utilities for manipulating files.

If your application inhabits the Microsoft.Net framework, you can leave the encoding to the framework, and simply write:

```
      System.IO.File.WriteAllText filename text
```

The default encoding used is UTF-8, but this can also be selected using an optional third argument, for example:

```
      System.Text.Encoding.UTF16
```

The Classic Edition

Most applications written in Dyalog should load and run in the Unicode version without any changes – if they only use APL's own storage mechanisms (workspaces and component files) [3]. Applications which use Microsoft.Net, OLE/COM and the ODBC interface, will generally also only require minor changes, if any. Data moving through these interfaces was already being translated to UTF-16 for the first two and ANSI for ODBC.

However, many of our users have applications which share data with the outside world using other mechanisms. These applications will almost certainly contain some code which is dependent on the internal representation of character arrays. Even if an application is not actually going to use any new characters, modified code will need to be verified and tested in the new environment.

Even in the case where an application loads and runs without changes, components which the APL code connects to may not be able to deal with data outside of the ASCII or ANSI range. The application may need new validation code to be inserted in order to avoid breaking partner code. *If* it is the intention to extend the domain of data that can be handled to include new Unicode characters, the format of any external storage and encodings used in all connections with the outside world also need to be reviewed and possibly changed.

In recognition of the fact that many applications are going to have to go through a conversion cycle which *might* take some time to plan and execute, a 'Classic Edition' will be available for several future releases of the product. Dyalog Classic Version 12.0 will continue to use the single-byte atomic-vector-based representation of characters and all of the old translation mechanisms for data entering or leaving the system, and is intended to be 100% upwards compatible with earlier versions.

Although it is 100% compatible with old versions, the Classic edition is able to read the new data formats that the Unicode system uses in component files and TCP sockets. The intention is that users will safely be able to experiment with building Unicode-capable versions of their applications without having to have two sets of source code. Classic and Unicode variants of the Dyalog product itself are built from the same source code, differing only in whether character data is translated and limited to one byte per character – so the burden of supporting them both is mostly a QA and packaging issue.

We could have decided to carry the 'old style' character type forward into a single new system (as APL2 and J have done), but although this would possibly have made the transition smoother in the short term, it would quickly become a burden both for us and for our users. We felt that having two character types which required translation when moving between the two would be an endless source of confusion, particularly for new users.

Instead of complicating the product indefinitely, we decided to have two separate editions of product for a period of time, and design these in such a way that they inter-operated easily, in order to provide an environment which encouraged people to move to Unicode. This required a significant amount of additional work for us, and perhaps a little extra work for our users in the short term, but should result in a simpler system going forward.

Inter-operability between Classic and Unicode

A large part of the work involved in producing the Unicode implementation has been aimed at minimizing the effort required to move an existing application to the Unicode system – and in particular doing everything reasonable to avoid 'big bang' data conversion events, which can be daunting in large systems with many components – and tend to discourage conversions.

The Unicode edition is able to load workspaces and share component files with Classic editions (versions of Dyalog APL before 12.0 are also considered to be Classic). Component files created by Classic editions are considered to be 'non-Unicode', and new files created by Unicode editions can optionally be created with the Unicode flag switched off. TCP Socket objects have an equivalent flag. Any character data written to non-Unicode files or sockets is translated to the old format as it is written. This makes it possible to move part of an application to Unicode, allowing unconverted parts of the application to continue to work in Classic mode, in a controlled fashion.

In order to share data with Classic systems, a Unicode interpreter needs to know what the atomic vector 'used to look like'. A new system variable ⎕AVU (Atomic Vector-to-Unicode) defines the Unicode code points of the Atomic Vector [4]. In the Classic edition of version 12.0, which still uses single-byte characters based on an atomic vector, ⎕AVU defines how to map characters to Unicode (for example, when writing data to a component file with the Unicode flag enabled). The Unicode edition uses the variable to map data received from old APL systems, and to translate back to the old format when writing to non-Unicode component files and TCP sockets. Users who have defined translate tables and fonts to provide regional atomic vectors can set ⎕AVU to ensure that old data is correctly converted to Unicode. ⎕AVU can be localized and has namespace scope, making it straightforward to integrate data from different regions, or old applications using different conventions. ⎕AVU is intended as a migration tool. It should be used as a temporary measure to access data which absolutely has to remain in the old format (which might take a decade or two, but hopefully not longer than this).

Interoperability between the Unicode edition and versions before 12.0 is limited to the ability of the Unicode system to write to component files in the old format, and the ability of the Unicode system to)LOAD old workspaces. However, from Version 12.0, both editions can read each other's data formats (workspaces, sockets and files) – so long as a Classic edition is not required to receive or read characters which cannot be represented in its atomic vector. In this situation, a TRANSLATION ERROR is signalled.

Name Association

The Microsoft Win32 API is still used by many applications to access services provided by the Windows platform. As part of the migration to Unicode that is still ongoing in the platform itself, Windows generally provides two versions of every Win32 API function: a version which expects string arguments to contain single-byte ANSI data (with a name ending with the letter `A`), and a version which expects Unicode data encoded as UTF-16 (with a name ending with `W` for Wide).

The system function `⎕NA` has been extended to make it simple to write code which can run on both versions of the API. If an API function is named with a trailing `*`, `⎕NA` will link to the A function from the Classic edition, and the W function from the Unicode edition.

Likewise, the argument type `T` without a width specification is interpreted to mean a wide character according to the convention of the host operating system. This translates to `T1` in the Classic edition, `T2` under Windows Unicode, and `T4` under Unix or Linux.

For example, the following function will display a Message Box with OK and Cancel buttons in both editions (under Windows):

```
    ∇ok←title MsgBoxmsg;MessageBox
[1]  ⎕NA'I user32|MessageBox* I <0T <0T I'
[2]  ok←1=MessageBox 0 msg title 1 ⍝ 1=OK, 2=Cancel.
    ∇
```

Underscores

Somewhat surprisingly, Unicode seems to spell the death of the use of the underscored alphabet in APL identifiers. The APL Standards committee deserves huge credit (Praise them with Great Praise, except for the unfortunate episode regarding the naming of tacks [5]) for getting all other APL symbols that have ever been used or proposed in an APL system into the standard. However, the underscoring of letters of the English alphabet is seen as a form of emphasis, and underscored characters have not been assigned code points.

In most Dyalog installations, it is a long time since the 26 characters that used to be the underscored alphabet have been ‘recycled’ to provide a selection of western European accented letters:

```
      ⎕AV[97+⍳26]
ÁÂÃÇÈÊËÌÍÎÏÐÒÓÔÕÙÚÛÝþãìðòõ
```

However, some users of Dyalog APL still have significant quantities of code using underscored identifiers. To allow these users (even more) time to migrate, underscores have been mapped to a circled alphabet – which somehow *did* manage to get included. Adrian Smith's font APL385 Unicode, which has become the standard font for most Unicode APL systems, displays these characters ('incorrectly') as underscores:

```
      ⎕AVU[97+⍳26]←⎕UCS 9397+⍳26
      ⎕AV[97+⍳26]
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

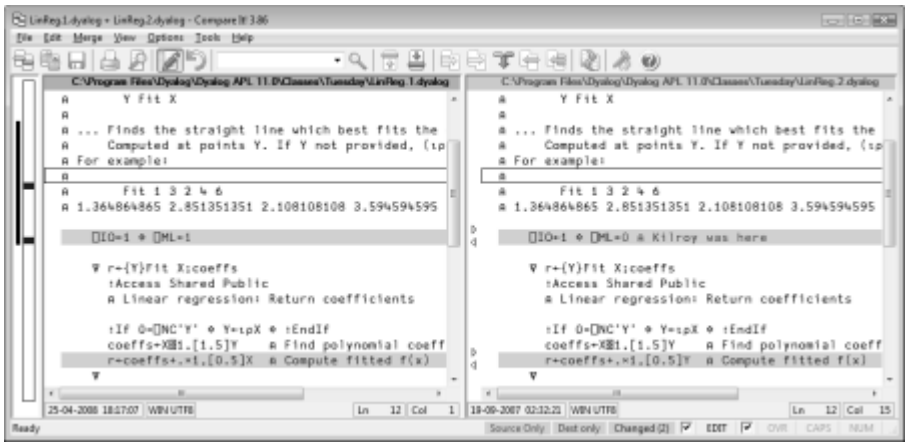
The first statement above specifies that the relevant part of the atomic vector should be mapped to the circled alphabet in Unicode, the second displays the 'third alphabet'. In any other Unicode font, these would display as the circled alphabet, Ⓐ to Ⓩ. We strongly recommend that anyone still using underscored letters makes plans to give them up. The underscored alphabet now constitutes the sole remaining incompatibility between the APL character set and Unicode.

Future challenges

We believe that Version 12.0 of Dyalog has achieved a number of important objectives:

- Dyalog applications can make full use of Unicode data in applications.
- A reasonably smooth migration path exists from the Classic to Unicode editions of the product.
- The use of Unicode script files for APL source code makes it easy to tap into a large collection of software development tools that we can now share with users of other programming languages.

There is not space here to discuss the latter point, but the following screenshot shows an inexpensive file comparison tool called *Compare It!*, which the author spent about an hour finding and downloading from the internet. It was able to compare and merge APL source files without any other work than selecting the APL font for display:



A number of challenges remain to be resolved in future releases:

Allowing new characters in names

Unicode supports just about all human alphabets, which raises questions about whether letters from other languages should be allowed in identifiers named by APL users. In the first Unicode version, we have not taken any steps in this direction, except to allow letters from the circled alphabet *and* the alternative European letters shown in the discussion of underscores (these were previously mutually exclusive, as pre-Unicode users had to pick one or the other of these sets).

The main reason that we have hesitated is that there is significant potential for confusion between similar characters. In particular, the APL language uses a number of Greek letters as primitives. There is a body of opinion which holds that the clearly visibly distinction between primitive symbols and user-defined names is an important aspect of the readability of APL language statements. Greek APL users are undoubtedly justified in wanting to write `αρίθ←{ρω}` in place of `count←{ρω}`, but (depending on the font), readability may suffer.

Other languages can cause confusion due to similarity with English letters. For example:

```
      ⎕UCS 65 913 1040
AAA
```

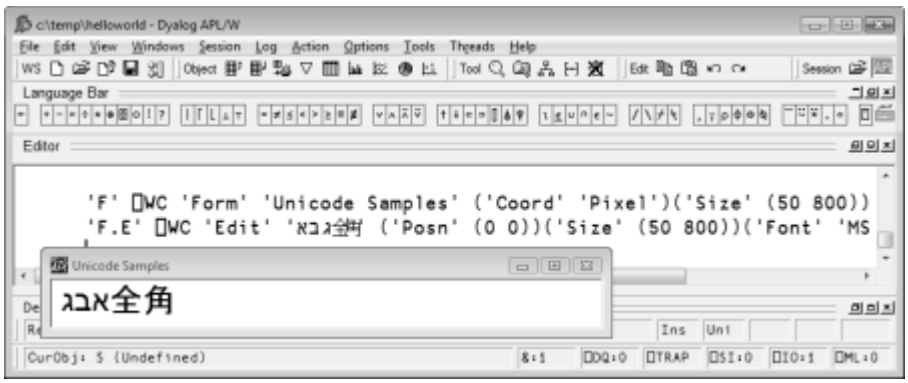
The above statement returns the first letter of the Latin, Greek and Cyrillic alphabets. I'm not sure I would like to debug code in which `A`, `A` and `A` were three different variables – or go looking for a font in which I could tell the difference between all the capital As in Unicode.

The issues above were enough to convince us to chicken out of extending the set of characters allowable in user-defined names, until we have had a bit more time to think about the consequences. In an age where writing components for use by other languages is getting more and more important, a conservative approach to naming seems like a good idea.

Improved display

While the use of fixed-pitch fonts for APL sessions and code editors has clearly been the best choice to date, Unicode poses new challenges. Some scripts use symbols which should occupy twice as much space as Latin letters. In addition, some languages are expected to be displayed from right to left. The following screen shot illustrates both issues. The code sample creates a form containing an edit field, and inserts into it:

- The three first letters of the Hebrew alphabet
- Two double width Japanese “Zenkaku” characters.



Windows controls have the knowledge required to correctly render this text: The three Hebrew letters are displayed from right to left, and the Japanese characters are given more space. The extent to which it would help developers if the session did the same, is not clear. It is probably a good thing if we could avoid having the Japanese characters overlapping in the session, as they do above. As an interim measure, Version 12.0 has a property on the session which can be used to increase the spacing between characters if you are using characters which are wider than normal.

On the other hand, it seems that one would want it to be easy to predict the result of a statement like:

```
      3⊃'אבג全角'
ג
```

In this case, it seems to make sense that the session manager of an array oriented language should stick to the view of each character as separate element of the array, and always display arrays in the same way – first element on the left.

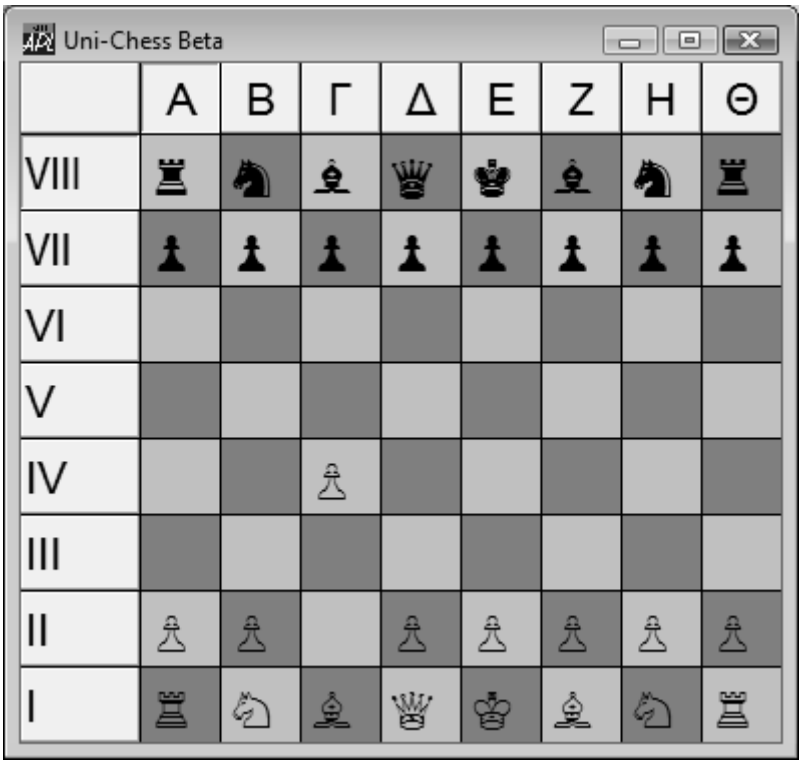
Fun and Games with Unicode

Unicode can be fun, too:

```
ChessPieces←⊃'♔♕♖♗♘♙' '♚♛♜♝♞♟'
Officers←3 5 4 2 1 4 5 3 ⋄ Pawns←6
White Black←1 2
ix←Black White∘.,Officers,8⍴Pawns
Pieces←4 8⍴ChessPieces[0 ¯8⌽ix]
Board←(2⌽4/1 0)⍉Pieces
'Chess' ⎕WC 'Form' 'Uni-Chess Beta'(40 20)(341 381)'Pixel'
'Chess' ⎕WS 'Font' 'Arial Unicode MS' 30
'Chess.Board' ⎕WC 'Grid' Board (0 0) Chess.Size
Chess.Board.(TitleWidth CellWidths←60 40)
Chess.Board.ColTitles←,¨'ABΓΔEZHΘ'
Chess.Board.BCol←(192 192 192)(127 127 127)
Chess.Board.CellTypes←(⍳8)⌽8 8⍴2 1
Chess.Board.RowTitles←,¨⎕UCS 8543+⌽⍳8
```

Pawn to Queens Bishop 4?

