Gunther Cornelissen
Wilberd van der Kallen
Jeroen Sijsling
*Mathematisch Instituut*
*Universiteit Utrecht*
*Postbus 80.010*
*3508 TA Utrecht*
*g.cornelissen@uu.nl*
*w.vanderkallen@uu.nl*
*j.r.sijsling@uu.nl*

Willem de Graaf
*Dipartimento di Matematica*
*Via Sommarive 14*
*I-38050 Povo (Trento), Italië*
*degraaf@science.unitn.it*

Bart de Smit
*Mathematisch Instituut*
*Universiteit Leiden*
*Postbus 9512*
*2300 RA Leiden*
desmit@math.leidenuniv.nl

Jan Stevens
*Matematik*
*Göteborgs universitet*
*Chalmers tekniska högskola*
*SE-412 96 Göteborg, Zweden*
*stevens@chalmers.se*

Freek Wiedijk
*ICIS, Radboud Universiteit*
*P.O. Box 9010*
*6500 GL Nijmegen*
freek@cs.ru.nl

## Wiskundig werktuig

# Wiskundig onderzoek per computer?

**De computer is tegenwoordig vermoedelijk het meest gebruikte wiskundige werktuig. In de derde aflevering van deze mini-serie laat Gunther Cornelissen collega's vertellen over hun digitale hulpmiddelen.**

De rubriek *Wiskundig werktuig*, over de 'gebruiksvoorwerpen' uit het wiskundig onderzoek, gaat dit keer over het inzetten van computers, in het bijzonder over verschillende computerpakketten voor symbolisch rekenen, bewijzen en bewijsverificatie. Hieronder schrijft een aantal mensen over hun favoriete pakket en hun ervaringen daarmee — natuurlijk een selectie, er is veel meer beschikbaar.

### HOL

Bewijsassistenten zijn programma's waarmee de computer wiskunde op correctheid kan controleren. Dat wil zeggen: mits op de juiste wijze gecodeerd. Zulke gecodeerde wiskunde heet 'een formalisatie' en ziet er in de huidige systemen uit als volslagen onbegrijpelijke computercode. Er zijn vandaag de dag een viertal bewijsassistenten waarmee serieus wiskunde 'geformaliseerd' wordt: Mizar, HOL, Isabelle en Coq. Mijn onderzoeksterrein is het ontwikkelen van technologie om deze systemen *beter* te maken.

Er zijn nog niet veel hedendaagse wiskundigen die een helder beeld van bewijsassistenten hebben. Laat ik daarom eerst eentweetal misverstanden over bewijsassistenten uit de weg ruimen. Ten eerste worden bewijsassistenten *niet* gebruikt om de wiskunde verder te helpen, om nieuwe resultaten te bereiken. Je mag al blij zijn als je wiskunde die je door en door begrijpt met een heleboel werk voor een bewijsassistent verteerbaar kunt maken. Ten tweede zijn bewijsassistenten iets totaal anders dan systemen voor computer algebra en ook iets totaal anders dan automatische stellingenbewijzers. Deze drie soorten software hebben momenteel niets met elkaar te maken.

Het *aardige* van bewijsassistenten is dat als je je wiskunde eenmaal 'geformaliseerd' hebt, en het systeem zegt dat het klopt, dat je dan ook *volkomen* zeker kan zijn dat het inderdaad klopt. Over hoe zeker dat 'volkomen zeker' is kun je een hoop woorden vuil maken, maar laten we zeggen: een heel stuk zekerder dan op welke andere wijze dan ook bereikbaar is. De *enige* ruimte voor 'fouten' bij formalisatie is dat je denkt dat er wat anders staat dan er staat, dus dat je intuïtieve begrip niet overeenkomt met wat je in werkelijkheid hebt gedefinieerd en bewezen. Maar *wat* er staat is zo zeker als wat. En dat heeft iets heel moois. Een formalisatie is als een volkomen feilloze diamant.