```
Chess.Board.(Values[5 7;3]←Values[7 5;3])
```



The hard part is now done; the rest is left as an exercise for the reader.

Acknowledgements

I would like to thank:

- David Liebtag and Chris Burke for helping me get the story a bit straighter on the details of the APL2 and J Unicode implementations.
- Anssi Seppälä for the use of his name in an example, and Alexey Miroshnikov for “Здравствуй мир”
- John Daintree, John Scholes, Geoff Streeter, Nicolas Delcros and Vincent Chan – and many others at Dyalog.

For more on Dyalog Version 12.0, see www.dyalog.com/help

Notes

1. Or in the case of APL systems on 9-bit DEC architectures, 512.
2. If the Japanese example doesn’t actually mean “Hello World”, it is the Internet’s fault!
3. The only language APL primitive affected is monadic *grade up*, which now sorts according to Unicode rather than Atomic Vector indices. For more information on coding

changes, see http://www.dyalog.com/help/html/relnotes/converting%20to%20unicode.htm

4. The system variable ⎕AV still exists in the Unicode edition, in order to allow old code to continue to run. It is still a 256-element character vector, which is now 'defined' by ⎕AVU, in that ⎕AV≡⎕UCS ⎕AVU.

5. It is unfortunate that Unicode names ⍫ "APL functional symbol up tack jot" and ⊤ "Down Tack".

L E A R N

Rain flips its q

A database engine for simple folk

by Adrian Smith
adrian@apl385.com

These notes are lightly adapted from the script I wrote for the Sunday night light-entertainment at Dyalog 08. Yes, the apparent integration of q-sql with the APL session was a gross deception using ⎕TRAP but the basic idea of running a low-tech relational database inside your workspace is a perfectly good one. I pretty well depended on it for the 10 years I led a team who wrote applications in APL, and who knew no other way. Morten also made very good use of the query tools in his examples of monitoring APL primitives, which was nice to see. Sometimes the best ideas are the ones you come up with under so much time pressure that you have no time to look ahead. We shall see. Anyway, here is the script that I spoke to, and which was included on all the memory sticks, along with the workspace.

Motivation

There was a time, long ago and far away, when I used to write proper business applications in APL. I am thinking back to 1983 and my Loughborough paper on databases, which reminded me of just how heavily I depended on a simple, reliable DBM working entirely within the confines of the APL workspace. Subsequently, I had evolved into something of a tool-builder rather than a tool user, and this state persisted until a few weeks ago.

When I had a phone call from a old Rowntree friend (he used to run Assortments in York) for whom I wrote a lot of planning and scheduling applications. He is kinda retired now, and has ended up managing the rebuild of the east end of York Minster 'in his spare time'. This is a big project (around £21M of lottery funding over 5 years) and has put huge pressure on the Minster Stoneyard to turn out carved blocks to a predictable schedule. Herding cats comes to mind.

Carver at work

So Peter called me, and wondered if I would be interested in reviving some 20-year old skills and cranking out a little scheduler for them, so at least they might know a little way ahead who will be carving which blocks when, and consequently when various bits of the overall project might get completed. Was I interested – of course I was!

Panic then set in fairly quickly, as I realised that an essential chunk of the software I needed was last seen on a VS APL system that was turned off in late 1999. So I looked around my collection of heaps of things and found a flipdb user guide and (of course) a copy of *q for Mortals*, which I recently reviewed. Towards the end of this review, I caught myself musing around the possibility of implementing something comparable in APL. After all, kdb was originally just a bunch of q and Paul Mansour had gone down a very parallel road with flipdb. Time to give it a go, thought Adrian.

A database architecture for APL

Firstly, this has to be designed to make the raw data available to your application in the simplest way imaginable. You must *not have to* run a query (or call a server) just to get some numbers back. In the old days I just used variables in the workspace, with a 'master variable' called (say) `⍙emp` which was just a namelist of other variables like `⍙name`, `⍙sal` and so on. A few simple utilities implemented operations like take and compress on the table as a whole. The description

variable developed a few whistles and bells, the main one being support for foreign keys (when the emp table has a pointer to a dept table) which were implemented as simple (zero-tolerant) 1-origin indexer variables. So when you deleted a department, it knew to fix up the indices in any other tables which referenced it (or yell at you if the deletion was not acceptable).

Let's slide forward from the VS APL timeframe where this was all written, to the 21st century and Dyalog APL. Namespaces could be very handy here, and maybe those foreign keys could just be refs? If we start with the basic notion that a table maps to a namespace with some variables in it, then we pass the first hurdle of data accessibility. What could be easier than:

```
      emp.sal
12340 2345 2345 1200 1234 0 1234
      +/emp.sal
20698
```

Rather than using prefixed names in the old-fashioned way, we just prefix with the namespace. This makes it pretty trivial to code up (for example)

```
      8 db.take emp
      emp.sal
12340 2345 2345 1200 1234 0 1234 0

     ∇ {tbl}←len take tbl;nl2
[1]   ⍝ Normal APL take/overtake for all vars in namespace <tbl>
[2]    nl2←tbl.⎕NL ¯2
[3]    :With tbl
[4]      ⍎¯2↓∊(⊂'↑⍨←len ⋄ '),⍨¨nl2
[5]    :End
     ∇
```

Of course you need to be sure that the only variables you leave lying around are valid to be taken/compressed by the same expression. Anything else we want to store about the data is going to be tucked away in a sub-namespace. Alert readers should have spotted the flaw in the above function. I should really have formatted the length and used dyadic execute rather than `:with` here. One day I'll have a variable called `len` in a table, and it will all go horribly wrong.

Anyway, moving quickly on, if we could throw some descriptive stuff into a sub-namespace, that would help with the formatting of the output, and also give me somewhere to record those pesky relationships.

```
      db.Columns emp
 Varname         Key  Type  Format      References  Shape
 -------         ---  ----  ------      ----------  -----
 #.emp.id     A  *    num                               8
 #.emp.name   A       text                              8
 #.emp.dept   A       ptr               #.dept          8
 #.emp.sal    A       num   £##,##0.00                  8
 #.emp.bonus  A       num                               8
 #.emp.band   A       num   00                          8
 #.emp.sex    A       ptr               #.sex           8
```

The references really are refs here, everything else in the description namespace is just text, so is pretty trivial to edit. There are a few handy utilities called `db.CreateTable` and `db.CreateColumn` which help you to get things in the right order. So far so good. If I reorder the dept table it can check for any refs to itself and go fix up the corresponding pointers (fortunately most primitives like iota and membership work fine on refs now) so we are still quite low on rocket science. Just formatting the content and throwing it into the session is very easy now:

```
      db.Show emp
   id    name         dept*         sal  bonus  band  sex*
 ----    ----         ----   ----------  -----  ----  ---
 5728 |  William       451   £12,340.00    100    01  Male
 1234 |  Gill          777    £2,345.00    100    03  Female
 1238 |  Richard       451    £2,345.00    400    02  Male
 1240 |  Tim           822    £1,200.00    100    02  Male
 1241 |  Beryl         451    £1,234.00      0    01  Female
 1242 |  Hello there   822        £0.00      0    01  Male
 1243 |  Farewell      451    £1,234.00    321    02  Never
    0 |                  0        £0.00      0    00
```

Better get rid of that dodgy extra row (the overtake a few lines back) but at least you can see what it did.

How hard can it be

to write a `db.Select` utility that takes a table and some expressions, and throws you back what is effectively another table?

```
      db.Select emp ('' 'sal>1000')
#.db.sel
```

Well, it returned a ref all right, can we see the content with the same kit we used to see the original table?

```
    db.Show qq←db.Select emp ('' 'sal>1000')
  id    name      dept*         sal  bonus  band  sex*
----    ----      ----   ----------  -----  ----  ---
5728 │  William    451   £12,340.00    100    01  Male
1234 │  Gill       777    £2,345.00    100    03  Female
1238 │  Richard    451    £2,345.00    400    02  Male
1240 │  Tim        822    £1,200.00    100    02  Male
1241 │  Beryl      451    £1,234.00      0    01  Female
1243 │  Farewell   451    £1,234.00    321    02  Never
```

More to the point, can we run a further query on the result of this one?

```
    qqq←db.Select qq ('name,sal,bonus,both←sal+bonus' 'bonus>100')
    db.Show qqq
name            sal  bonus  both
----      ---------  -----  ----
Richard   £2,345.00    400  2745
Farewell  £1,234.00    321  1555
```

This is the property of *closure* that the database gurus insist is so important. It allows you to write a proper relational algebra, with functions like set-union and set-difference that act on tables and return tables. It is one of the great strengths of q and is something we should constantly keep in mind. Of course we can still get at the raw data very easily:

```
     qqq.(name both)
 Richard  Farewell   2745 1555
```

The only magic here is the magic that comes with your interpreter. Honest!

More interesting queries

Hands up everyone who hates writing joins in SQL. Both Arthur and Paul (independently, I think) had the brilliant idea of using the dot-notation to allow a query to 'look down the link' in a very intuitive way. An example will make this clear:

```
    sel←db.Show∘db.Select
    sel emp ('name,dname←dept.name,mgr←dept.mgr.name' 'bonus<100')
name         dname        mgr
----         -----        ---
Beryl        Main office
Hello there  Dogsbodies   Gill
```

The dot is appropriate for many-to-one relationships, and of course you can chain them to follow the links to the end of the earth (and back again, as in the example above). Just try asking Oracle to list all employees who earn more than their manager, and you will rapidly appreciate the power of this simple extension. Incidentally variables get 'sensible' new names if you leave them to default, but giving them names explicitly generally makes sense.

Paul takes this a step further by allowing us to look through the wrong end of the telescope and see the many from the one, like this:

```
     sel dept 'id,name,emp[dept].bonus'
 id    name                          bonus
---    ----                          -----
451 │  Main office                   [100 400 0 321]
977 │  Real workers are here again   []
822 │  Dogsbodies                    [100 0]
777 │  The End                       [100]
```

Which illustrates another great thing about q (and flipdb, of course) which is that you are allowed to group things without applying a reduction. Of course you can reduce the groups if you want to:

```
     sel dept 'id,name,max emp[dept].sal'
 id    name                                          sal
---    ----                          -------------------
451 │  Main office                   12340
977 │  Real workers are here again     ¯1.797693135E308
822 │  Dogsbodies                     1200
777 │  The End                        2345
```

… but be aware that this is APL and that reductions on empty arrays don't always do what you expect! I need to have a proper think about missing-value support before I fix this one.

Grouping is something we need to do all the time, so it makes sense to accommodate it in the normal query syntax. Along comes another optional

argument (sorry Arthur, I put these in a different order) to the selection to give us the new keys for the output table:

```
      sel emp ('count,sum sal,avg sal+bonus' '' 'dept.name')
name           count  sum sal  avg col4
----           -----  -------  --------
Main office |      4    17153    4493.5
The End     |      1     2345    2445
Dogsbodies  |      2     1200     650
```

Note that the grouper(s) are marked as keys (as they are always unique combinations) and the output is once again a table that we can do further processing with, or just treat as a handy repository for some working data. If we group on two columns:

```
      sel emp ('count,sum sal,avg sal+bonus' '' 'dept.name,band')
name         band    count  sum sal  avg col5
----         ----    -----  -------  --------
Main office    01 |      2    13574      6837
The End        03 |      1     2345      2445
Main office    02 |      2     3579      2150
Dogsbodies     02 |      1     1200      1300
Dogsbodies     01 |      1        0         0
```

Then we get two keys (duh!) and something stirs deep in Adrian's subconscious. Maybe we could use these as the row/column indices into a crosstab, like this:

```
  xtab←db.XTab∘db.Select
  xtab emp ('sum sal' '' 'dept.name,band')
name                01       03       02
----           -------  -------  -------
Main office |    13574               3579
The End     |               2345
Dogsbodies  |        0               1200
```

As of today, this is just a fancy way of displaying a table with two keys, but a sense that it should become another first-class object, related to a table, but not quite the same. Perhaps *brick* or *box* would be good names for it. *Cube* has already been done to death, I think! The bit in the middle looks awfully like a matrix.

A quick look beneath the surface

There is really only one important idea behind this. Having checked out the tokens in each expression, the `Select` function runs the expressions in the table (so they can see all the variables) but with the path set to look for the utility namespace which lives inside `#.db`. This allows you to write any APL you like in the query, or make use of some simple canned functions if you can't be bothered to re-invent `average` every time you want it. Here are the 6 wettest days we have had, in date order:

```
      sel rain ('' '{ω>¯6+⌈/ω}⍋⍋rain')
date        rain  mintemp  maxtemp
----        ----  -------  -------
25/10/92 │  38.9       ¯2        6
08/06/99 │  38.1        6       11
02/08/02 │  49.3       15       20
10/08/04 │  55.4       18       21
15/06/07 │  53.4       10       12
25/06/07 │  38.1       12       18
```

… and here is what our `avg` function does:

```
avg←{1{(+/ω)÷↑⍴,ω}godeep ω}
godeep←{
  (1+⍺)>|≡ω:⍺⍺ ω
  ⍺ ∇¨ω
}
```

… with much the same stuff for all the standard summary functions. I am sure 10 mins of Alan Sykes' time will add lots more, like variance, median and all those other ones that standard databases never have when you want them.

With one mighty bound

We have a functional query engine, we have an APL session. Perhaps we should put them together to make one of those little languages that we all remember from the 1970s. Remember where we came in? Well, *Stones* have *Types* and *Types* have *estimated days*, so wouldn't it be great if we could type:

```
xtab count,ManDays←sum Type.Mason+Type.Carve
from ym.Stone
by Type.Id,Level.Name
where Level.Id in 'B,C'
```

```
  asc Type.Id

Id                          Level B          Level C
--                       count  ManDays    count  ManDays
Arcading               |     1       25
Arcading/string        |     3       45
Ashlar                 |     4        4        3        3
Grotesque              |     2       80        2       80
Niche head             |                       4      180
Niche head course 2    |     2       48        5      120
Shaft moulding         |    31      248       44      352
Shaft stooling         |     2       20
String/pedestal        |                       3       60
```

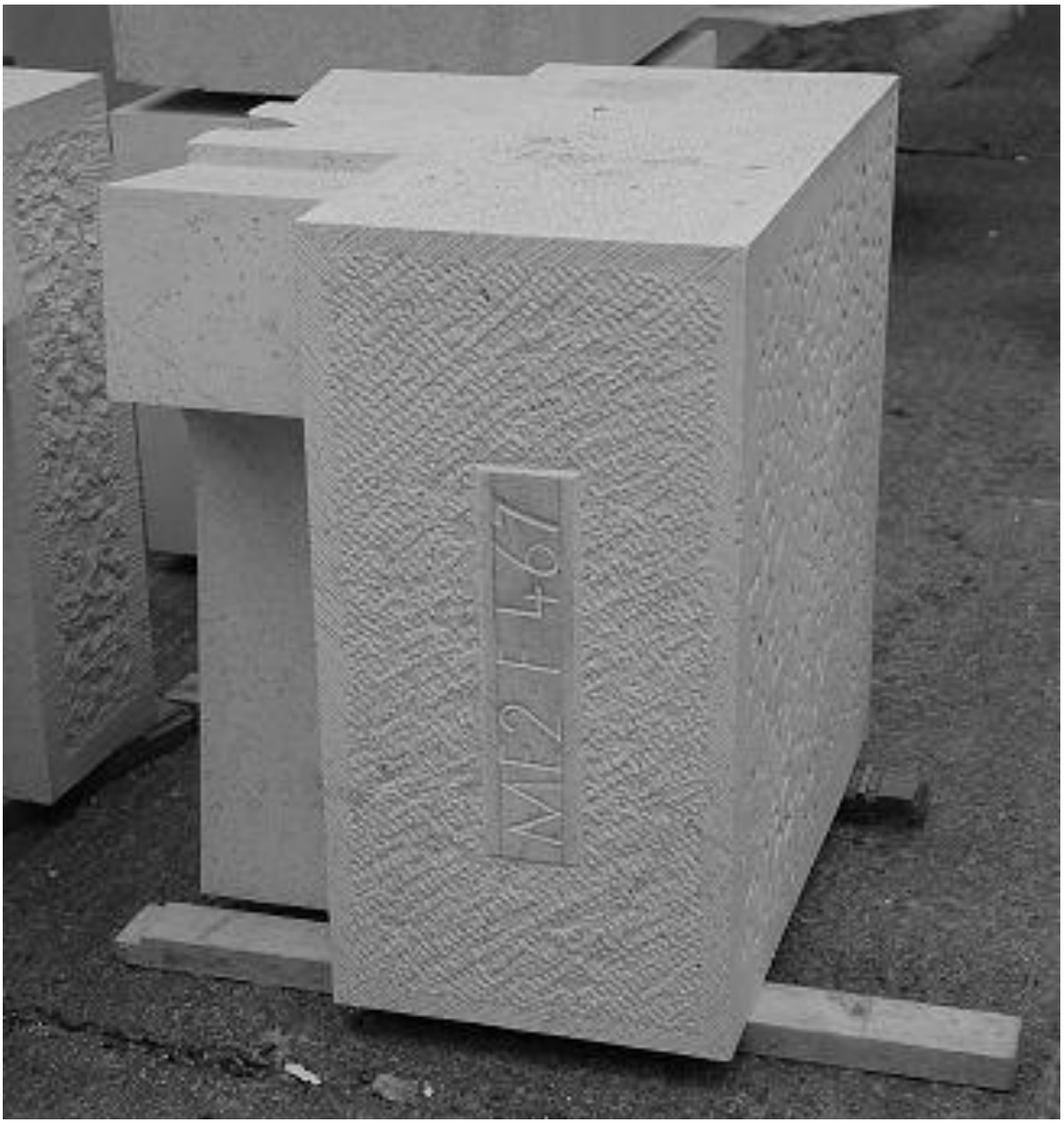
... to get an overview of how much work is left. Perhaps some of you are old enough to remember those special daisy-wheels that allowed you to plot to some small fraction of a character? Well, with Unicode we have some 1/8th blocks, so it's back to the future again:

```
select ManDays←sum Type.Mason+Type.Carve '⎕24 ###9'
from ym.Stone by Type.Id asc Type.Id where Type>0

Id                                          ManDays
--                        ----------------------------
Arcading                |  ▌ 25
Arcading/string         |  ▌ 45
Ashlar                  |  ▌ 12
Ashlar return           |  ▌ 12
Carved pedestal         |  ▌  60
Grotesque               |  ▌▌▌  160
Niche head              |  ▌▌▌▌▌▌  315
Niche head course 2     |  ▌▌▌▌▌  216
Shaft moulding          |  ▌▌▌▌▌▌
```

"Suppliers and Parts" data that everyone uses these days. Of course what would be really nice would be:

```
select photo from ym.Stone where Id=467
```



Stone with id

… but that one will have to await the next update to the Dyalog session!

Roundup

This is first and foremost a programmer's toolkit, and I would expect to use the functional form of `db.Select` nearly all the time. The stuff with queries and the session I see mostly as a handy debugging tool, but you never know. That is really how q got started, and look what happened to it. The session is a much under-

appreciated resource, so exploiting it in this simple way was good fun, and may develop into something useful. Let me know.

Postscript

As always in live talks, I overstated my case in a few places. In particular I annoyed several people by slagging off Oracle (and its friends) rather too enthusiatically. My SQL primer is dated 1985 and was written by one Lawrence Ellison (now whatever happened to him?) so I am definitely a touch behind the pace. However it would be good if some of you SQL buffs out there could pick up the challenge and show me how to code up:

```
t2:select sum qty by p.color from sp
```

```
/s)select p.color,sum(sp.qty) from sp,p where sp.p=p.p group by
p.color order by color
```

This is a simple example from the scripts shipped with personal kdb+ where the standard sql is shown alongside the q dot-notation. I agree that you *can* do this in standard sql, but the dot-notation is so much cleaner, that the 'tool of thought' argument comes into play even here. Now for a couple of flipdb examples which look like more of a challenge!