Twee van de meest indrukwekkende formalisaties die ik ken zijn door John Harrison gedaan. Voor deze formalisaties gebruikte hij de mooiste van de vier bovengenoemde bewijsassistenten, zijn eigen versie van het HOL systeem. In zijn werk bij Intel gebruikt hij dit om floating point processors correct te bewijzen. Formalisatie van wiskunde doet hij er in zijn vrije tijd bij, voor de aardigheid.

De eerste van deze formalisaties bewijst de correctheid van de *kernel* van zijn HOL systeem. Het laat zien dat je er zeker van kunt zijn dat dit programma geen 'bugs' bevat waardoor per ongeluk incorrecte bewijzen als correct worden geaccepteerd. Nu kun je morren dat dit een kip en ei probleem is (de checker kan fout zijn en daarom de formalisatie accepteren terwijl die niet klopt), of over hoe het dan zit met Gödels tweede onvolledigheidsstelling, maar dat zijn bezwaren die geen hout snijden. Wat wel een serieus bezwaar is, is dat de formalisatie momenteel alleen het 'moeilijke' stuk van de HOL kernel behandelt (het deel dat te maken heeft met het hernoemen van variabelen bij substitutie). Het is dus niet af, in zekere zin. Als evenwel binnenkort de *hele* HOL kernel correct bewezen is, zal er een heel interessant punt bereikt zijn!

De tweede formalisatie is een vertaling van het bewijs van de priemgetalstelling uit de

analytische getaltheorie. Hierbij wordt door beschouwing van de nulpunten van de Riemann zeta-functie bewezen dat de verhouding tussen het aantal priemgetallen $\leq n$ en $n/\ln n$ in de limiet van $n$ naar oneindig gelijk is aan 1. Een paar jaar geleden gaf verzamelingstheoreticus Bob Solovay het analytische bewijs van deze stelling als uitdaging aan de bewijsassistentengemeenschap. Hij verwachtte dat het nog decennia zou duren voor de technologie zo ver zou zijn dat dit bewijs in de praktijk geformaliseerd kon worden. Toen kwam John Harrison die in één maand tijd, naast zijn gewone werk bij Intel, een bestand van 4.314 regels HOL code produceerde waarin hij een vijftal pagina's uit een boek van Donald J. Newman formaliseerde. (Die HOL code blijkt ongeveer *acht* keer zo groot als de LaTeX van Newmans bewijs: dit heet de *de Bruijn factor*.) Hij deed dit voor het *Festschrift* voor zijn promotor die dat jaar zestig was geworden. Het controleren van deze formalisatie – inclusief het checken van alle gebruikte basiswiskunde – kost de computer ongeveer twintig minuten. Gedurende die tijd construeert het HOL systeem een gerichte acyclische graaf met 22.882.354 punten, die loopt van drie beginpunten voor de drie axioma's van het HOL systeem aan de ene kant, naar een eindpunt dat correspondeert met de priemgetalstelling.

Dat John Harrison dergelijke indrukwekkende dingen met zijn HOL systeem kan doen is omdat hij één van de beste *bibliotheken* van geformaliseerde wiskunde heeft die er zijn. Het is een aantrekkelijke gedachte (een soort wiskundige versie van het 'human genome-project') om een dergelijke bibliotheek te maken voor *alle* wiskunde die je als algemeen ontwikkeld wiskundige hoort te kennen. Dat is dus ongeveer de wiskunde die vroeger in het kandidaats- en nu in het bachelor-programma wordt behandeld. Ik heb ooit uitgerekend dat het ongeveer 140 man-jaar zal kosten om dat allemaal te formaliseren. Een interessante dagdroom is om je af te vragen of dat er ooit van zal komen. En zo ja: wanneer dan, en in welke vorm. *Freek Wiedijk*

*HOL: http://www.cl.cam.ac.uk/~jrh13/hol-light/*
*Freeware*

**GAP**

GAP (Groups, Algorithms and Programming) ontstond aan de universiteit van Aken als geesteskind van Joachim Neubüser. Na diens pensionering is het 'hoofdkwartier' verhuisd naar St. Andrews in Schotland, waar de verdere ontwikkeling van GAP gecoördineerd wordt.