```
⍝ Suppliers with at least 700 in shipments
 select sno,sname,city from s where 700≤sum sp[sno].qty
⍝ Who supplies ALL parts
 select sno,sname,city from s where sp[sno].pno seteq p[].pno
```

Or even just my rather trivial home-oriented database:

```
   usage←(delta meter)÷delta date '##0',notes from lecky
   select date,date.weekday,meter,usage
 date        weekday  meter  usage  notes
 ----        -------  -----  -----  -----
 23/09/08 │  Tue      11908     12  Kitchen fridge off
 24/09/08 │  Wed      11919     11
 25/09/08 │  Thu      11929     10  Outside fridge off
 26/09/08 │  Fri      11940     11
 27/09/08 │  Sat      11949      9
...
```

This clearly needs more thought and probably a bit more work, but if Arthur can take the ultimate pragmatist approach of switching parser on seeing `select` then maybe APL could too. Keep watching this space.

Functional calculation

2: The year 1997

by Neville Holmes
neville.holmes@utas.edu.au

Functional calculation does with operations applied to functions and numbers what numerical calculation does with functions applied to numbers. In preceding articles an introduction was given to what could be done with one commonly available tool for functional calculation, using a notation called J, then details were given of simple numerical calculation. This article is intended to allow the reader to consider how simple numerical calculation can be done in J by showing numeric expressions to produce whole numbers below `100` starting from the digits of the number `1997`.

Numerical calculations

Preceding articles have introduced numerical calculation using the interpreter for the J notation. This article gives a change of pace in which the notation already introduced is used in modest examples so that the reader can get used to its differences from the more usual (though inconsistent) mathematical notation.

To set up some simple examples, expressions to produce all the non-negative integers of fewer than three digits are to be sought. There are several restrictions.

Firstly, only the following scalar primitive functions are to be used, though clearly many of them are not very useful – there are more comparisons than are needed, and *nand* and *nor* have very restricted domains.

+	conjugate	add	+.		GCD	+:	double	nor
-	negate	subtract	-.	not		-:	halve	
*	signum	times	*.		LCM	*:	square	nand
%	reciprocal	divide				%:	square root	root
|	magnitude	residue						
^	exp	power	^.	$\log_e$	logarithm			
=		equal						
<		less than	<.	floor	lesser	<:	decrement	not more
>		more than	>.	ceiling	greater	>:	increment	not less
						~:		not equal
!	factorial	choices	o.	pi times	circular	p:	prime	
[	(same)	(left)						

Secondly, only the digits `1 9 9` and `7` must be used as arguments, all those digits, and only in the sequence given. Thus up to four arguments may be used in an expression, but no argument may be a list, that is, all arguments must be scalar. The significance of this will be clearer later.

In the following, two expressions are given for each number, both because there is room for them, and to add interest. The reader should, as an exercise, look for solutions better in some way than those given – there should be plenty, and looking for them should lead to insights, whether the search is successful or not.

One avenue to explore is the use of numbers other than simple integers. These are avoided in the examples but, for instance, `19+<.|9j7` gives `30` using only three functions, which is fewer than in the expressions given below. Similarly, `>.%:1p9-9p7` uses fewer functions to get `52` than do the examples.

Making 1997 give 0 to 19

Taking the four digits `1 9 9` and `7`, in that sequence, combine them using J functions and operations, in as short and simple an expression as possible, to yield each of the numbers between `0` and `19` (here, `99` later), and to yield them as scalars using only scalar arguments and scalar functions. (As a matter of aesthetics, parentheses and decimal points are also avoided as far as possible. Also as a matter of style, the negative sign is avoided and subtraction or negation is used to similar effect.)

0	19=97	^.*1997	10	19-9>.7	|1-9+9-7
1	*1997	199>7	11	19-9-*7	19--:9+7
2	19|97	+:*1997	12	1+9+9-7	19-9|7
3	19-9+7	199-*:+:7	13	-:19+9|7	1*9+%:9+7
4	19|9^7	1+%:9s<.97	14	199|+:7	1+9+%:9+7
5	19-+:9|7	19|>.^.97	15	>.19%9%7	1+9+>.^.97
6	19|9*7	1+9-%:9+7	16	19|9+7	<.19*9^.7
7	1!9-9-7	19|9!+:7	17	1+9+9|7	19-9-7
8	1+9|97	19|%:>:9*7	18	19-*97	1+9+9-*7
9	19<.9>.7	19-9+*7	19	1*p:9-9-7	1+9+9>.7

Two expressions are given in the table for each number. Several expressions use only the one primitive function, but none needs more than four.

Expressions abound for `0` and `1`, particularly expressions using functions that yield logical values. Thus `1>997` and `199<7` yield `0`, while `19<97` and `19+.97` yield `1`. The restriction of yielding a scalar rules out `/:1997` and `=1997` which otherwise give a quite respectable `0` and `1`. Also, `#1997` gives a scalar `1`, and `#19 97` gives a scalar `2`, but the restriction on scalar components rules the second of these two possibilities out of order.

Certain components of expressions are repeatedly used in the table above. Thus, `199|` (and `19|`) is used to pass over the digits of the left argument of `|` when the value of its right argument is lower than that of its left. In particular, `9|7` simply yields `7`. Otherwise, `<.` or `>.` are widely used where the digits to the right or left of the `<.` or `>.` are to be ignored and the simpler and more pleasing `|` won't serve instead. Notice, though, that the instances of `19|` in `19|9^7` and `19|9*7` and `19|9!+:7` are not ignorant.

Expressions starting with monadic increment (`>:`) and decrement (`<:`) functions can often make a shorter expression than the example given, but since these will only be trivially different from neighbouring expressions they are avoided if possible. Monadic halve (`-:`) and double (`+:`) are useful, but these are also a bit trivial for generating smaller or larger numbers.

The year 1997 is interesting in the scope it gives for the square root function (`%:`). In the above, `%:9<.97` is used to give `3`, `%:9+7` is used to give `4`, and `%:>:9*7` is used to give `8`. In fact `-:9+7` and `9-*7` also give `8`, but maybe not so cutely. Otherwise, `>.*:1.997` could have been used for `4` and `>.^.1997` for `8`, both of which are quite interesting.

Making 1997 give 20 to 99

Making numbers beyond 19 follows a similar pattern, and it is convenient here to take them twenty at a time. Of course, expressions starting 19+ will be common in the next table.

20	19+9>7	19+*97	30	19+9++:*7	1+9++:9+*7
21	19+9-7	19+>:*97	31	<.19*9%:7	<.%:-:1997
22	<.-:%:1997	>:19+9-7	32	1+<.%:997	+:19|9+7
23	19+%:9+7	p:1+9-9-7	33	>.19^.^97	1+9>.+:9+7
24	19-9-+:7	19+p:9+7	34	+:19-9-7	1!<.9*9^.!7
25	19+9|<:7	1!9+9+7	35	19+9+7	>.19+^.9^7
26	1+9+9+7	19+9|7	36	+:19-*97	1!9*>.9^.!7
27	19+-:9+7	19+9-*7	37	1+-:9+9*7	>.^.199^7
28	19+9>.7	1+99|!<:7	38	1+p:9+9-7	19*9-7
29	>.199%7	19+9+*7	39	-:-19-97	<:+:19+9>7

At a pinch, all the numbers here, and in the next table for that matter, can be constructed from the solutions of the previous table by doubling (monadic `+:`) possibly combined with incrementing (monadic `>:`) or decrementing (monadic `<:`). However, these solutions will only reluctantly be used here, in the absence of some other expression.

Getting these larger values is somewhat more difficult, so it useful more often to go to non-integer intermediate values and then use the floor (`<.`) or ceiling (`>.`) function to get an integer. There are no solutions given in the 20-39 table which need only the one function, but quite a few need only two. On the other hand, no solutions need more than five functions.

40	+:19+*97	1-9-<.-:97	50	-:1+99>.7	1-9-9+*:7
41	1!<.9*^.97	1+9+-:<:9*7	51	1+99-*:7	19++:9+7
42	+:19+9-7	19+9++:7	52	+:1+9+9+7	1+>.99%^.7
43	>.%:19*97	1*<.9*9^.!>:7	53	1*-:99+7	|1+9-9*7
44	-19-9*7	<.%:1997	54	1+-:9+97	-.1+9-9*7
45	>.%:1997	>.^.19!97	55	1-9-9*7	-.1*9-9*7
46	1!-:99-7	19+9+p:7	56	1+-.9-9*7	19+>:9*-:>:7
47	1+-:99-7	1!>.9*^.+:97	57	-:1+99++:7	19*<.9^.!7
48	1+>.*:^.997	-1-9-9-*:7	58	-:19+97	19>.9+*:7
49	*:199<.7	-:1+9>.97	59	1+9+-:>:97	19-9-*:7

For the 40s and 50s, `9*7` gives 63 to work down from, and `-:97` and `-:99` gets into the high 40s, as does `*:7`. At least one expression is given for each of these numbers which does not need more than four primitive functions.

60	>.19*o.9>7	1+<.+:%:9*97	70	+:19+9+7	|1-<.9*%:9*7
61	-:199|!>:7	>.%:+:19*97	71	-1-9+9*7	-:>.^.%:19^97
62	19>.<:9*7	-:>:199|!<:7	72	-.1-9+9*7	1*-:9*9+7
63	1>.9*9<.7	-.1-9>.9*7	73	1+9+9*7	19+9*<:7
64	1+9>.9*7	1+9+9*<:7	74	>.19*9^.!7	<.19++:^.9!7
65	199|!7	199|!+:7	75	-:199-*:7	19+9+p:+:7
66	-1-9+9+*:7	1*<.9*%:9*<:7	76	19*%:9+7	19*>.9^.!7
67	19+<.-:97	19+-:<:97	77	>.19*o.9%7	>.+:1+%:99*7
68	19+>.-:97	19+-:>:97	78	-19-97	19+p:9+7
69	>.19*^9%7	19++:9++:7	79	-.19-97	>.19*^.9*>:7

For the 60s and 70s, `9*7` provides a good starting point. Of interest are the two expressions for 65, which look the same but are interestingly different, and otherwise only 78 and 79 have expressions with only two primitive functions. There is plenty of room for improvement here, though only 66 is shown as needing more than five primitive functions.

80	1-+:9--:97	1*99-p:7	90	>.+:%:1997	19+<.9^^.7
81	1>.9*9>.7	-.1-9+9*>:7	91	1+99|!7	-1-99-7
82	19+9*7	1+9+9*>:7	92	1!99-7	-.1-99-7
83	>:19+9*7	1++:<.9*^.97	93	1+99-7	1--:>:9-+:97
84	+:+:19+9-7	1+>.+:9*^.97	94	>:1+99-7	1--:<:9-+:97
85	1!99-+:7	<.19*^9%<:7	95	19*>.^.97	19*>:%:9+7
86	<.19*^.97	1+99-+:7	96	-1-9>.97	-:199-7
87	|1+9-97	>.19*^.97	97	19>.97	|1-99-*7
88	-.1+9-97	>.1*9*%:97	98	1+9>.97	-.1-99-*7
89	1-9-97	-:<:199|!>:+:7	99	1>.99>.7	-:199-*7

The 80s and 90s often use 99 or 97 and work down from there. This gives the quite short expressions shown here for 89 and 97, though `9*7` and 199 find occasional use.

In this group of numbers there is the one expression with only one primitive function, seven with two, lots with three, but the improvement to be sought is in those with five or six primitive functions.

Further examples

The examples given above can only suggest how arithmetic functions can be used in a simple to produce a variety of numbers. The reader is urged to consider the examples above with a J interpreter to hand, to try the examples out, to check them, and to try to find expressions that are better or in some way more interesting than those given here. When generating these numbers begins to pall, the reader perhaps should go on to consider how to generate the three digit numbers using the same rules. This could start `1+99>.7` then `199-+:*:7`.

Alternatively, expressions might be sought for other years. Some years will present special challenges. The following table gives a start for the year 2000, in which a choice is made between `0=0`, `0!0` and `0^0` to give a 1 largely on aesthetic grounds.

0	200=0	^.*2000	10	20%+:0=0	20->.o.o.0!0
1	*2000	200^0	11	-:20++:0!0	<.^+:*:200+0
2	2+0-0*0	+:*2000	12	20-<.o.^0=0	+:<.%:%:2000
3	2+0=0+0	2!>.0+^0=0	13	<.+:%:%:2000	p:<.^.200-0
4	2*+:0<0!0	>.^.20+0=0	14	<.%:200+0	<.^.-:*:2000
5	<.^.200+0	20-<.^^0=0	15	>.%:200-0	<.+:^.2000
6	<.%:%:2000	>.^.200+0	16	>.^.*:2000	20->.o.0=0
7	<.^.2000	>.^.-:2000	17	20->:+:0!0	>.+:^.+:2000
8	2^p:0+0!0	2^0]>:+:0=0	18	20-+:0=0	>.!o.%:2000
9	<.-:20-0!0	>.^.p:2000	19	20-0!0	>.^.p:*:2000

Apart from reducing the number of possibilities, having all those nought digits tends to give unsightly numbers like `00` and `000` which are therefore avoided here as far as is convenient. Thus, although `2+000` would be technically correct for 2, the expression `2+0-0*0` is shown, though many similar ones would be just as satisfactory. Not having a 1 at the front also reduces the possibilities a lot!

Another amusing possibility, though ultimately monotonous because expressions are restricted to monadic functions, is to try to develop all the numbers from only a single zero. A start to this is given in the following, and the aesthetic choice is between `!0`, `^0`, `>:0`, and `-.0` to get the initial 1, or `p:0` to get an initial 2.

0	`|0`	`*0`	10	`>.*:o.!0`	`>.o.o.>:0`
1	`!0`	`^0`	11	`<.-:^o.^0`	`p:+:p:0`
2	`+:-.0`	`p:0`	12	`-:!+:+:!0`	`-:>.^o.-.0`
3	`p:-.0`	`>.^^0`	13	`<.*:>:^^0`	`p:p:p:0`
4	`*:+:!0`	`>.o.>:0`	14	`+:<.!o.!0`	`+:<.*:^^0`
5	`>:+:+:>:0`	`p:p:0`	15	`<.^^^0`	`<:*:*:+:!0`
6	`!>:+:!0`	`+:>:+:-.0`	16	`*:*:>:>:0`	`*:+:+:-.0`
7	`<.^+:^0`	`p:p:^0`	17	`<.o.+:^^0`	`p:+:p:^0`
8	`<.o.^^0`	`>.^p:0`	18	`+:>.o.^^0`	`+:*:>:+:!0`
9	`*:>:+:!0`	`>.o.^^0`	19	`<:<.^>.^!0`	`p:p:>:p:0`

The shortness of some of expressions given here comes often from using the `o.` and `^` functions to bump up values quickly, then using floor (`>.`) or ceiling (`<.`) functions to get integers. Otherwise `p:` can be used most conveniently. Of course, where an expression starts with `<.` the next higher number can be got by substituting a `>.` and vice versa, mutatis mutandis.

The expressions given in these last two tables were selected unsystematically and hastily. The assiduous reader should have an interesting but fruitful time looking for improved and extended expressions.

Spice for Beginners

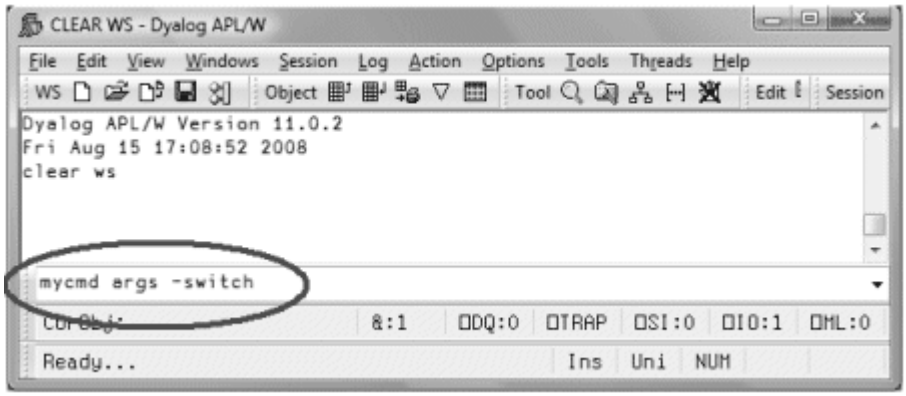
by Dan Baronet
danb@dyalog.com

Spice is a Dyalog development tool introduced with V11 in 2006. It allows you to execute code independently of the current workspace status. It works in conjunction with SALT. For those of you familiar with APL+'s User Commands this will sound familiar: APL+[1] uses the right bracket to invoke a *user command* as in

```
]XYZ
```

to execute[2] user command `XYZ`. It brings in all the code necessary, localising it before running it.

Dyalog does something similar, using an input window above the status line instead of the `]` syntax of APL+.



Upon hitting Enter, the content of the window is sent to the Spice processor, which then identifies the selected command, gets and localises the code to run it, runs it, then cleans up. Just as APL+ does.

And just like APL+, entering `?` displays a list of available commands, and `?XYZ` displays help about command XYZ.

To write a Spice command, a few rules must be followed. You must:

- Write a class containing the code
- Put that class in a file in the Spice folder with a `.dyalog` extension

- Define at least the minimum three public shared functions: `List`, `Help` and `Run`

A Spice class may be host to several (related) commands. Or just one.

Example 1: The TIME command

Here is a very simple example: let's say we want to create a Spice command that will show us the current time.

We first create a class that will handle the group of time-related functions:

```
:Class timefns
    ⎕ML ⎕IO←1 ⍝ always set here to avoid inheriting external values

    ∇ r←List
      :Access Shared Public
      r←⎕NS¨1⍴⊂''
      r.(Group Parse Name)←⊂'TimeGrp' '' 'Time'
      r[1].Desc←'Time example Script'
    ∇

    ∇ r←Run(Cmd Args)
      :Access Shared Public
      r←⎕TS[4 5 6]   ⍝ show time
    ∇

    ∇ r←Help Cmd
      :Access Shared Public
      r←'Time (no arguments)'
    ∇
:EndClass
```

The `List` function is used to describe the command to Spice. Spice is thus able to display a minimum of information when you type `?` in the Spice command line. This information is stored in `Desc`. Three more variables must be set: the command Name, the Group it belongs to and the Parsing rules. We'll get to those rules in a bit.

The `Help` function is used to report more detailed information when you type `?time` in the Spice command line. Since the class may harbour more than one command the function takes an argument. Here there is only one command and

the argument will always be `time` so we ignore it and return some help for the command `time`.

The `Run` function is the one executing your code for the command. It is always called with two arguments. Here we ignore them as all we do is call `⎕TS`. Easy peasy. We can write this code in a `timefns.dyalog` file using Notepad and put it in the SALT\Spice folder or write it in APL and use SALT's Save command[3] to put it there.

Once in the Spice folder is it available for use. All we need to do is move the cursor to the Spice command line and type time. *Et voilà!* The current time appears in the session as three numbers[4].

Example 2: Another command in the same class: UTC

We may want to have another command to display the current UTC time instead of the current local time. Since this new command is related to our first `time` command, we could – and should – put the new code in the same class, adding a new function `Zulu`[5] and modifying `Run`, `List` and `Help` accordingly. Like this:

```
:Class timefns
    ⎕ML ⎕IO←1

    ∇ r←List
      :Access Shared Public
      r←⎕NS¨2⍴⊂''
      r.(Group Parse)←⊂'TimeGrp' ''
      r.Name←'Time' 'UTC'
      r.Desc←'Shown local time' 'Show UTC time'
    ∇

    ∇ r←Run(Cmd Args);dt
      :Access Shared Public
      ⎕USING←'System'
      dt←DateTime.Now
      :If 'utc'≡⎕SE.U.lcase Cmd ⋄ dt←Zulu dt ⋄ :EndIf
      r←(r⍳' ')↓r←⍕dt ⍝ remove date
    ∇
```