Zoals de naam al doet vermoeden, is GAP een computeralgebra systeem met voornamelijk groeptheoretische toepassingen. En inderdaad heeft de meeste functionaliteit van GAP daarop betrekking. GAP is echter ook gemaakt met het doel 'open' te zijn, in tweeërlei opzicht: ten eerste kan men GAP 'gratis en voor niets' van de website halen, waarna men over alle broncode beschikt, en ten tweede heeft GAP een programmeertaal waarmee het mogelijk is nieuwe types objecten te introduceren. Dit laatste kan men gebruiken om eigen programma's in GAP te integreren. Bij voldoende enthousiasme is het dan mogelijk om de code te organiseren in een zogenaamd 'package' dat bij GAP aangeboden kan worden. Er volgt een beoordelingsproces, vergelijkbaar met de beoordeling van tijdschriftartikelen, waarna, bij bevredigend resultaat, het package bij GAP gevoegd wordt en van de website geladen kan worden.

Ter illustratie noem ik een voorbeeld uit eigen ervaring. In 2007 heb ik wat gerekend met nilpotente banen in enkelvoudige Lie algebras, dit om het vermoeden van Elashvili voor Lie algebras van exceptioneel type aan te tonen. Hiervoor heb ik in GAP een nieuw type object geïntroduceerd (namelijk, het type 'nilpotente baan', in GAP-taal: `IsNilpotentOrbit`). Objecten van dit type worden op een bepaalde manier geprint:

```
gap> L:= SimpleLieAlgebra("E",6,
  Rationals);;
gap> o:= NilpotentOrbit(L,
  [0,0,0,2,0,0]);
<nilpotent orbit in
  Lie algebra of type E6>
```

Vervolgens zijn op objecten van dit type een aantal functies toepasbaar, bijvoorbeeld om een corresponderend $\mathfrak{sl}_2$-triple op te vragen:

```
gap> sl2:= RandomSL2Triple(o);; sl2[3];
v.4+v.8+v.19+v.23
```

De berekeningen vatte ik samen in een artikel (*LMS J. Comput. Math.*, 11:280–297, 2008) en de code zette ik op mijn website. Het laatste bleek nuttig toen de programma's werden opgehaald door Boahua Fu, die ze gebruikte in zijn onderzoek (`arXiv:0809.5109`). Als ik niet GAP's typesysteem had kunnen programmeren, was het voor hem zeker veel moeilijker geweest deze programma's te gebruiken. *Willem de Graaf*

*GAP IV: www.gap-system.org*
*Freeware*

**Macaulay, Macaulay2, Singular**

Wiskunde een experimentele wetenschap? De computer op de werkplek maakt dit mogelijk. Altijd al is het berekenen van zorgvuldig gekozen voorbeelden belangrijk geweest, maar de nu beschikbare rekenkracht opent heel nieuwe perspectieven.

In mijn eigen onderzoek heb ik veel gewerkt met het programma Macaulay, dat goed is in het vinden van standaardbases. Dat begon nadat ik in 1990 voor 2000 dollar een Mac SE30 gekocht had. Ik deed eenvoudige berekeningen, want veel langer dan een minuut hield de computer niet vol. Het bleek dat je de klok uit moest zetten. Daarna ging het beter. Omdat mijn machine niet zo snel was, en niet zoveel geheugen had, was het een hele kunst om trucs te vinden om toch tot een resultaat te komen. Al met al heb ik heel wat versele deformaties van singulariteiten uitgerekend.

Als je nu naar Dave Bayers Macaulay homepage gaat, lees je: Wait!! It's 2003. That 70's haircut really has to go. While you're at it, are you sure you want this program? It warms our hearts every time we hear of a diehard Macaulay enthusiast, but even Dave has switched to Macaulay 2.

Zelf heb ik de overstap naar de opvolger Macaulay 2 (o.a. geschreven door Mike Stillman, de andere auteur van Macaulay) niet gemaakt. Er is altijd een drempel om iets nieuws te leren, en als bepaalde dingen je dan ook nog tegenstaan, komt er helemaal niets van. Het is best zinvol lange output van berekeningen en tussenresultaten, na nauwkeurige bestudering, met de hand te vereenvoudigen. Van de **Macaulay 2** documentatie kreeg ik de indruk dat zulke operaties doodzonden zijn.