```
    ∇ r←Help Cmd;which
      :Access Shared Public
      which←'time' 'utc'⍳⊂⎕SE.U.lcase Cmd
      r←which⊃'Time (no arguments)' 'UTC (no arguments)'
    ∇

    ∇ r←Zulu date
     ⍝ Use .Net to retrieve UTC info
      r←TimeZone.CurrentTimeZone.ToUniversalTime date
    ∇
:EndClass
```

The `List` function now accounts for the `UTC` command and returns a list of two namespaces so `?` will now return info for both commands. Same for `Help` which makes use of a `lcase` utility in `⎕SE.U`, a namespace of short utilities for use by SALT, Spice or anyone.

The `Run` function now makes use of the Cmd argument and, if it is `utc`, calls the `Zulu` function. It then returns the data nicely formatted, an improvement over the previous code.

Example 3: Time in cities around the world

We could then add a new function to tell the time in Paris, another one for Toronto, etc. Each time we would have to modify the three shared functions above *or* we could have a single function that takes an argument (the location) and computes the time accordingly.[6] Like this:

```
:Class timefns
    ⎕ML ⎕IO←1

    ∇ r←List
      :Access Shared Public
      r←⎕NS¨2⍴⊂''
      r.(Group Parse)←⊂'TimeGrp' ''
      r.Name←'Time' 'UTC'
      r.Desc←'Shown local time in a city' 'Show UTC time'
    ∇
```

```
    ∇ r←Run(Cmd Args);dt;offset;cities;diff
      :Access Shared Public
      ⎕USING←'System'
      dt←DateTime.Now ⋄ offset←0
      :If 'utc'≡⎕SE.U.lcase Cmd
          cities←'honolulu' 'montreal' 'copenhagen' 'sydney'
          offset←¯10 ¯5 2 10 0[cities⍳⊂⎕SE.U.lcase Args]
      :OrIf ' '∨.≠Args
          dt←Zulu dt
      :EndIf
      diff←⎕NEW TimeSpan(3↑offset)
      r←(r⍳' ')↓r←⍕dt+diff ⍝ remove date
    ∇

    ∇ r←Help Cmd;which
      :Access Shared Public
      which←'time' 'utc'⍳⊂⎕SE.U.lcase Cmd
      r←which⊃'Time [city]' 'UTC (no arguments)'
    ∇

    ∇ r←Zulu date
     ⍝ Use .Net to retrieve UTC info
      r←TimeZone.CurrentTimeZone.ToUniversalTime date
    ∇
:EndClass
```

Here `List` and `Help` have been updated to provide more accurate information but the main changes are in `Run` which now makes use of the `Args` argument. This one is used to determine if we should use the `Zulu` function and compute the offset from UTC by looking it up in the list of cities for which we know the time offset.

The first argument to `Run` is always the command name (here it is called `Cmd`) and the second argument is whatever you entered after the command (here it is called `Args`). When there are no special rules this argument will always be a string.

For example, if we enter in the Spice command line:

```
time Sydney
```

`Cmd` will contain `'time'` and `Args` will contain `'Sydney'`.

Special rules

There are times when it is easier to make a command accept variations than to write an entirely new command. A command switch (also known as *modifier* or *flag* or *option*) is an indication that the command should change its default behaviour.

For example, in SALT, the command `list` is used to list files in a folder. The command accepts an argument to restrict the files to list (e.g. `a*` to list only the files starting with `a`) and accepts also some switches (e.g. `-versions` to list all the versions). Thus the command `list a* -ver` will only list the files starting with `a` with all their versions instead of listing everything without version, which is the default.

In Spice the same thing is possible but this time you decide which switches are acceptable. When no rules are given to Spice via the `Parse` variable (in the `List` function) `Arg` is a string and you can do whatever you wish with it. If your command is to accept switch `-x` then you can look for a `-x` in the string and make a decision about that. It can become quite tedious to have to deal with the handling of switches every time you write a new command. Without getting into too many details let's say that Spice takes care of that for you.[7] It handles switches the same way SALT does.

Let's have a look at a more complex example.

Example 4: The sample command

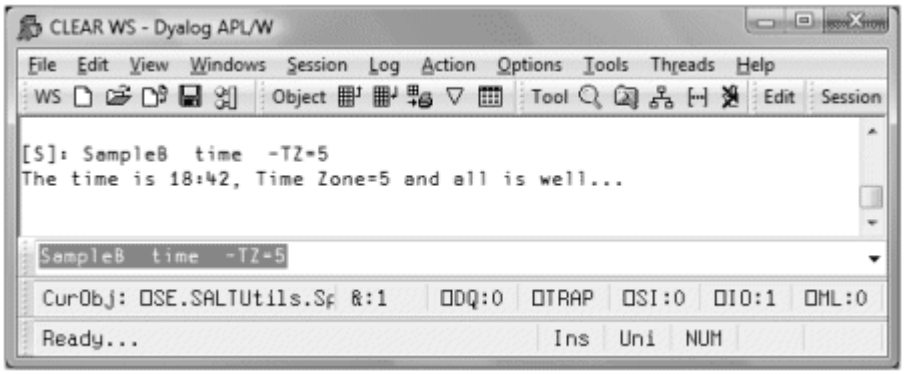
Spice comes with a sample command to demonstrate the use of arguments and switches.

In file `aSample.dyalog` you will find a class with two commands: one which does not use the parser, and one that does.

The second command, named `sampleB`, uses the parser. It is similar to the `time/utc` command above: it accepts one and only one argument and one switch, called `TZ`, which *must* be given a value. For example you could write:

```
Sampleb   time  -TZ=-5
```

or, as seen in 'real life':



Spice is unable to validate the contents of the argument but it can determine that there is only one argument. It can also ensure that `TZ`, if supplied, is given a value and that no other unknown switch appears.

The way to tell Spice about this is to set the `Parse` variable for that command in `List` to `1 -TZ=`.

The `1`, here, means that one and only one argument must be present and `-TZ=` means:

> use - as the switch delimiter, accept TZ as valid switch for the command, and make sure a value is supplied (with =) whenever it is used.

Switch names are case-sensitive and must also obey the rules for APL names.

If you don't specify the number of arguments, Spice won't check, and you can have as many or as few arguments as you wish, including none.

When your command is used Spice will check those conditions and if anything breaks the parsing rules it will complain and abort execution. It all goes well Spice will package the argument and switch(es) into a namespace and pass it on to `Run` in Arg.

`Arg` will contain `Arguments`, a list of text vectors (here only one) containing each one of the arguments and `TZ`, which will be either the scalar number 0 (if it was not specified) or the string given as value if it was specified.

Let's go over that again.

Here's what the user enters in the Spice command line:

```
sampleb  xyz -TZ=123
```

Spice will validate this, find it is OK since there is only one argument, `xyz`, and that the switch `TZ` has been given a value, here `123`.

It will then call `Run` with `sampleB` and a namespace in which `Arguments` is `,⊂'xyz'` and `TZ` is `'123'`. It is then up to the program to determine if this all makes sense.

Here's another example:

```
sampleB  x  y  z
```

Here three arguments have been supplied: `x`, `y` and `z` and Spice won't allow it:

```
[S]: sampleb x y z
Command Execution Failed:
too many arguments
Spice[64] arg←(⎕NEW ⎕SE.Parser rules).Parse arg
         ^
```

Another example:

```
SAMPLEB  'x  y  z'  -TZ
```

Here there is only *one* argument, as quotes have been used to delimit the argument of 5 characters: `'x  y  z'` *but* the switch `TZ` has not been given a value so:

```
[S]: sampleb 'x y z'  -TZ
Command Execution Failed:
value required for switch <TZ>
Spice[64] arg←(⎕NEW ⎕SE.Parser rules).Parse arg
         ^
```

One more:

```
Sampleb  zyx  -TT=321
```

Here is one argument, which is OK, but TT is not a recognized switch and:

```
[S]: sampleb zyx -TT=321
Command Execution Failed:
unknown or ambiguous switch: <TT>
Spice[64] arg←(⎕NEW ⎕SE.Parser rules).Parse arg
         ^
```

What if we don't supply *any* argument?

```
Sampleb  -T=xx
```

```
[S]: sampleb  -T=xx
Command Execution Failed:
too few arguments
Spice[64] arg←(⎕NEW ⎕SE.Parser rules).Parse arg
         ^
```

Here we supplied a proper `TZ` switch (Spice was able to determine that `T` stood for `TZ`) but 0 argument was not enough and therefore it complained.

As you can see Spice can be clever enough to figure out the number of arguments and which switches have been set and their values. The rules are fairly simple:

- All commands take 0 or more arguments and accept 0 or more switches
- Arguments come first, switches last
- Arguments are separated by spaces
- A special character (delimiter) identifies and precedes a switch
- Switches may be absent or present and may accept a value with the use of `=`
- Switches can be entered in any order
- Arguments and switch values may be surrounded by quotes (`'` or `"`) to include spaces and/or switch delimiters.

Spice, after verifying that the rules are being followed correctly, will put all the arguments (the space delimited tokens) into variable `Arguments` in a new namespace. It will also put in there variables of the same name as the switches. The namespace is then passed as the second argument to `Run`, which then runs.

There are a few more things the parser can do, but this should cover most cases. For a complete list, have a look at the documentation for Spice on the Dyalog site.

Notes

1. *APL+* refers to the APL/PC, APL/II, APL+Win family
2. This is similar to the way the system recognizes its own commands, i.e. the use of a right parenthesis, e.g. `)CLEAR` is a system command whereas `]CLEAR` would be a user command.
3. `⎕SE.SALT.Save 'timefns Spice\timefns'` will do it

4. This requires SALT/Spice version 1.3 or more. To see which version you are using type
   ⎕SE.SALT.Version

5. UTC is sometimes denoted as *Z time* – Zero-offset zone time – or *Zulu time* from the NATO phonetic alphabet.

6. The function does not deal with daylight savings time. An exercise for the reader?

7. If you wish to delve into this subject, have a look at *Vector* Vol 19.4: "Tools, Part 1. Basics."

IN SESSION

Congratulations not in order?

by Stephen Taylor
sjt@5jt.com

The card popped out of its envelope bearing the single word *Congratulations* surrounded by what looked like anagrams of it. This kind of thing preys on one's mind. What *was* my sister thinking? I *don't* have time on my birthday to stare at the phrases and decide whether they really *are* all anagrams or just *look* like they are, or just *some* of them are. Need to resolve this question quickly and get back to typesetting *Vector*.

Let's see… if we alphabetise… and ignore spaces:

```
      {ω[⍋ω]~' '}
```

then all that's left is to compare the results…

```
      'congratulations'{⊃≡/{ω[⍋ω]~' '}¨α ω}'iron nuts catalog'
1
```

Gluing on the left argument gets me a monadic test function.

```
      test←'congratulations'∘{⊃≡/{ω[⍋ω]~' '}¨α ω}
      test¨↓⎕←↑ANAGRAMS
nora taunts logic
iron nuts catalog
using cool tartan
caution long star
stout groin canal
so not a garlic nut
ration aunts clog
uncool giant rats
auto girls cannot
tuscan train logo
trust colon again
groan until ascot
snail conga tutor
1 1 1 1 1 1 1 1 1 1 1 1 1
```

There we go. Anagrams tested, male autism appeased.

J-OTTINGS 51

Suffer the little children to bring their homework…

to Norman Thomson

Early articles on APL sometimes speculated on how to do a work-around if one of the APL function keys was broken. Analogously one of the properties of 'clock arithmetic' as taught in the early stages of primary schools is that division is a disallowed (broken key!) operation. This reflects the historical fact that the concepts of division and fractions came relatively late in mankind's mathematical development. Despite their arithmetical skills and geometrical sophistication, the early Greek and Roman mathematicians had only crude notions of division into parts, and it was not until the invention of place value in the 11th century that fractions in the sense we know them today became part of the earliest stages of elementary arithmetic.

J adverbs provide a natural means for realising the concepts of finite arithmetic, for example the derived verb `+mod` is addition in modulo arithmetic:

```
   mod=.1 :'n&|@x.'      NB. 'define modulus' adverb
   n=.7                  NB. set modulus to n
   3+mod 6               NB. add 3 and 6 (mod 7)
2
   3*mod 6               NB. multiply 3 and 6 (mod 7)
4
```

Graduating from clock arithmetic to 'clock algebra' is where some at least of the little children might begin to suffer, since division in the conventional sense is no longer an option for solving e.g. 2x=3. This is known in clock arithmetic as a congruence rather an equation. In modulo 5 arithmetic, it is not too difficult to spot that x=4 as a solution. However solving 17x=5 in modulo 23 arithmetic is a little more difficult. If the inverse of 17 were known (that is the solution of 17x=1) then the solution of 17x=5 is simply 5 times 17-1:

```
   n=.23                 NB. set modulus
   (17*mod i.n)i.1       NB. locate 1 in multiples of 17
19
   5*mod 19              NB. solve 17x=5
3
```

This is readily confirmed by multiplying 17 times 3 = 51 = 5 in modulo 23 arithmetic. The second line in the above J sequence suggests a general technique for solving linear equations (congruences) of the form ax + b = 0 by first defining inverse as

```
   inv=.4 : '(x.|y.*i.x.)i.1'
   23 inv 17
19
```

(tacit definition enthusiasts may want to write this as `inv=.i.&1@([|]*i.@[)`, although arguably the above version is more expressive.) It is easy to confirm that in modulo p arithmetic (p prime), all integers in 1, … p-1 have an inverse, for example:

```
   iota=.>:@i.           NB. integers from 1 to y.
   13|t*13 inv&>t=.iota 12
1 1 1 1 1 1 1 1 1 1 1 1
```

A first try at a solution of the linear equation ax + b = 0 is then

```
   lsol=.4 : 'x.|(x. inv {.y.)*(-{:y.)'
   23 lsol 17 _5
3
```

However, inverses exist only for numbers which are relatively prime to the modulus. In clock arithmetic terms 17x=5 is an invitation to find how many chunks of 17 steps are needed to arrive at 5, to which the answer is 3. But for 2x=5 in modulo 6 arithmetic no solution exists because 2 and 6 have a common factor, which means that only some of the clock points are reachable. On the other hand 2x=4 has two solutions, x=2, the 'obvious' one, and also x=2+3=5. To generalise this, the number of solutions of ax=b in modulo n arithmetic is either none if b is not a multiple of GCD(a,n), otherwise it is GCD(a,n), in which case these solutions are found by adding {n/GCD(a,n)} successively GCD(a,n) times to the solution of ax=b after a, b and the modulus n have all been divided by GCD(a,n) :

```
linsol=.4 : 0                          NB. linear equation solver
if.1=gcd=.x.+.{.y.                     NB. if a and n are co-prime
   do.x.lsol y.
elseif.0=gcd|{:y.                      NB. if gcd(a,n) divides b ..
   do.(m lsol y.%gcd)+(m=.x.%gcd)*i.gcd
end.                                   NB. otherwise null result
)

   21 linsol 6 _15                     NB. solns of 6x=15 in mod 21
6 13 20
```

The divide symbol `%` appearing in the long line of `linsol` reflects the 'cancellation' of ax=b to its prime form, and does not conflict with the disallowance of divide in clock algebra.

ax=b has now been solved with complete generality.

Simultaneous linear congruences

Unlike ordinary simultaneous equations where, barring degeneracies, the number of equations must exactly equal the number of variables for there to be a unique solution, there is no limit to the number of simultaneous congruences for which a solution can be sought. Further, a theorem called the Chinese Remainder Theorem (so called because such results were known in China from about 100 A.D.) guarantees that *provided the various moduli are coprime*, then a set of simultaneous equations such as

$$x=0 \pmod 2,\ x=1 \pmod 5,\ x=2 \pmod 7$$

has a solution which is unique modulo the product of moduli.

The algorithm for obtaining such a solution consist of multiplying three lists:

1. a list of the b's as in ax=b;
2. a list of products of the a's omitting one at a time; and
3. the inverses of the products in (2) relative to their matching moduli,

and then multiplying the items of the resulting list modulo the product of moduli. This is described as readily, and certainly more unambiguously, in J with as input

Left arg (x.) a list of moduli – these must be coprime;

Right arg (y.) a matching list of pairs of coefficients as for `linsol`.