Na allerlei veranderingen in het computersysteem op mijn werk en thuis beschik ik niet meer over een werkende Macaulay versie. Wat ik nu gebruik is Singular, waarvan ik de ontwikkeling van het begin af aan gevolgd heb. Mijn eerste pogingen het programma serieus te gebruiken zijn gestrand op de onvolkomenheden van de vroege versies. Nog steeds is het zo dat ik in Singular niet alles voor mekaar krijg, wat ik in Macaulay kon doen. Het fijne aan het oude Macaulay, met zijn beperkte mogelijkheden, is dat het zo lekker primitief is. Van een programmeertaal kun je niet spreken, en er is een heel beperkte lijst van mogelijke opdrachten. Maar ik had nooit last van naamconflicten, iets wat ik met Singular, waar je veel meer mee kunt doen, wel heb. Kennelijk heb ik me verkeerde gewoontes aangeleerd.

Singular kan inderdaad veel meer. Wat erg prettig is, is dat polynomen gefactoriseerd kunnen worden. Singular rekent ook met inhomogene polynomen, in tegenstelling tot Macaulay, waar alle input gewogen homogeen moet zijn. Nu was die beperking niet

zo erg, want zonder die voorwaarde zijn de problemen vaak te moeilijk. Met de computer kun je meer aan dan met de hand, maar niet zoveel meer. Maak je een probleem een klein beetje moeilijker (bijv. een variabele meer), dan duurt de berekenening veel langer. Je komt gauw van de situatie waarin je niet merkt dat de computer rekent, tot berekeningen die na een kop koffie nog niet klaar zijn. Het grondlichaam heeft grote invloed heeft op de omvang van berekeningen.

Meestal wil je rationale coëfficienten hebben. Dat gaat eerst prima, maar als de berekeningen langer duren, kun je beter opnieuw beginnen met een eindig lichaam. In Macaulay kan het alleen zo, default is karakteristiek $31993$. Over het algemeen merk je het verschil met rationale getallen niet.

Rekenen over een eindig lichaam kan zijn voordelen hebben. Om kwadratische vergelijkingen op te lossen is het handig het getal $i$ te hebben. Een mogelijkheid is een extra variabele $w$ in te voeren met zijn minimaalpolynoom $w^2+1$. Maar je kunt ook een geschikt eindig lichaam kiezen, bijv. $\mathbb{Z}/101$, waar $10$ een wortel uit $-1$ is. Met een beetje geluk kun je de resultaten dan terugvertalen door bijvoorbeeld $9$ als $i-1$ op te vatten. Dit is een voorbeeld van het gebruik van de volgende variant op de hoofdstelling van de algebra: *ieder polynoom uit* $\mathbb{Z}[x]$ *heeft een lineaire factor* $(x-n)$ *modulo een geschikt gekozen priemgetal* $p$. In het algemeen werkt $n=2$: je vult gewoon $2$ in het polynoom in, en neemt een (niet te kleine) priemfactor van het resultaat als karakteristiek.