```
each=.&.>

simlsol=.4 : 0    NB. simultaneous congruences
r1=.>-@{:each y.
r2=.(%~*/)x.
r3=.>x. linsol each <"1 r2,._1
r=.(*/x.)|+/r1*r2*r3
)
```

The solution of the above set of congruences is:

```
   2 5 7 simlsol 1 0;1 _1;1 _2
16
```

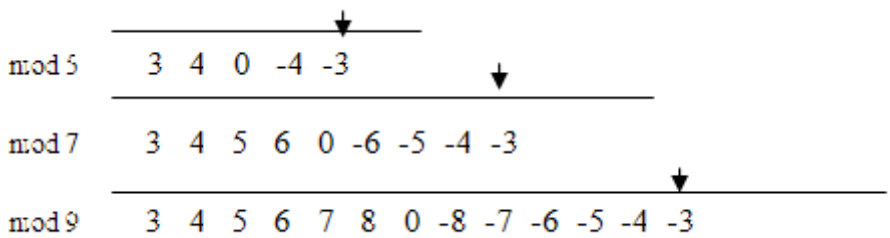
For those who like puzzles `simlsol` lends itself to solutions of problems such as

1. what is the smallest integer divisible by 7 whose remainders on division by 2,3,4,5 and 6 are 1,2,3,4 and 5?
2. what is the smallest integer divisible by 7 whose remainders on division by 2,3,4,5 and 6 are all 1?

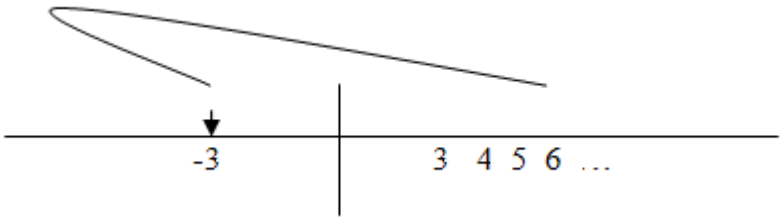
```
   4 3 5 7 simlsol 1 _3;1 _2;1 _4;1 0
119
   4 3 5 7 simlsol 1 _1;1 _1;1 _1;1 0
301
```

Quadratic congruences

Here the simplest case is $x^2=a$. In ordinary arithmetic the solution is simply ± the square root of a, and it is useful to picture how finite arithmetics converge towards normal arithmetic as the modulus increases towards infinity. Represent say -3 by points on number lines corresponding to arithmetics with successively larger moduli:



Eventually the arrow 'goes off to infinity' and returns 'on the other side of zero' as -3 in the conventional sense:



Returning to the problem of solving $x^2=a$ in modulo n arithmetic, Since only integers are admissible, there can only be solutions if a is one of those integers in 1…n-1 which are squares in modulo n, and since $k^2 = (-k)^2$ it is only necessary to consider the range 1…½(n-1) in order to establish all such squares. These are called 'quadratic residues' in finite arithmetics, and are obtained as:

```
   qres=.|*:@:(iota@(-:@<:))        NB. quadratic residues
   qres 13                          NB. squares modulo 13
1 4 9 3 12 10
   qres 17                          NB. squares modulo 17
1 4 9 16 8 2 15 13
   qres 29                          NB. squares modulo 29
1 4 9 16 25 7 20 6 23 13 5 28 24 22
```

so, for example, in modulo 13 arithmetic, the square root pairs of 3, 12 and 10 are (4,13-4), (5,13-5) and (6,13-6), i.e. (4,9), (5,8) and (6,7) respectively, and a must be one of the six values `qres 13` if the equation $x^2=a$ is to have a solution. One such solution is then `1+(qres n)i.a` so, for example one solution of $x^2=5$ in modulo 29 arithmetic is

```
   1+(qres 29)i.5
11
```

and the other is 18, which is confirmed by observing that both 121 and 324 = 5 in modulo 29 arithmetic. This leads to the following definition of a verb which delivers a 'single square root' verb in finite arithmetic:

```
   sqrt=.>:@(qres@[i.])             NB. sqrt of y. in modulo x.
   13 sqrt 12
5
```

This can readily be generalised to find any root:

```
   res=.[ | iota@(<:@[) ^ ]          NB.generalised residue
   13 res 3                          NB.cubes in modulo 13
1 8 1 12 8 8 5 5 1 12 5 12

   iall=.>:@(= # i.@#@[)             NB.iota all (origin 1)

   root=.(({.res{:)@[)iall]          NB.all kth. roots, e.g. …
   13 3 root 12                      NB.cube roots of 12 mod 13
4 10 12
```

Read the above lines as 'in modulo 13 arithmetic, the 3-roots (i.e. cube roots) of 12 are 4, 10 and 12'.

The suite of verbs `res`, `iota`, `iall` and `root` makes `qres` and `sqrt` redundant, and allows the solution of any equation x^n=a. Where no solution exists a null result is returned.

Now turn to the solution of the more general quadratic $ax^2 + bx + c = 0$. In ordinary arithmetic, the solution is found by the technique of completing the square to give the standard formula with 2a as the denominator. With division disallowed, the trick is to multiply the left hand side by 4a to obtain a leading term $(2a)^2x^2$ and factorise $4a(ax^2 + bx + c)$ as $(2ax + b)^2 – (b^2 – 4ac)$. Then write $d = b^2 – 4ac$ and $y = 2ax + b$, so that $ax^2 + bx + c = 0$ becomes $y^2=d$ which has already been solved provided that d is one of the quadratic residues. If not, there are no solutions. So define the verb `disc` standing for 'discriminant' to compute $b^2 - 4ac$ in the ordinary way, so that the discriminant of e.g. $5x^2 – 6x + 2$ is –4:

```
   disc =.(*:@(1&{))-4&*@{.*{:
   disc 5 _6 2
_4
```

`disc`, like other verbs, can be modified with the adverb mod using the current modulus n:

```
   disc mod 5 _6 2                   NB. n is currently 13
9
```

so that the first step in the solution is

```
   13 2 root disc mod 5 _6 2         NB. sq roots of 9 mod 13
3 10
```

All that remains is to transform these two solutions in *y*s back to *x*s, specifically to solve 10x – 6 = 3 and 10x – 6 = 10 (the latter being equivalent to 10x – 6 = –3),

that is the two linear equations 10x – 9 = 0 and 10x – 16 = 0, for which a technique is already available:

```
   13 linsol&>10 _9;10 _16
10 12
```

solutions which are confirmed by

```
   10 12 #.mod&>< 5 _6 2
0 0
```

The technique is consolidated in the verb

```
qsol=.4 : 0                          NB. quadratic solver
t=.(n,2)root disc mod y. [ n=:{.x.
n linsol&><"1(2 1*}:y.)+"1(0,&>t)
)
```

```
   13 qsol 5 _6 2
12 10
```

and confirmation is obtained by

```
   (13 qsol 5 _6 2)#.mod&>< 5 _6 2
0 0
```

Not all quadratics have genuine solutions, and the simplest way to proceed is to execute `qsol` regardless but disregard any solutions which fail the confirmation test above. This leads to a completely general quadratic solver:

```
quadsol=.4 : 0                       NB. general quadratic solver
t=.(n=:x.)qsol y.
if.0 0-:t#.mod&><y.do.t
else. i.0 end.
)
```

```
   13 quadsol 5 _6 2
12 10
   13 quadsol 5 6 _2                 NB. change of coefficients
(null result)
```

the results of which can be checked by

```
   (13 quadsol 5 _6 2)#.mod&>< 5 _6 2
0 0
   (13 quadsol 5 6 _2)#.mod&>< 5 6 _2
(null result)
```

Thus a single session of algebra has provided solutions for all linear and quadratic equations in countless algebras, and all with a quite modest amount of suffering!

P R O F I T

Cauchy curves

by William R. Jones & Cliff Reiter
jonesw@lafayette.edu and *reiterc@lafayette.edu*

Abstract Cauchy's Integral theorem guarantees that certain complex integrals along closed paths are zero. This means that the intermediate values of 'partial path integrals' themselves form paths. We explore the marvelous variety of curves that arise this way using J's excellent facilities for complex arithmetic and plotting.

[To see the figures in full colour, see the article online. *Ed.*]

Introduction

A.L. Cauchy (1789-1857) was the founder of complex integration theory. He made hundreds of contributions to various branches of mathematics and mathematical physics. A brief biography of Cauchy may be found in [1,2] and a detailed discussion of his mathematical work on complex functions appears in [9]. Some modern books on complex analysis include [3,7]

His theorem called the Cauchy Integral Theorem tells us that the integral of a well-behaved complex function around a well-behaved closed path in the complex plane is zero. Such integrals are called 'path integrals' and we write the conclusion of the theorem as follows.

$$\oint_C f(z)\,dz = 0$$

A closed path begins and ends at the same point. Here the closed path is well-behaved if it is a loop around some region in the complex plane and it should be 'rectifiable' (have a sense of length). Suffice it to say circles, ellipses, and squares are well-behaved, but a Koch snowflake [8] is not (it has fractal dimension, not a finite length). Here well-behaved functions are analytic on and inside of the closed path. Suffice it to say that polynomials, the sine, cosine and the exponential are well-behaved (analytic everywhere). Even rational functions are well-behaved so long as the zeros of their denominators do not appear on or inside the closed path.

We consider in this note the set of values assumed by integrals over the course of the path of integration. In each example we approximate integrals of a given function over increasingly longer portions of a given path of integration. We plot the set of values of these 'partial-path integrals' and call the result a *Cauchy curve*. The Cauchy Integral Theorem assures that these Cauchy curves begin and end at zero (thus forming closed paths themselves); along the way, however, marvelous variety can occur.

We feel that the resulting curves are interesting for their complexities, novelties, and simple beauty. Moreover, these curves give a different way of looking at analytic functions. This note investigates Cauchy curves using J's excellent facilities for complex arithmetic, plots, and adverbs. The inclusion of complex arithmetic can be seen in the original dictionary of J [4]. There is recent documentation of J's complex number operations at [5] and a recent *Vector* article includes [10].

First example

For our first example our closed path of integration is the unit circle in the complex plane. We use the fact that the complex function

$$r(t) = e^{it} = \cos(t) + i\sin(t)$$

runs once around the complex unit circle for $0 \le t \le 2\pi$. Here we look numerically at stepping around the circle in three steps and then graphically consider 10000 steps.

```
   require 'plot'

   r=: ^@j.

   r steps 0 2p1 3   NB. sin 2r3p1 is 0.866025
1 _0.5j0.866025 _0.5j_0.866025 1j_2.44929e_16

   $C=: r steps 0 2p1 10000
10001
```

Since we usually want circles to look round, we define a utility to make plots with aspect ratio one and we thicken the curves slightly.

```
   cplot=: 'Aspect 1;pensize 2'&plot

   cplot C
```

That plot is shown in Figure 1.

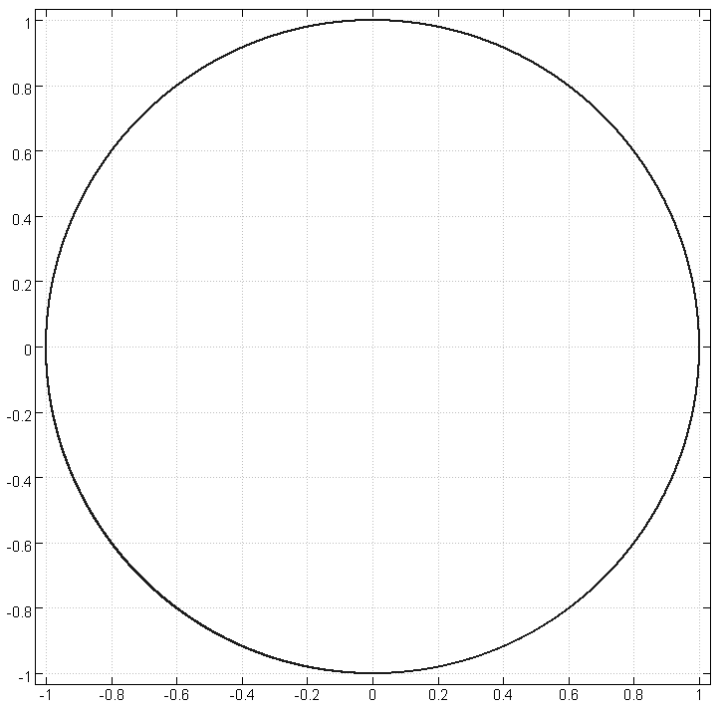


Figure 1. The unit complex circle centered at `0j0`.

For our first example of a Cauchy curve, we will let $f(z) = 1+3z^3$ and use the `C` defined above for the closed path of integration. The 10001 complex numbers in `C` represent the 'z' values in the path integral that we are estimating.

$$\oint_C f(z)\,dz$$

We compute dz by taking differences with the verb `diff`, defined below. We use the function `midpt` to find the points between the `z` values and evaluate `f` at those points (the 'midpoint rule' for integration gives quite good numerical approximations). Thus, dz is `diff C` in J while $f(z)$ is `f mdpt C`. The integral sign means 'limit of sums', so we approximate it by using a sum as follows in J.

```
   diff=: }. - }:

   mdpt=: [: -: }. + }:

   f=: 1 + 3 * ^&3

   +/(f mdpt C) * diff C
1.19488e_14j_3.08564e_15
```

The result should be zero by Cauchy's theorem; it is near 0 to 13 decimal places. To get the Cauchy curve in the above situation we just replace the sum `+/` with a list of partial sums `+/\` and plot the result.

```
cplot +/\(f mdpt C) * diff C
```

The result is shown in Figure 2. This is our first Cauchy curve.

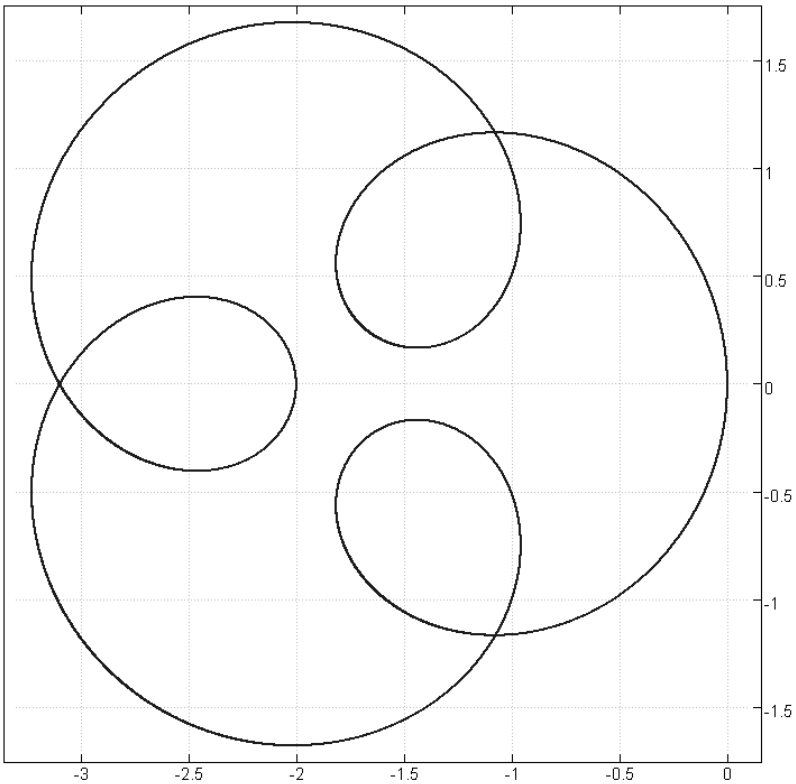


Figure 2. Cauchy curve for f(z) = 1+3z³.

We note that in our computational point of view the curve `C` is given as a numeric list of points and therefore we find the midpoints numerically as above. The integrals could be made more accurate by using midpoints from the parameterization or using higher order integration methods such as Simpson's rule. Since our purpose is plotting curves, 13 decimal places of accuracy are sufficient.

Also, technically we think of paths as beginning and ending at the same point and the first term in the partial sum is typically not zero, thus we pre-append a 0 to the list of partial sums to make the result a closed path in our sense. It is convenient to introduce an adverb `chycu` to compute these Cauchy curves. Its adverb argument is the function being integrated, the right argument is the closed path given as a list of approximating points. The dyad case uses `x&u` as the integrand and this will later be used to compute several curves for one plot.

```
   chycu=: 1 : 0    NB. CAUchy cuRVE
0,+/\(u mdpt y) * diff y
:
x&u chycu y
)

   cplot f chycu C
```

The above line duplicates Figure 2.

Cauchy curves and magnitude

If we plot the Cauchy curve for a monomial `^&n` the result is a circle. Here, and in all the examples until the last section, we again use the unit circle around the origin as our path of integration.

```
   cplot ^&5 chycu C    NB. Cauchy curve is a circle
```

However, the plot of the Cauchy curve as a circle is misleading since it wraps around several times. Any guess as to how many times it wraps? Here we define `cmplot` which shows the Cauchy curve along with the magnitudes of the points on the Cauchy curve plotted in the direction of the angle (`12 o.`) to the corresponding points on the closed path `C`.

```
   cmplot=: 1 : 0
z=. u chycu y
opts=. 'pensize 2;aspect 1;penstyle 0 1'
opts plot z,:(|z)r. 12 o.y
)

   ^&5 cmplot C
   ^&5 cmplot sC=: r steps 0 2p1 100
```

Figure 3 shows such a plot with the unit circle `sC` having a small number of points so that the magnitude curve can be seen in dashed form.

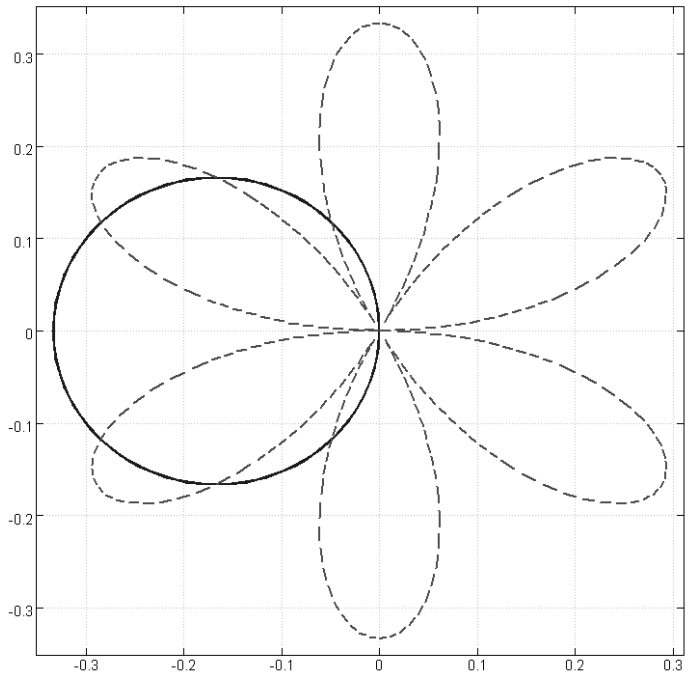


Figure 3. Cauchy curve and magnitude for $f(z) = z^5$.

In general, `^&n` gives a circle that wraps n+1 times. If multiple terms are used (as in Figure 2) in the polynomial, the results can be quite complicated. For example, consider the Cauchy path with magnitude for $f(z) = 1+9z^8$, shown in Figure 4.

```
(1+9*^&8) cmplot sC
```

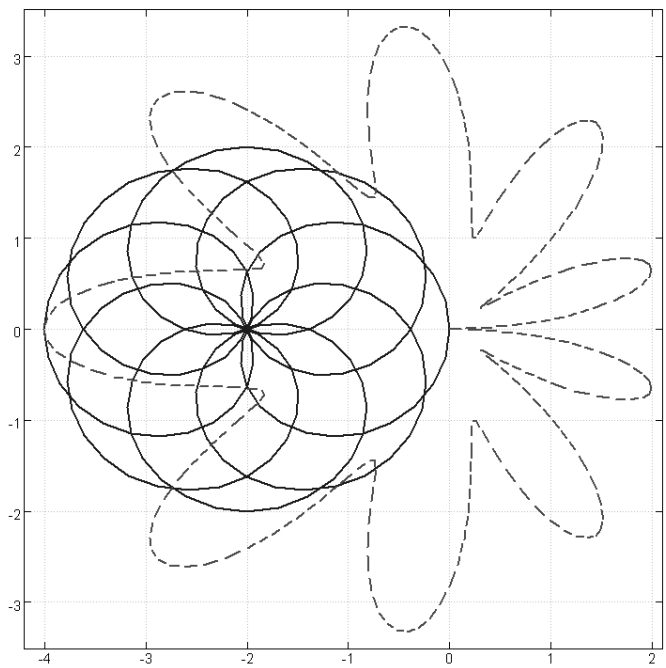


Figure 4. Cauchy curve and magnitude for $f(z) = 1+9z^8$.

Rational functions

With rational functions we need to be careful. Here we consider the rational function given by the following.

$$f(z) = \frac{1+2z^6}{1.01+z^{12}}$$

Notice that the roots of the denominator in the complex plane will be slightly larger than one in magnitude (about 1.00083), so the unit circle gets near, but does not reach or encompass the bad points.