Tenslotte, wat bereken ik zoal? Laatst was ik geinteresseerd in irreguliere oppervlakken: gladde complexe oppervlakken (reëel vierdimensionaal) met positief eerste Betti getal. De algemene hypervlaksnede is dan geen normaalkromme, ofwel deze kromme is de projectie van een kromme in een hogerdimensionale ruimte van dezelfde graad. Het eenvoudigste geval is een elliptisch regeloppervlak van graad $5$ in $\mathbb{P}^4$. Zo'n oppervlak kan door heel mooie vergelijkingen gegeven worden. Er zijn vijf vergelijkingen van graad $3$, en de relatiematrix is een $5\times 5$-matrix $L$ met $L_{ij}=s_{i-j}x_{2i-j}$, waarbij de indices modulo $5$ genomen moeten worden, en $s_k=-s_{-k}$ constanten zijn, die de elliptische modulus bepalen. De vector $(x_0,x_1,x_2,x_3,x_4)^t$ ligt in de kern van $L$ en daarom zijn de kolommen van de matrix van $4\times 4$ onderdeterminanten steeds veelvouden van deze vector: dit geeft de vijf polynomen van graad $3$; als je ze expliciet uitschrijft, is het al minder mooi. Voor $s_1=s_2=1$ heb ik met Singular wat gerekend aan een algemene hypervlaksnede, gegeven door $x_0+2x_1+3x_2+4x_3+5x_4=0$. Zoals het hoort kreeg ik voor de normalisatie van de resulterende vijfdegraads elliptische kromme vijf tweedegraadsvergelijkingen, die er echter nogal ingewikkeld uitzagen. De volgende stap is het berekenen van de relatiematrix. Bij geschikte keuzes is dit een antisymmetrische $5\times 5$-matrix en de vijf vergelijkingen zijn dan de Pfaffianen van deze matrix (de wortels van de hoofdminoren, want dat zijn volledige kwadraten). Natuurlijk maakt de computer de verkeerde keuzes, maar door te vegen kun je de matrix antisymmetrisch maken. Dit helpt een beetje, maar erg mooi wordt het niet. En dat terwijl je de elliptische normaalkromme door nog mooiere vergelijkingen kunt geven dan het elliptische regeloppervlak. Wat hier aan de hand is, is dat de mooie vergelijkingen een erg grote symmetriegroep hebben. Door een algemene hypervlaksnede te nemen, maak je de symmetrie kapot. Zoiets kun je beter niet doen. Zo zie je maar weer dat je heel gauw tegen de grens aanloopt, van wat je kunt of wilt berekenen. *Jan Stevens*

Great Moments in Computer History:
Vikings use plunder ratio calculating software.

*Macaulay: www.math.columbia.edu/~bayer/Macaulay*
*Macaulay2: www.math.uiuc.edu/Macaulay2*
*Singular: www.singular.uni-kl.de*
*Freeware*

**Magma**

Ik zag Magma voor het eerst tijdens een landelijk mastervak Algebraïsche Getaltheorie in 2004. Door de hierop volgende huiswerkopdrachten op de computer werd het programma subtiel opgedrongen. Met succes: als ik vandaag de dag expliciet een bepaald straalklasselichaam nodig heb, weet ik waar ik het moet halen.

Wat Magma destijds deed verschillen van overige pakketten was het gemak waarmee het te begrijpen is. Het programma probeert zoveel mogelijk de taal te spreken die wiskundigen zelf ook bezigen. Dit helpt. Om een voorbeeld te noemen, om $K=\mathbb{Q}(\sqrt{13})$ te creëren, maak je eerst de polynoomring in de variabele $x$ door te typen

```
R<x>:=PolynomialRing(Rationals());
```

en vervolgens bouw je $K$ via

```
K<r>:=NumberField(x^2-13);
```

De naam Magma, naar de structuur gevormd door een verzameling met een binaire operatie, doet grotere algemeenheid vermoeden dan het programma eigenlijk biedt: het werkt voornamelijk in de algebraïsch-meetkundige en groepentheoretische hoek, waar het dan ook erg dominant is. Toch heeft het programma wel degelijk Bourbakistische pretenties. Veel functies werken alleen op objecten die in de juiste categorie leven, en weigeren pedant enige andere input te accepteren, ook al zou voor een goed verstaander duidelijk zijn wat wordt bedoeld. De multiplicatieve inverse van het gehele getal 2 kent Magma bijvoorbeeld niet. Dit is even wennen, maar dwingt je wel om duidelijke algoritmes te schrijven — en zorgt ook voor wat scherpslijperige voldoening als het programma desalniettemin beweert dat de eenhedengroup van een getallen*lichaam* eindig voortgebracht kan worden. `UnitGroup(K)` is een `Abelian Group isomorphic to Z/2 + Z`

Voor mij persoonlijk is er nog een reden om Magma te gebruiken: een aantal prachtige recente algoritmes in modulaire arithmetische meetkunde zijn in Magma geïmplementeerd. Voor bijvoorbeeld Fuchsgroepen bij quaternionalgebra's, ontwikkeld door John Voight, en voor Hilbert modulaire vormen, ontwikkeld door Lassina Dembélé en Steve Donnelly.

Een voorbeeld met deze algoritmes. Stel, we willen de automorfe eigenvormen van gewicht 2 berekenen voor de Shimurakromme horend bij het ideaal $I=(2)$ in $O_K$. Na `I:=ideal<Integers(K)|2>` typen we dan

```
>Q:=QuaternionAlgebra(I,[InfinitePlaces
  (K)[1]]);
>Signature(FuchsianGroup(Q));
<1, [ 2 ]>
```

Er is dus maar één eigenvorm op scalairen na. Dit klopt met de volgens Jacquet-Langlands equivalente Hilbertmodulaire aanpak:

```
>Dimension(HilbertCuspForms(K,I));
1
```

Na wat rekentijd komen de eigenwaarden van Hecke voor deze laatste spitsvorm ook uit het programma rollen. Dat gaf goed vergelijkingsmateriaal voor eigen onderzoek, waarin één

en ander op naïevere wijze was berekend. *Jeroen Sijsling*

*Magma: magma.maths.usyd.edu.au/magma/ Licensie $1200; studentenlicensie $100.*

**Mathematica**

Er zijn allerlei computeralgebra pakketten, meestal ontworpen voor en door specialisten in een tak van wiskunde. Als je een probleem hebt dat daar in past, is zo'n pakket geweldig. Maar als je bijvoorbeeld een niet-commutatieve situatie hebt, dan is een pakket ontworpen door commutatieve algebraïci een ramp, ook al kan het pakket in principe alles. Wat ik aan Mathematica zo prettig vind, is dat je er gemakkelijk zaken in kunt toevoegen. Als een berekening op papier uit de hand is gelopen kom ik met Mathematica meestal verder. Zo ben ik momenteel bezig geschikte graad twee cocykels te zoeken in complexen van tensorprodukten van geadjungeerde voorstellingen van een (kleine) algemene lineaire group over de ring van de gehele getallen $\mathbb{Z}$. Mathematica kende van dit alles alleen $\mathbb{Z}$. Geen probleem: het pakket is gebaseerd op het herkennen van patronen en het voorschrijven van vervangingsregels. Verder is er niet besloten dat je in een bepaalde stijl moet programmeren, dus object-georiënteerd mag bijvoorbeeld, maar hoeft niet.

Ik heb dus tensorproducten nodig. In mijn berekeningen worden de tensoren op een basis uitgeschreven. Ik moet Mathematica dus leren om $a \otimes (b+c) \otimes d$ altijd te expanderen tot $a \otimes b \otimes d + a \otimes c \otimes d$, evenals $a \otimes (3 * b)$ tot $3 * (a \otimes b)$. Enzovoort. Dat gaat dan zo:

```
tensor[u___, v_ + w_, x___]:=
tensor[u,v,x]+tensor[u,w,x];
tensor[u___, v_?NumberQ * w_, x___]:=
v*tensor[u,w,x]
```

Of mysterieuzer, maar efficiënter:

```
tensor[u___, v_Plus, x___]:=
tensor[u,#,x]&/@v;
tensor[u___, v_?NumberQ * w_, x___]:=
v*tensor[u,w,x]
```

Dat lijkt als een trein te werken, tot je opeens een tensor met component nul in je output ziet. Dus ook even toevoegen:

```
tensor[u___,0,x___]:=0
```

Als je berekening kan worden georganiseerd als veel herhalen van substituties, wat vaak het geval is, dan is Mathematica een uitkomst. Natuurlijk gebruik je ook wel eens een wiskundig algoritme dat Mathematica kent. In mijn geval het LLL algoritme, wat overigens in een eerdere versie fout was geïmplementeerd. Dat algoritme helpt om eenvoudige cocykels te vinden onder de mogelijke cocykels. Als ik thuiskom kijk ik of er al output is. *Wilberd van der Kallen*

*Mathematica VII: www.wolfram.com Volledige versie: ca. € 3200; Home Edition: ca. € 300.*

**SAGE**

Sage is een project met een missie: *Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab*. Onder leiding van William Stein (Seattle) wordt Sage in hoog tempo ontwikkeld door een grote groep actieve wiskundigen en programmeurs. Ze zijn nog maar vier jaar bezig, en het is al een indrukwekkend systeem, dat ik zelf met veel plezier gebruik.