```
f=:1 0 0 0 0 0 2&p.%1.01+^&12
lC=:r steps 0 2p1 30000
cplot f chycu lC
```

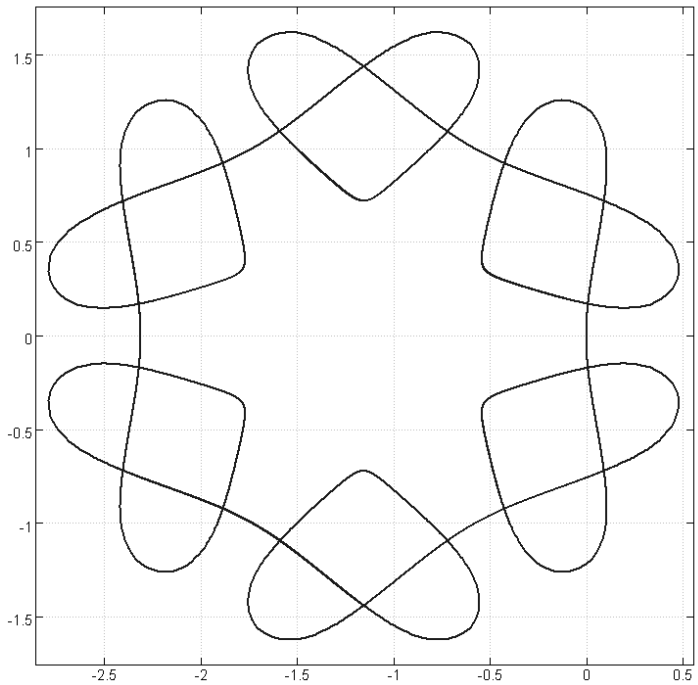


Figure 5. Cauchy curve for a rational function.

That path of integration is so near to bad points that we increased the number of steps on the unit circle to 30000 to avoid a rough appearance. This creates a curve with six heart shapes has a cute, simple appearance. This curve is very sensitive to parameters. What happens if you change the exponent 12 to 11 or 13? What happens if you vary the coefficient of the sixth power away from 2? These are worthwhile experiments for the reader to try. We offer a gallery of these variants and more at [6].

Families of Cauchy curves

Next we consider families of Cauchy curves. The adverb `cfplot` takes a dyad `u` as its verb argument and produces the Cauchy curves for the functions `a&u` for each item `a` of `x`. Figure 6 shows the Cauchy curves (all circles here) arising from z^n for $2 \le n \le 12$.

```
   require '~addons/media/image3/prevare.ijs'

   cfplot=: 1 : 0
:
opts=. 'pensize 2;aspect 1;itemcolor '
opts=. opts,":,Hue steps 0 2r3 ,#x
opts plot x u chycu"_1 _ y
)

   (2+i.11) (]^[) cfplot C
```

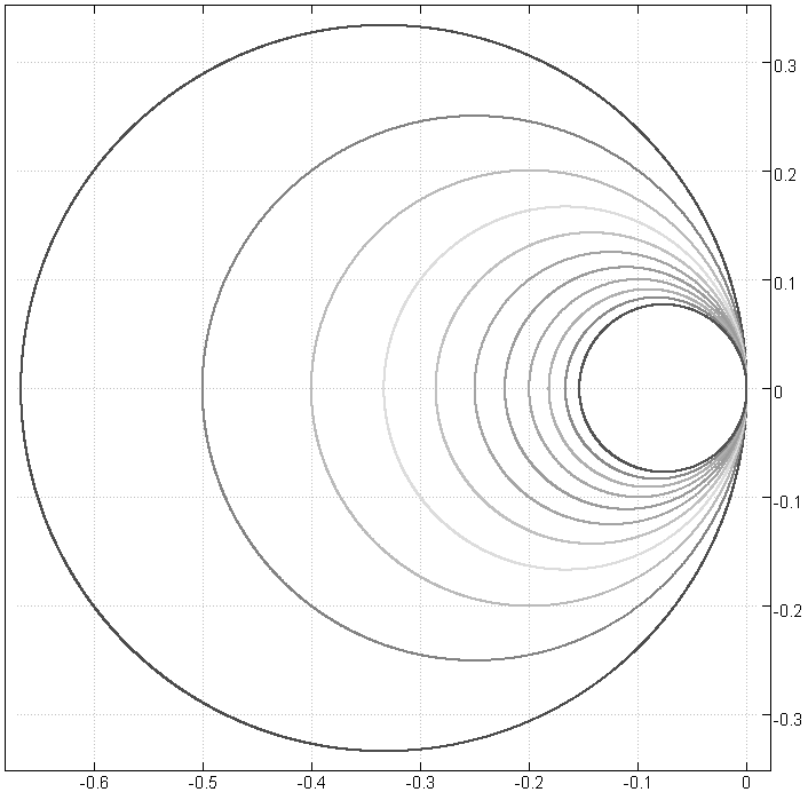


Figure 6. Cauchy curves from $f(z) = z^n$.

Next we consider plots of

$$f(z) = \frac{1+2z^6}{1.01+z^a}$$

for exponents a geometrically distributed near 12.

```
   fh6=: 1 0 0 0 0 0 2&p.@] % 1.01+]^[
   lC=: r steps 0 2p1 30000

   ({.,{:)a=: 12*1.07^ steps 0 0.36 100
12 12.2959

   a fh6 cfplot lC
```

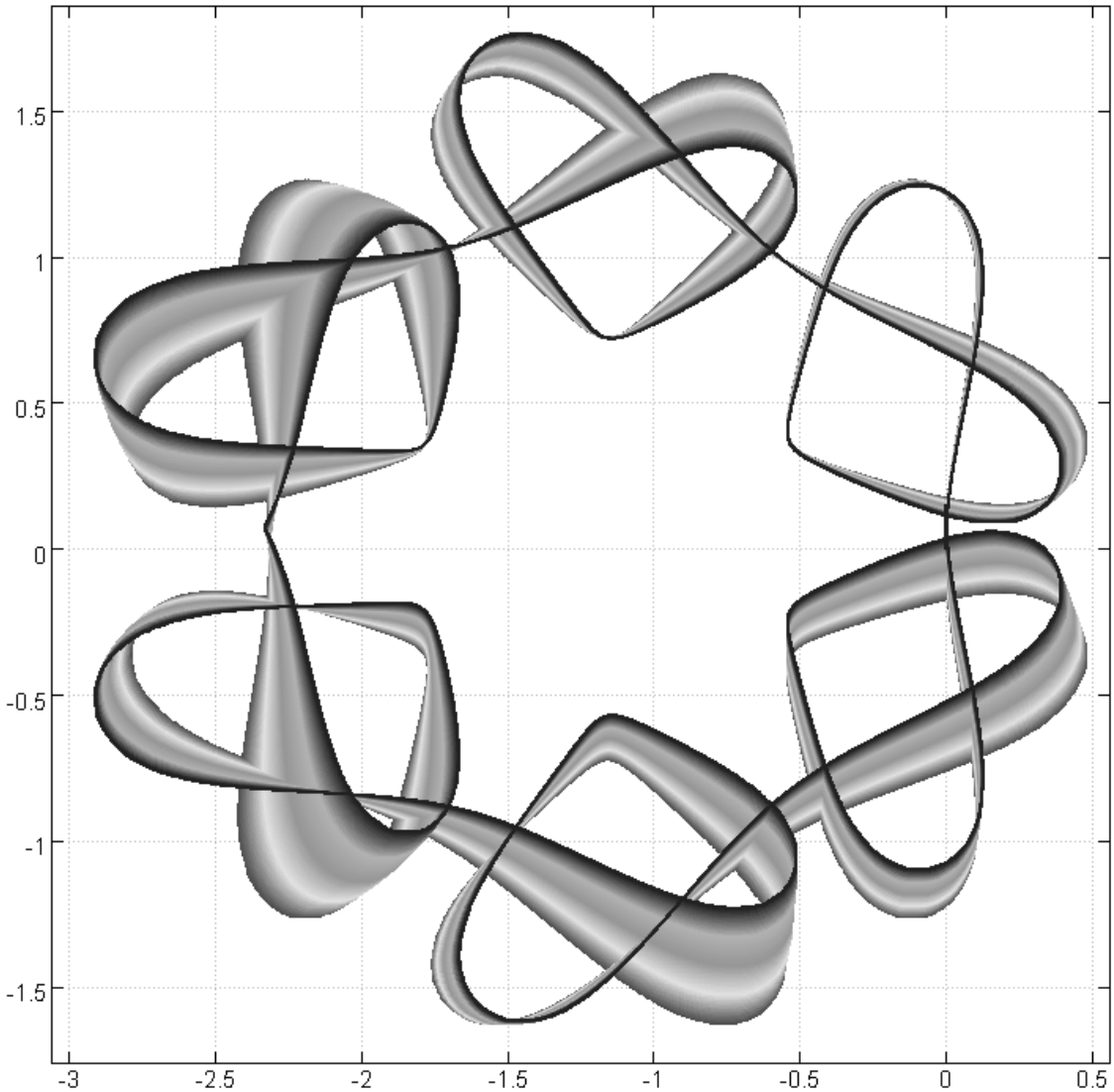


Figure 7. Cauchy curves from `fh6`.

Next we look at the Cauchy curves from the family of functions $f(z) = \cos(a\ z^2)$. The result is shown in Figure 8.

```
   ({.,{:) a=: 4*0.9^steps 0 1 50
4 3.6

   a cos@:([**:@]) cfplot C
```

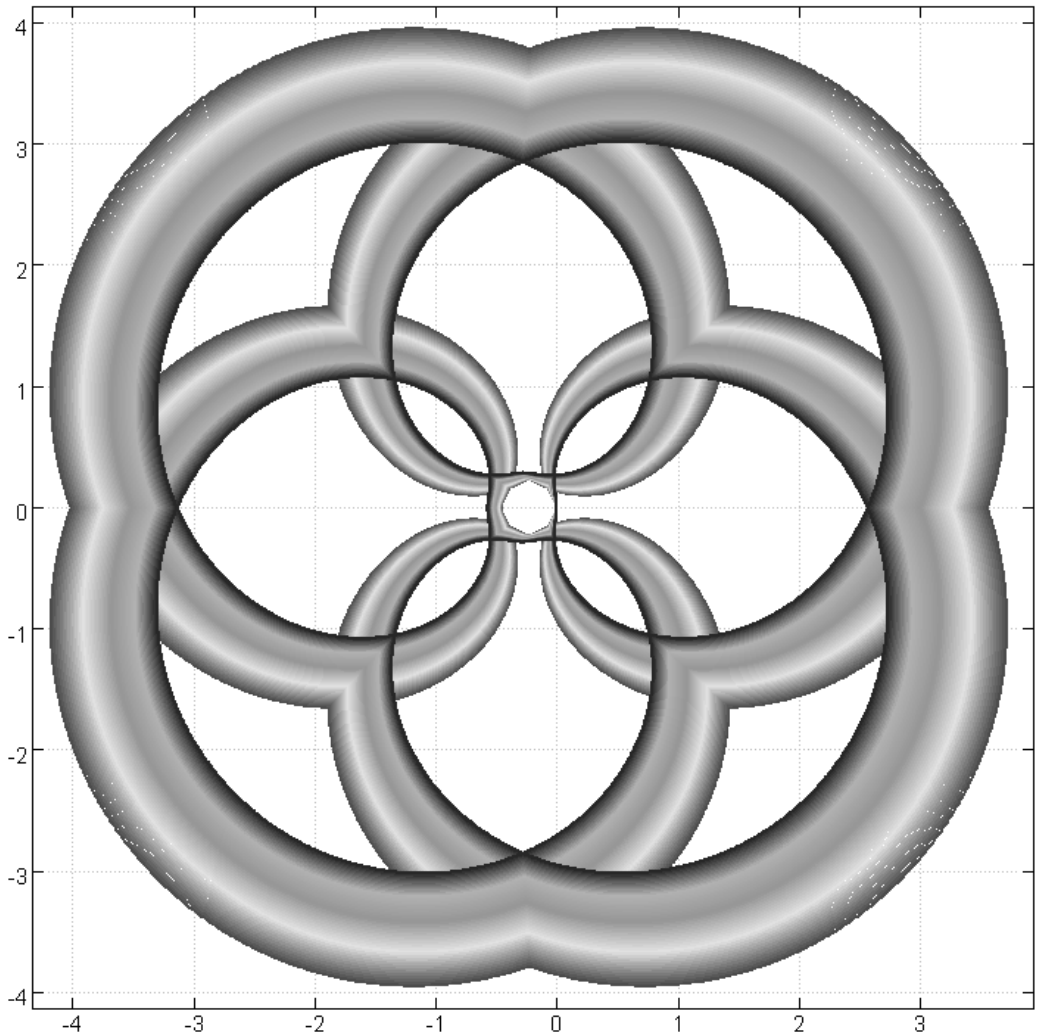


Figure 8. Cauchy curves from f(z) = cos(a z²).

Several other plots of families of Cauchy curves may be found in the gallery [6]

Alternate closed paths of integration

In this last section we note that we need not restrict ourselves to using the unit circle `C` as our closed path for the integration. We consider below examples using a square and an ellipse. The square has vertices at `_1j_1`, `1j_1`, `1j1` and `_1j1`.

```
pp=: 2 : '(m&*@-.+n&*)@(1&|)'

ss=: _1j_1 pp  1j_1

ee=: 1j_1 pp  1j1

nn=: 1j1  pp _1j1

ww=: _1j1  pp _1j_1
```

```
rsq=: ss`ee`nn`ww@.(4|<.)"0

sq=: rsq steps 0 4 50000

(2+i.11) (]^[) cfplot sq
```

Figure 9 gives the Cauchy curves of $f(z) = z^n$ with a square as the closed path of integration. Compare to Figure 6.

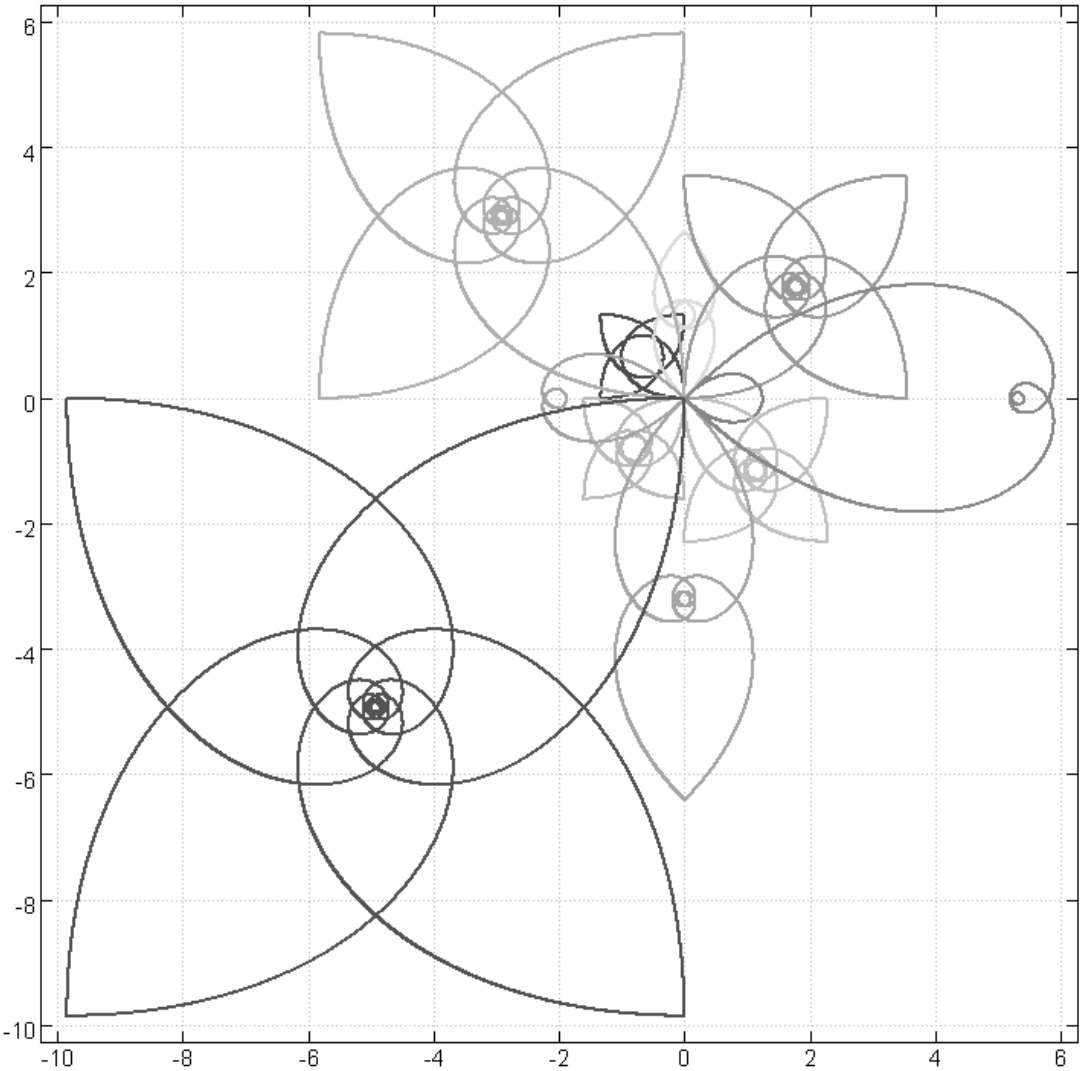


Figure 9. Cauchy curves from $f(z) = z^n$ on a square.

In Figure 10 we show the result of using a very mild ellipse (the ratio of the axes is 0.9) as the path of integration for the function `fh6`. Compare to Figure 7. Notice the hearts are gone. Cauchy curves are very sensitive to the path of integration.