Een sterk uitgangspunt bij de ontwikkeling van Sage is het streven om niet het wiel opnieuw uit te vinden. Voorbeeld: elk computeralgebra-systeem heeft een *interpreter* nodig, waarmee commando's van de gebruiker in een bepaalde vorm verwerkt worden. Allerlei systemen hebben zelf interpreters ontworpen met allemaal hun eigen gebruiksaanwijzing en gebreken. By Sage hebben ze goed rondgekeken naar gangbare computertalen die hun bestaan bewezen hebben. De taal *Python* kwam als beste uit de bus en Sage is daarom gebouwd op Python. Dit heeft een enorm voordeel: alle verbeteringen en uitbreidingen die anderen voor Python ontwikkelen kunnen meteen worden gebruikt. Als ik bijvoorbeeld berekeningen wil doen die interactie met een *database server* vergen, dan zijn daar al standaard Python modules voor die door duizenden mensen gebruikt worden, en die zonder al teveel gedoe ingezet kunnen worden. Probeer dat maar eens vanuit Maple.

Over de praktische, politieke en religieuze voordelen van *open source* wordt wellicht meer geschreven dan over computer algebra. Laat ik hier volstaan met een eenvoudige observatie. Ik heb er wel eens last van dat ik in Maple of Magma niet kan achterhalen *hoe* iets berekend wordt. Soms wil ik dat weten omdat er iets niet klopt, soms wil ik het weten vanuit een interesse in algoritmes en soms wil ik weten waarom het zo lang duurt. Er is typisch geen enkele manier om hierachter te komen — het gebruik van de software is als het raadplegen van een orakel. Voor het gebruik in serieus wiskundeonderzoek is dat op zijn best onbevredigend. Open source software is wellicht lastig te ontcijferen, maar wie echt wil, kan erachter komen wat er gebeurt. Bovendien is het in een open source systeem mogelijk om tijdkritische stappen in een berekening efficiënter te gaan aanpakken, waardoor het hele systeem meer slagkracht krijgt.

Einstein (8ste van rechts) bij het Yerkes Observatory in 1921

Op gebruikersgemak scoort Sage goed. De grafische omgeving is te gebruiken via een *browser* — weer zo'n wiel — en alle functies zijn te vinden door slimme *tab completion*.

Kijk eens op de webpagina. Volgens mij heeft dit project de toekomst. *Bart de Smit*

*SAGE: www.sagemath.org Freeware*

**Een (Japanse) collega**

Soms is het handig wanneer in plaats van een computer, een collega een berekening uitvoert. Bedenk hierbij dat men zegt dat vrouwen nauwkeuriger rekenen dan mannen — vandaar wellicht de vele dames op de foto bij Einstein (zie boven).

Een aantal jaren geleden: ik ben in Japan en zit met Fumiharu Kato te staren naar een cohomologiegroep waar we niet uitkomen. Het is groepencohomologie op een oneindig-dimensionale vectorruimte. Het is vrijdag en Kato deelt mij mee dat hij zaterdag met de familie moet doorbrengen, maar dat we op zondag wel weer aan het werk kunnen.

Zondagochtend komen we elkaar tegen in een café bij de Universiteit van Kyoto voor ホットコーヒー ('hotto kohi'). Kato pakt een schrift, zegt: "So yesterday I was with my family, and I did the following calculations", en laat bladzij na bladzij zien met eindig-dimensionale benaderingen van het probleem. Dit zegt ook iets over Japanse familiemores. Het valt meteen op dat het resultaat na een paar benaderingen stabiel wordt. Een kopje koffie later zien we hoe we het probleem met Hilbert 90 kunnen herleiden naar een korte, eindig-dimensionale berekening. Ik weet niet of we dat ooit hadden ontdekt zonder die pagina's vol berekeningen.

Moraal: neem volgend weekend je schriftje mee naar Artis. *Gunther Cornelissen*

*Collega's: Overal gratis beschikbaar, binnen Nederland wel in steeds kleinere aantallen.*