```
   $ell=: (cos j. 0.9 * sin) steps 0 2p1 30000
30001

   (12*1.07^ steps 0 0.5 100) fh6 cfplot ell
```

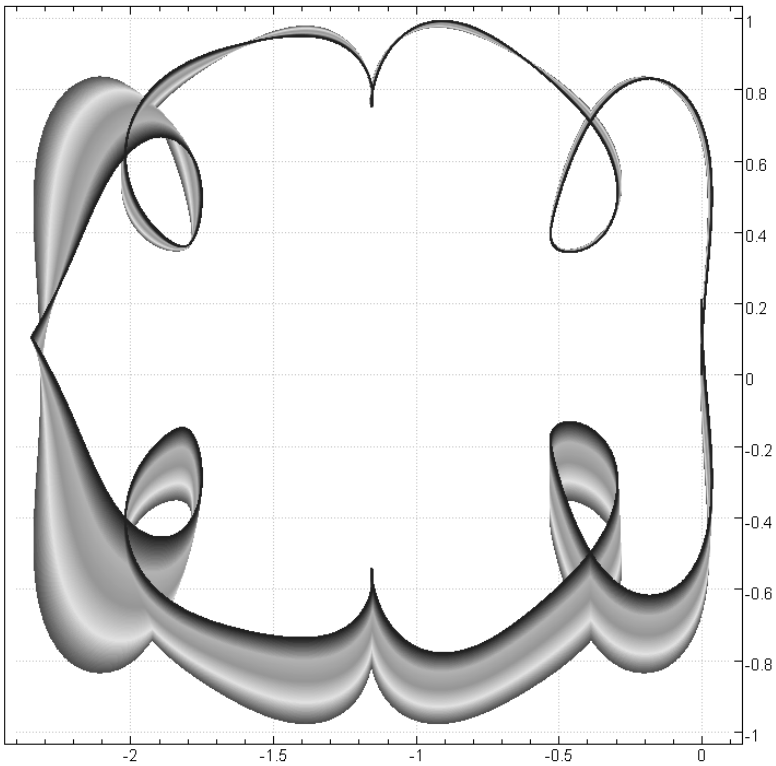


Figure 10. Cauchy curves from `fh6` *on an ellipse..*

References

1. E. T. Bell, *Men of Mathematics*, Simon and Schuster, 1965.
2. David M Burton, *The History of Mathematics*, 6th ed., McGraw Hill, 2007.
3. John B. Conway, *Functions of one complex variable*, 2nd ed. Springer-Verlag, c1978 (1995 printing).
4. Ken Iverson, "A Dictionary of J", *Vector* 7.2 (1990) 99-117.
5. "Complex Operations", www.jsoftware.com/jwiki/Essays/Complex_Operations
6. William R. Jones and Cliff Reiter, *Cauchy Curves* Auxiliary Gallery, www.lafayette.edu/~reiterc/mvq/cc/index.html
7. Tristan Needham, *Visual complex analysis*, Oxford University Press, 1997
8. Cliff Reiter, *Fractals, Visualization and J*, 3rd ed., Lulu.com, 2007
9. Frank Smithies, *Cauchy and the creation of complex function theory*, Cambridge University Press, 1997
10. Norman Thomson, "J-ottings 50: If you think J is complex try j", *Vector* 23.4 (2008) 106-118, www.vector.org.uk/archive/v234/jot50.htm

About polynomials

Part 2

by Gianluigi Quario
giangiquario@yahoo.it

This is the second and concluding part of an article dedicated to polynomials. Evaluation of polynomials. Construction of functions for evaluation. Zero finding of a polynomial. A stable and accurate function for finding complex zeroes in polynomials of higher degree.

Introduction

The classical problem of solving an Nth-degree polynomial equation

$$c[N]\times Y^{N} + c[N-1]\times Y^{(N-1)} + \ldots + c[1]\times Y^{1} + c[0] = 0$$

has substantially influenced the development of mathematics throughout the centuries and still has several important applications to the theory and practice of present-day computing. The name *Fundamental Theorem of Algebra* itself evidences that.

The theorem was demonstrated by Gauss in 1799 and says that every polynomial equation of degree N has exactly N solutions in the complex field. The early mathematicians tried to express the solutions by means of general formulas including the coefficients of the polynomial and using just arithmetical operations and radicals. The cases of degree 3 and 4 were solved by Italian thinkers during 16th century. Then for more than two centuries mathematicians tried in vain to find analogous formulas for degree 5.

At the beginning of 19th century Ruffini & Abel demonstrated that it is not possible to have such general formulas from degree 5 up.

The following interest in the solution of polynomial equations was concerned with iterative methods to obtain approximate numerical values.

Many algorithms were proposed and realized. Wolfram Mathworld [1] enumerates the following:

Bailey's Method, Bairstow's Method, Bernoulli's Method, Bisection, Brent's Method, Crout's Method, False Position Method, Graeffe's Method, Halley's Irrational Formula, Halley's Method, Halley's Rational Formula, Horner's Method, Householder's Method, Hutton's Method, Inverse Quadratic Interpolation, Jenkins-Traub Method, Laguerre's Method, Lambert's Method, Lehmer-Schur Method, Lin's Method, Maehly's Procedure, Muller's Method, Newton's Method, Ridders' Method, Schröder's Method, Secant Method, Tangent Hyperbolas Method

There are state-of-art root-finding packages available using multiprecision, such as MPSolve implemented by Bini *et al.* [2] and Eigensolve by Fortune [3].

Nevertheless this subject of research is still a current topic.

Certainly one reason is that none of these algorithms can give adequate results in all realistic circumstances. The inevitable rounding errors generated during the calculation sometimes obstruct the convergence of the iterations, even when using extended-precision arithmetic.

The challenge for the present-day research is to develop algorithms of numerical solutions guaranteed and with reasonable computing time.

I'll present a Dyalog APL implementation of Aberth's method, following a Fortran 77 program written by D. Bini [4].

I wish to thank Prof. D.A. Bini sincerely for his agreement to disclose current work.

Complex numbers

When dealing with polynomials it is convenient to adopt complex numbers.

Complex number c corresponds to a point in a 2-dimensional plane. It can be represented with a pair of real coordinates (a, b) with the orthogonal real and imaginary axes forming a basis.

As this work is based upon Dyalog APL, that does not support computation with complex numbers, we will define a complex number to be a simple 2-element numeric vector. Each complex number is considered to be a scalar. For example:

```
c ← ⊂1 1
```

is a complex scalar. And

```
c← (1 1)(0 2)(1 0)
```

is a 3-element complex vector.

Evaluation of real and complex polynomials

The following function returns the real values of a real polynomial using the Ruffini-Horner method:

```
ZrPoly←{
    ⍝ values at ⍵ of real polynomial with coefficients ⍺
    ⍝ e.g. : 1 3 3 1 ZrPoly ⍳9
     ⎕IO←0 ⋄ coe pts←⍺ ⍵
     ruffini←{⍺+⍺⍺×⍵}
     ⊃pts ruffini/coe
}
```

And this returns the complex values of a real or complex polynomial:

```
ZcPoly←{
    ⍝ values at ⍵ of complex polynomial with coefficients ⍺
    ⍝ e.g. : 1 3 3 1 ZcPoly ⍳9
    ⍝      : (1 0)(3 0)(3 0)(1 0) ZcPoly (1 0)(2 0)(3 0)(1 ¯1)(2 2)
     ⎕IO←0 ⋄ coe pts←Zr2c¨⍺ ⍵
     ⍝         ⍺  + ⍺⍺       ×         enclose if simple ⍵
     ruffini←{(⊂⍺)+¨⍺⍺{(⍺-.×⍵),⍺+.×⌽⍵}¨(,∘⊂∘,⍨(1=≡,⍵))⍵}
     ⊃pts ruffini/coe
}
```

where `Zr2c` transforms a real into a complex number

```
Zr2c←{
    ⍝ array of complex from simple array of real numbers
    ⍝ ⍵ is a simple array (or scalar) of real numbers
     2=|≡⍵:⍵  ⍝already complex (the depth of a complex is 2)
     ⍵,¨0
}
```

Construction of functions for evaluation of polynomials

Let `poly` be a numerical vector of coefficients of polynomial p(Y),

$$p(Y) \longleftrightarrow poly[N]\times Y^{N} + poly[N-1]\times Y^{(N-1)} + \ldots + poly[1]\times Y^{1} + poly[0]$$

In the first part we have seen that the algebrists tend to study the polynomial functions as if they were not functions but generic objects. We continue to represent polynomials by their coefficient vector `poly` (ascending order). Vector `poly` is either a real or a complex vector.

Given `poly` it is natural to define the associated functions for its evaluation `poly∘ZrPoly` or `poly∘ZcPoly`.

When `poly` is real it is possible to define a direct function using only the primitives `+` and `×`. Let us start with a trinomial and consider that it can be evaluated by means of Ruffini-Horner method:

```
poly[0] + Y × (poly[1] + Y × poly[2])
```

The expression `poly[1] + Y × poly[2]` gives us the function

```
{(poly[1] ∘+) ∘ (poly[2] ∘× ) ⍵}
```

and the expression `Y × (poly[1] + Y × poly[2])`

```
{poly[0] ∘+ ⍵} × {(poly[1] ∘+) ∘ (poly[2] ∘× ) ⍵}
```

This is a monadic fork of three functions `{f ⍵} × {g ⍵}` and is equivalent to

```
×∘+∘(poly[1]∘+)∘(×∘+∘(poly[2]∘×)⍨
```

Now the final evaluation function of the trinomial is

```
PolyEval←(poly[0]∘+)∘(×∘+∘(poly[1]∘+)∘(×∘+∘(poly[2]∘×)⍨)
```

In a recursive way we can obtain for a polynomial of degree 3

```
PolyEval←(poly[0]∘+)∘(×∘+∘(poly[1]∘+)∘(×∘+∘(poly[2]∘+)∘(poly[3]∘×)⍨)⍨)
```

and similar direct functions for higher degree.

Zero finding of a polynomial

From a functional point of view the zero-finding of polynomial `poly` is a straightforward problem. We have the function for polynomial evaluation and the *inverse* operator `⍣¯1` [5].

Zero finding should be reached in this way:

```
(poly∘ZcPoly⍣¯1) 0
```

or

```
(PolyEval⍣¯1) 0
```

provided that the inverse operator is applied to a primitive or an expression of primitive functions combined with primitive operators. We can follow only the second way.

Let `poly` be the trinomial `2 3 1`. Then we have

```
      PolyEval←(2∘+)∘(×∘+∘(3∘+)∘(×∘+∘(1∘×)⍨))
```

and

```
      (PolyEval⍨*¯1) 0
¯1
```

This is a genuine root of the trinomial.

I built this kind of function for a degree-99 polynomial. (It's a very long string: you can find it in [6].)

```
      poly ← 100⍴⌽⍳10
```

and the returned result was

```
¯1.338591185893
```

I cannot imagine the hard work done by the interpreter, but I feel admiration for John Scholes and its implementation. This is also a genuine root of the polynomial… even if the result has forgotten many companions.

Unfortunately the inverse operator is not so cute as to perceive that the function `PolyEval` is not invertible: the inverse operator is able to tell the truth but not the whole truth. We cannot follow the functional route.

A stable and accurate function for finding complex zeroes in polynomials of higher degree

J has the primitive monadic function `p.` [7] which finds roots of a polynomial by means of a internal iterative algorithm. I think that also the APL programmer needs such a tool.

Dyalog APL lacks extended-precision numbers and accordingly it is more difficult to obtain stability (avoid that errors introduced at one time step cannot grow unboundedly at later times) even with methods equipped with the property of convergence.

I found a Fortran program [3] implementing the Aberth method in standard floating-point arithmetic.

The Aberth method is a root-finding algorithm for simultaneous approximation of all the complex roots of a univariate real or complex polynomial. It derives from Newton's method, but is less susceptible to a failure of convergence.

This algorithm has the advantage of an upper limit to the number of iterations and that, besides the approximated roots, the output contains the corresponding error bounds.

An excerpt from its abstract:

```
X*************************************************************************
X*    NUMERICAL COMPUTATION OF THE ROOTS OF A POLYNOMIAL HAVING          *
X*        COMPLEX COEFFICIENTS, BASED ON ABERTH'S METHOD.                *
X*                     Version 1.4,  June   1996                         *
X*    (D. Bini, Dipartimento di Matematica, Universita' di Pisa)         *
X*                        (bini@dm.unipi.it)                             *
X*************************************************************************
An algorithm for computing polynomial zeros, based on Aberth's method, is
presented. The starting approximations are chosen by means of a suitable
application of Rouché's theorem. More precisely, an integer q  >=1 and a set
of annuli A i for i=1,...,q, in the complex plane, are determined together with
the number k i of zeros of the polynomial contained in each annulus A i. As
starting approximations we choose k i complex numbers lying on a suitable circle
contained in the annulus A i for i=1,...,q. The computation of Newton's
correction is performed in such a way that overflow situations are removed. A
suitable stop condition, based on a rigorous backward rounding error analysis,
guarantees that the computed approximations are the exact zeros of a "nearby"
polynomial. This implies the backward stability of our algorithm. We provide a
Fortran 77 implementation of the algorithm which is robust against overflow and
allows us to deal with polynomials of any degree, not necessarily monic, whose
zeros and coefficients are representable as floating point numbers. In all the
tests performed with more than 1000 polynomials having degrees from 10 up to
25,600 and randomly generated coefficients, the Fortran 77 implementation of our
algorithm computed approximations to all the zeros within the relative precision
allowed by the classical conditioning theorems with 11.1 average iterations. In
the worst case the number of iterations needed has been at most 17.
```

I translated it in a dynamic function (Dyalog APL v.11), taking some liberties with respect to iterations/recursions and the calculation of starting values. The function is named `p_` to evoke J's primitive `p.`

```
⍝#######################################################################
⍝              //////////  A P L  documentation  \\\\\\\\\\\
⍝#######################################################################
⍝

⍝   The right_argument of this APL function is a vector that contains
⍝       the real or complex coefficients of a polynomial.

⍝   The left_argument α:
⍝       when α is numeric, value of polynomial is calculated at α.
⍝       when α is NOT numeric:
⍝         * case α is missing or 0-length vector: default case

⍝           The function returns a 2 elements vector:
⍝         [1]  hiCoe          scalar    is the highest_degree coefficient;
⍝         [2]  valueRoots     vector    of the computed approximations to
⍝              roots
⍝         * case α='x' :  extended results

⍝         The function returns a 7 elements vector:
⍝         [1]  hiCoe          scalar    is the highest_degree coefficient;
⍝         [2]  valueRoots     vector    of the computed approximations to
⍝              roots

⍝         [3]  boolRoots      vector    is 1 for successful approx. of
⍝              the i-th root.
⍝              More specifically, the disk of center valueRoots[i] and
⍝              radius radiusRoots[i] ,in the complex plane, contains a root
⍝              of p(x)  for i=1,...,n.
⍝              It contains informations about the computed approximations:
⍝              = 1 if the approximation of the i-th root has  been
⍝              carried out successfully, i.e., the computed approximation
⍝              can be viewed as the exact root of a slightly perturbed
⍝              polynomial.
⍝              = 0  if more iterations are needed for the i-th  root
⍝              or if the corresponding root cannot be represented as
⍝              floating point due to overflow or underflow

⍝         [4]  radiusRoots    vector    is the absolute error bound
⍝              If there exist roots which cannot be represented without
⍝              overflow or underflow, then the corresponding components
⍝              of RadiusRoots are set to -1

⍝         [5]  radiusRoots÷Modulus(valueRoots)
⍝                             vector    is the relative error bound
```

```
⍝         [6]  moduRoots      vector    moduli of computed approximations

⍝         [7]  iterNo         scalar    is the number of iterations
⍝              needed by the algorithm.

⍝         [8]  msgs           vector    segmented string of warning msgs
⍝                                       (maybe empty)
⍝         * case ⍺='i' :  iteration runtime values
⍝         The function returns a 8 elements vector of case 'x' and:

⍝         [9]  valueInit      vector    2 elements:
⍝                                       - vector of complex init guesses
⍝                                       - bool vector of warnings over guesses

⍝         [10] valueIter      matrix    rows of guesses at each step
⍝         * case ⍺≡'rootcenter' :  the function
⍝         returns the root-center(centroid)of the (not computed) roots
⍝         * else :  default case

⍝   The right_argument of this APL function can be a 2 elements nested vector
⍝    that contains
⍝         [1]  hiCoe          scalar    is the highest_degree coefficient;
⍝         [2]  valueRoots     vector    the N roots of a N-degree polynomial
⍝         hiCoe and valueRoots can be real or complex
⍝    Then the function returns a (N+1)-elements vector  that contains
⍝       the complex coefficients of the polynomial.
```

Note that all the roots are approximated roots and that they are never rounded. The roots are always returned as complex numbers. Sometimes a real root is represented as having a imaginary part, but that is due to approximation of the real root over the complex plane.

For example the real roots (-1 and -2) of real polynomial

```
      p_ 2 3 1     ⍝ short result
```

are returned as

```
(1 0)  (( ¯1 ¯1.058791184E¯22)  (¯2 1.009741959E¯28))
```

where `(1 0)` is the highest degree coefficient.

If we write

```
      'x' p_ 2 3 1 ⍝ extended result
```

we get

```
(1 0) ((¯1 ¯1.058791184E¯22)(¯2 1.009741959E¯28)) (1 1)
… (7.68274333E¯15 1.110223025E¯14)
… (7.68274333E¯15 5.551115123E¯15)  (1 2) 5 ('')
```

where we can see that the distance of the computed roots from the actual roots is less than

```
(7.68274333E¯15 1.110223025E¯14)
```

and that 5 iterations were made.

The function `p_` is tailored in order to solve polynomials of degree until 500 and 1Mb of available workspace is more than sufficient. It is possible to raise the maximum-allowed degree: you can change the value assigned to local variable `MAXDEG` inside `p_`. I made some rewarding tests with degree 2000.

`p_` is self-supported and contains some local functions for complex arithmetics.

When the argument of function `p_` is a 2-element nested vector, where the first element is a scalar and the second is a simple vector, the polynomial is returned whose roots are the elements of second vector.

For example `p_ 1(¯1 ¯2)` is the polynomial `2 3 1`

The function `p_` and its companion `p_Display` are stored in file `PolyRoot.DWS` [6].

```
      p_Display 'x' p_ ¯1 9 8 7 6 5 4 3 2 1      ⍝ formatted display of extended result
1) valid=Y   8.75708424346 6E¯1 ¯8.910997943917E¯1  R= 2.E¯14 RR= 2.E¯14 modulus= 1.25E0
2) valid=Y   8.75708424346 6E¯1  8.910997943917E¯1  R= 4.E¯14 RR= 3.E¯14 modulus= 1.25E0
3) valid=Y   1.277725973022E¯1  1.319267183775E0   R= 4.E¯14 RR= 3.E¯14 modulus= 1.33E0
4) valid=Y   1.277725973022E¯1 ¯1.319267183775E0   R= 4.E¯14 RR= 3.E¯14 modulus= 1.33E0
5) valid=Y   1.011379823900E¯1  9.573914554732E¯19 R= 7.E¯16 RR= 7.E¯15 modulus= 1.01E¯1
6) valid=Y  ¯7.418904085081E¯1 ¯1.149403115215E0   R= 6.E¯14 RR= 4.E¯14 modulus= 1.37E0
7) valid=Y  ¯7.418904085081E¯1  1.149403115215E0   R= 6.E¯14 RR= 4.E¯14 modulus= 1.37E0
8) valid=Y  ¯1.312159604336E0   4.525676361664E¯1  R= 7.E¯14 RR= 5.E¯14 modulus= 1.39E0
9) valid=Y  ¯1.312159604336E0  ¯4.525676361664E¯1  R= 9.E¯14 RR= 7.E¯14 modulus= 1.39E0

  Centroid= ¯2.222222222222E¯1  4.440892098501E¯16
```

The workspace `PolyRoot.DWS` contains the variable `Describe` with some test polynomials.

References

1. mathworld.wolfram.com/PolynomialRoots.html
2. D.A. Bini and G. Fiorentino, *MPSolve: Numerical computation of polynomial roots* v. 2.0, FRISCO report (1998)
3. S. Fortune, "An iterated eigenvalue algorithm for approximating roots of univariate polynomials", *JSC* (2002)
4. D.A. Bini "Numerical computation of polynomial zeros by means of Aberth's method", *Numerical Algorithms*, 13 (1996)
5. www.dyalog.com/help
6. `PolyRoot.dws` at tech.groups.yahoo.com/group/dyalogusers/files/Software
7. www.jsoftware.com/jwiki

Quick calculation of Kendall's Rank Correlation Distribution

by Gordon Sutcliffe
gsutcliffe@talk21.com

The nub of the paper is the `km2` function with `krfd` as a user-friendly cover function for quick calculation of Kendall's rank correlation frequency distribution. The remainder is intended to provide a useful, but by no means exhaustive, background.

Introduction

Kendall's Rank Correlation Distribution provides the statistical probability for the assertion that two sets of n ranks are significantly different, rather than just random fluctuations. The cumulative probability used for their solution can be produced from the integer frequencies calculated here.

Currently the probabilities for n>10 ranks are managed by using approximations to the normal distribution. However, it is not easy to determine the degree of approximation.

The problem from the calculation point of view is that Kendall's original method takes an enormous amount of computer time to calculate when the number of ranks is more than 12. On a PC [1] the frequency distribution for n=13 ranks would take days, with the time increasing as factorial n.

Quick calculation is based on Kendall's theory, but uses a recurrence relationship which enables the frequency distribution to be calculated using floating point arithmetic in a fraction of a second up to n=170. (Factorial 170 is the limit for standard floating point arithmetic on a PC.)

The integer floating-point frequencies are exact up to n=18, but at higher n, precision drops to about 13 to 14 significant digits [2]. J extended-integer arithmetic produces exact frequencies up to n=170 in 2000 seconds, or up to n=300 in 14 hours on a PC.

Methods of calculation

Two calculation methods are considered. Method 1 is based on first principles, as in Kendall's work. Runtime usually limits calculation to about n=12 ranks at most. The second method is based on a recurrence relation. It is practical up to n=170 ranks using floating point arithmetic and can provide exact frequencies to over n=300 using J extended-integer arithmetic.

Both methods agree with *Biometrika* Table 45, [3] up to its limit of 10 ranks. Basic properties of the Kendall's rank correlation frequency distribution against Q for n>0 are:

Distribution symmetry about:	(max. Q)/2	Q=0 frequencies:	1
Sum of the frequencies:	!n	Q=1 frequencies:	n-1
Number of frequencies, Q:	1+n×(n-1)/2	Q=2 frequencies:	(-1)+n×(n-1)/2 [for n>1]

Q and n are discussed further under Method 1.

Method 1: First principles

Taking the case for n=3, the 3 ranks have 6 = 3 × 2 × 1 permutations. Assuming that a rank order `a b c` corresponds to perfect correlation, the (minimum) number of neighbour interchanges required to transform each permutation to `a b c` is ascertained:

Permutations for n=3	a b c	a c b	b a c	b c a	c a b	c b a
Required neighbour interchanges, Q	0	1	1	2	2	3
Frequency total for each unique Q value above	1	—	2	—	2	1

The frequencies for each value of Q = 0, 1, 2, 3 are repeated in the table below in the row for n=3.

Table of Frequency Distributions for n=1 to 6 Ranks

The italic figures are temporary additions to aid explanation of the recurrence method

NO. OF RANKS	NUMBER OF NEIGHBOUR INTERCHANGES, Q FOR THE FREQUENCIES IN THE COLUMN BELOW THEM																SUM OF FREQS.	COUNT OF QS
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	!n	1+n×(n-1)/2
0	1	*(-1)*															1	1
1	1		*(-1)*														1	1
2	1	1		*-1*	*(-1)*												2	2
3	1	2	2	1	*-1*	*-2*	*-2*	*(-1)*									6	4
4	1	3	5	6	5	3 *-1*	1 *-3*	*-5*	*-6*	*-5*	*-3*	*(-1)*					24	7
5	1	4	9	15	20	22	20 *-1*	15 *-4*	9 *-9*	4 *-15*	1 *-20*	*-22*	*-20*	*-15*	*-9*	*-4*	120 *(-1)*	11
6	1	5	14	29	49	71	90	101	101	90	71	49	29	14	5	1	720	16

Method 2: Recurrence relation

Row r+1 is based [4] on the previous row r. Rows are calculated from row n=1 initially. To calculate the frequencies for the next row, first negate a copy of the current row and insert as shown by the italics at the bottom of the row. The row is then summed progressively from column 0 to the right to produce the next row, stopping before the *(_1)* column, which would otherwise produce an unwanted zero.

Notes and references

1. PC system specification s/w: Windows XP home ed. 5.1.2600, Service pack 2; J v6.01c; H/W: Processor: x86 Family 15 Model 2 Stepping 9 GenuineIntel ~2605 MHz; RAM: 512 MB.

2. More precisely the proportional error is in the range: (1 - 6e-15) to (1 + 4.5e-15).

3. E.S. Pearson and H.O. Hartley, ed., *Biometrika Tables for Statisticians* Volume 1, 2nd Edition, 1958, Table 45, 84-5, 85-6, 88-9

4. M.G. Kendall & A. Stewart, *The Advanced Theory of Statistics*, Volume 2, 3rd Edition, Inference and Relationship, Robust and Distribution-Free Procedures, Ch. 31.25 Eqn (31.34)

Appendix: J code

Summary

Verbs grouped according to their main application: in the last column, `n` is the number of ranks, `Q` the number of interchanges.

KENDALL'S RANK CORRELATION DISTRIBUTION

`km1`	Original method	`km1 n`
`km2`	Recurrence method	`km2^:(<:n) 1`
`krfd`	Cover function for km2, Frequency Distribution	`krfd n`
	Kendall's Rank Probability Distribution	`( % +/) krfd n`
	Kendall's Rank Cumulative Probabilities; used for Biometrika [3] check	`(] +/\@:% +/) krfd n`

PROBABILITY OF RANDOM VARIATION IN TWO SETS OF RANKS

`r2q`	Ranks to Q values	`A_ranks r2q B_ranks`
`q2p`	Q values to Probabilities	`n q2p Q`
`q2t`	Q values to Kendall's tau	`n q2t Q`
	Single- and double-tail comparison of rank sets	see Listing

MISCELLANEOUS

`dsp`	Display of results and inline debugging aid	`… dsp right_arg …`
`ktcv`	Kendall's tau Critical Values	`n ktcv nominal_probability`

Listing

```
dsp =: 1!:2&2  NB. Utility monad: display and return argument
n =. 3         NB. n indicates the number of ranks

NB. === Kendall's Rank Correlation Distribution ====================

  NB. Both km1 (Original method) and km2 (Recurrence method) create
  NB. Kendall's Rank Correlation Frequency Distribution for n ranks.

km1 =: [: +/"1 [: = [: +/"1 (i.@! ({. +/@:> ])\."1@:A. i.)
  NB. Monad Syntax:  km1  n   Note: n>10 may fail on memory limit

km2 =: 3 : 0  NB. Monad SYNTAX:  km2^:(<:n) y=.1  Note: n>0 ; y=1
        NB. n-1 recurrences update the starting value of y
 n =. +/ _1 x: 2{. y              NB. l is the no. of Q values for
 h =. >.-: l=. >: -: ( * >:) n    NB.  the new distribution (n+1)
 ( , |.`(|.@}:)@.(2|l)) +/\  (h&{. - [: (-h)&{. (0>.h-n+1)&{.) y
)

krfd =: 3 : 0  NB. Syntax: [initial_distrbn]  krfd  iterations_reqd.
 1 krfd y      NB. Monad default x=1 sets floating point arithmetic.
               NB. x can be a previous result and/or have an 'x'
               NB.  suffix for extended integer arithmetic
:
 km2^:(y-1=#x) x  NB. Reduce effective y by 1 when x=1 (or 1x)
)

(] % +/) krfd n      NB. Probability distribution
( +/\@:% +/) krfd n  NB. Cumulative probability
                     NB. For Biometrika Table 45 Ref [1] check

NB. === Probability of Random Variations of two sets of ranks ======

r2q =: 3 : 0                     NB. Ranks to Q
NB. SYNTAX:  [A_ranks]  r2q  B_ranks   Monad assumes y pre-ordered
 if.  (] -&# ~.) y do. 'r2q: y ranks indiscrete' return. end.
 Q =. +/ ({. +/@:> ])\. y         NB. Exit with Q value
:
 if.  (] -&# ~.) x do. 'r2q: x ranks indiscrete' return. end.
 r2q (/:x) { y                    NB. Reorder y as ascending x
)
```

```
  NB. Convert Q to probability in single- or double-tail test
q2p =: 4 : 0  NB. SYNTAX:  no._of_ranks  qp2  Q_value
 if. -.y e. i.>: l=.-:(* <:) x do. 'q2p: incompat. args' return.
end.
 st =. (!x)%~ +/ (y+1){. d=:krfd x  NB. Single-tail probability
 z =. y <. l - y                    NB. Double-tail: complement y
 dt =. 1 <. +: (!x)%~ +/ (z+1){. d  NB. Double-tail probability
 hd =. '   n       Q  Single-tail Double-tail', LF  NB. Headings
 hd, 4 8 10j6 12j6": x, y, st, dt   NB. Results
)

  NB. Convert Q to Kendall's tau
q2t =: 1: - 4: * ] % ( * <:)@[  NB. SYNTAX:  n  q2t  Q

  NB. Rank comparison to single-, double-tail probability
'r0 r1 r2' =: 0 1 2 3 4 5  ;  5 4 3 2 1 0 ; 6?.6  NB. Rank data
6 q2p r0 r2q r0    NB. Both results very significant
6 q2p r0 r2q r1    NB. Double-tail result very significant
6 q2p r0 r2q r2    NB. Neither result significant

NB. === Miscellaneous ============================================

  NB. Critical values of Kendall's tau in cumulative probability
ktcv =: 4 : 0  NB. SYNTAX:  n  ktcv  Nominal_s-tail_signif._prob.
 d =. (+/\@krfd % !) x      NB. Cumulative probability
 x q2t (d <: y) i: 1        NB. Critical value of Kendall's tau
)
```

Consultants

This directory lists companies and consultants who provide commercial consulting services in one or more of the APLs.

For vendors of interpreters, please see the Vector web site at vector.org.uk.

If your interest in the APLs is non-commercial, please consider creating a home page for yourself and your interests on the APL Wiki at aplwiki.com.

Please advise changes or additions to this directory by email to Ray Cannon: ray_cannon@compuserve.com.

United Kingdom

APL Team Ltd aplteam.com	Analysis, design and programming
Alastair Kinloch alastair.kinloch@btinternet.com	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
Andrews ADWilson@kencomp.net	APL programming and analysis, algorithms, tree processing and design programs for craft work.
Causeway Graphical Systems Ltd causeway.co.uk	On-site training for Causeway, RainPro and NewLeaf. Customisation and enhancement to meet local needs. Code review and pre-implementation check of Causeway applications.
Ellis Morgan apl@ellismorgan.co.uk	Business Forecasting & APL Systems.

First Derivative Analytics Ltd KenChakahwata@compuserve.com	Analysis, design, prototyping, development & testing of APL (especially financial) applications: Sharp, Dyalog APL.
First Derivatives plc firstderivatives.com	Financial trading software in q, K and kdb+
General Software Ltd martin@gsoft.plus.com	Over 20 years experience with every version of APL, large mainframe systems and small PC based programmes.
Graeme Robertson Ltd GraemeDR@nildram.co.uk	Design and write custom software. Maintain and upgrade APL systems. Deliver customized APL training courses.
HMW Computing HMW@4xtra.com	System design consultancy, programming. HMW specialize in banking and prototyping work. Full members of DSDM consortium and Agile Alliance.
Hoekstra Systems Ltd Bob.Hoekstra@HoekstraSystems.ltd.uk	APL consultancy, programming, etc. Also UNIX system administration
Michael Hughes Michael@Hughes.uk.com	APL consultant with 20 years experience with all versions of APL. I can create your dynamic Web sites using the full power of APL working with Microsoft IIS on Windows NT or 2000.
Optima Systems Ltd optima-systems.co.uk	APL professionals with many years experience in pharmaceutical, industrial and financial systems on both PC and Mainframe platforms.
Paul Chapman 100343.3210@compuserve.com	24-hour programmer: APL, Smalltalk, C; Windows front end design a speciality.

Phil Last Ltd phil.last@ntlworld.com	APL Consultancy, Modelling and Programming.
Ray Cannon ray_cannon@compuserve.com	APL, C, C++, Assembler, Windows, Graphics on PC and IBM Mainframe. Experience in Insurance, Chemical, and Airline Industries
Stephen Wynn centre@boltblue.com	Most experience of financial planning & mathematical areas: operational research, quality control, experimental design

Canada

APL Borealis Inc. aplborealis.com	Hands-on courses in Introductory, Intermediate, Advanced and Windows APL. Courses are customized and flexible, and may be delivered on-site, with strong emphasis on methods for efficient and maintainable APL systems development.
Godin London Incorporated godin.com	Applications in food manufacturing, travel agency, airline bookings and product lease management.
Milinta Inc. milinta.com	Design, development, maintenance, conversion, documentation in all APLs, most APs and some specific Sharp products (LOGOS, ViewPoint, Retrieve). Experience in multi-user, multi-task systems, databases, Windows.
Snake Island Research Inc. snakeisland.com	APL interpreter and compiler enhancements, intrinsic functions, performance consulting. APL parallel compiler APEX is giving very good initial performance tests with convolution faster than FORTRAN.

Denmark

Insight Systems ApS
insight.dk

We have experience with just about every APL system and platform in common use during the last 20 years, from SHARP APL under MVS or Linux to APL+Win and in particular Dyalog APL under Windows 9x, NT or 2000. If you have decisions to take about adapting your APL application to take advantage of emerging technologies, or would like your strategy reviewed, give us a call. We have extensive experience in all areas of APL development, from legacy systems, up, down and sideways migrations, to the development and support of shrink wrapped solutions based on APL. Even if we don't have time to do the work ourselves, we will know where to find someone who is an expert in your version of APL and your application area, on your continent.

Finland

ADVOCORP Oy
rett.fi

APL application conversions, APL Windows interfaces, APL to API level interfacing to any system under Windows, TCP/IP network and database connectivity, APL based financial client/server applications, Cognos Planning and ReportNet consultancy.

RE Time Tracker Oy
4ts.com

APL application conversions, APL Windows interfaces, APL to API level interfacing to any system under Windows, TCP/IP network and database connectivity.

France

Lescasse Consulting lescasse.com	A range of consultants, experts in Windows programming with APL+Win and Dyalog APL/W. More than 100 major APL applications already developed. We all have additional expertise in Formula One and Delphi.

Germany

Dittrich & Partner Consulting GmbH dpc.de	APL programming and analysis; APL workshops and training on the job
Dr Nussbaum GmbH nussbaum-gift.de	IT Consultant with a strong focus on APL.
Skelton Consulting GmbH skelton.de	K\Q Consultancy

Netherlands

Adfee adfee@concepts.nl	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
Oasis b.v. oasis.nl	Introductory courses in APL. Advanced courses for different APL versions. Expertise in APL system design, project management, conversion, migration, tuning; for all APL versions. (10+ years experience)

New Zealand

MasterWork Software Ltd fraser.jackson@xtra.co.nz	Consulting and J programming for econometrics and statistics in public policy, health and food industries.

Russia

INFOSTROY infostroy.ru	Broad experience in various APL platforms. Special skills and knowledge in developing complex applications for investment, financial and construction markets. Implementation of hybrid solutions based on APL, Delphi, C#, VBA, SQL servers.

Sweden

Evestic AB olle.evero@mail.com	Excellent track record from 15+ years of APL applications in banking, insurance, and education services. All dialects, platforms and project phases. SQL expertise.
RadSys Technologies AB randolph.schrab@radsys.se	Area of expertise: financial systems, risk analysis systems, healthcare systems

United States

APL Solutions Inc. aplsi@starpower.net	APL systems design, development, maintenance, documentation, testing and training. Providing solutions in APL since 1969.
Omega Computing Inc. alangraham@mindspring.com	APL consultancy, programming, etc.
Rex Swain rexswain.com	Independent consultant, 25 years experience. Custom software development.

Shepp & Associates LLC ashepp@compuserve.com	We do APL applications development and consulting, especially in the travel industry, especially on small computers. 25 years experience in APL programming.
The Rochester Group Inc. rochgrp.com	Specialise in MIS using Sharp APL

Subscribing to Vector

Your *Vector* subscription includes membership of the British APL Association, which is open to anyone interested in APL or related languages. The membership year runs from 1 May to 30 April.

Name _____

Address _____

_____

Postcode/Zip and country _____

Telephone number _____

Email address _____

UK private membership	£20	___
Overseas private membership	£22	___
+ airmail supplement outside Europe	£4	___
UK corporate membership	£100	___
Overseas corporate membership	£110	___
Sustaining membership	£500	___
Non-voting UK member (student/OAP/unemployed)	£10	___

Payment should be enclosed with your membership as a sterling cheque payable to *British APL Association*, or by quoting your Visa, Mastercard or American Express number:

I authorise you to debit my Visa/American Express/Mastercard (mark which)

account number _____ expiry date ___ /_____

for the membership category indicated above

_____ annually, at the prevailing rate, until further notice

_____ for one year's subscription only

Signature _____ Date _____

Send this completed form to:
BAA, c/o Nicholas Small, 12 Cambridge Road, Waterbeach, Cambridge CB25 9NJ