The Ax Computer Algebra System

Roman Pearce

July 2026

On the internet at `http://axcas.net`

- download binaries or run it on the server
- text based interface with downloadable output
- not open source, but Gröbner basis code is released
- experimental, compact C code, no external dependencies

Quick Tour

```
A = randommatrix(100,100,10);
determinant(A); # 125 digits, 0.03 sec
```

Let $A, B \in \mathbb{Z}[x_1, \ldots, x_n]$ for $n > 2$:

```
gcd(A,B); # Huang and Monagan separating terms
```

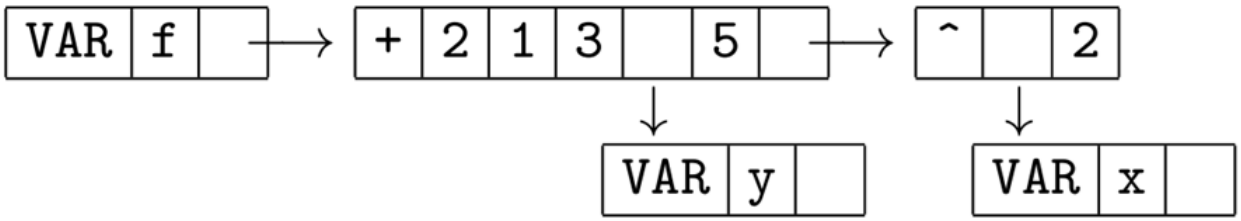
```
digits(30); realroot(x-cos(x),x); # Ridder's method
   0.739085133215160641655312087673
```

```
digits(30); quadrature(sin(x),x,0,1); # tanh-sinh
   0.459697694131860282599063392557
```

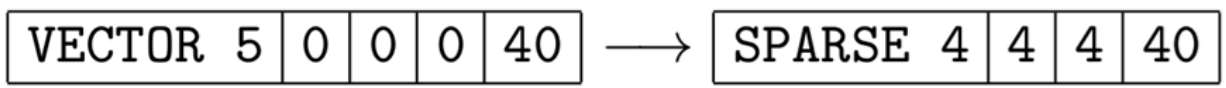
Rule of thumb: start with twice as many digits as desired.

Formula Representations

```
f = 2 + 3*y + 5*x^2;
```



- speed limited by the number of memory references
- immediate representations for small numbers
- simplify to more efficient representations
- pre-allocate temporary work space



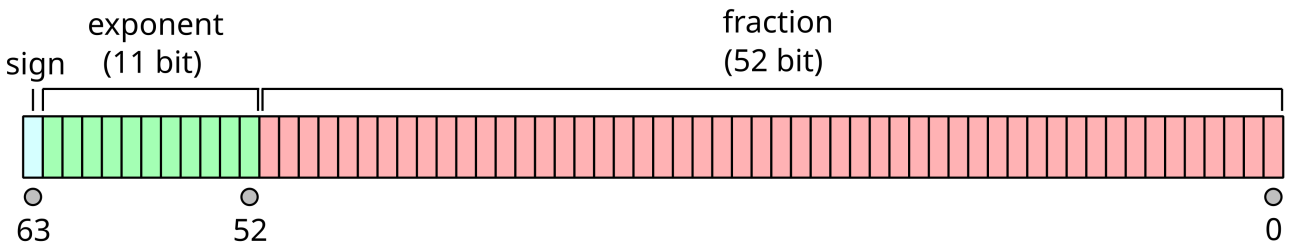
Immediate Numbers

Problem: dealing with objects (dags) is slow.

The bottom 3 bits in a 64-bit pointer are zero so encode values:

$(x \,\&\, 7)$	some kind of immediate value
$(x \,\&\, 3) = 2$	a signed integer in the top 62 bits
$(x \,\&\, 7) = 4$	a rational a/b, a signed, 31 and 30 bits
$(x \,\&\, 1)$	a floating point number with one bit missing

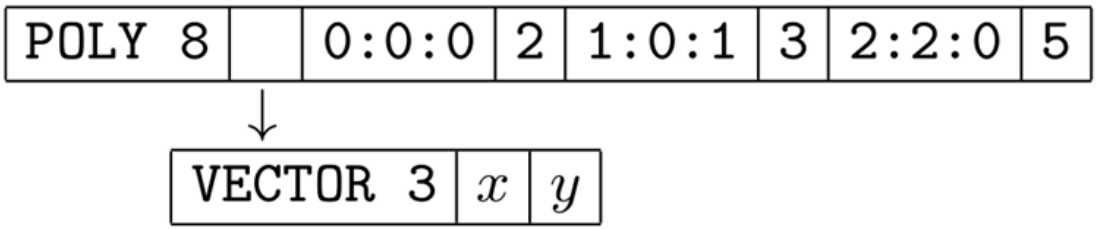
Wikipedia:



- we take a bit from the exponent if possible
- fall back on creating a dag when $|exponent| \geq 512$

Polynomial Representation

```
f = 2 + 3*y + 5*x^2;
```



- developed by Monagan and Pearce for Maple 17 (2013)
- immediate monomials in a machine friendly format
- heap algorithms for multiplication and division
- rational and floating point coefficients too

$f = (1 + x + y + z + t)^{20}$	chained heap	naive heap	no POLY
`f = expand(f);`	0.032	0.036s	0.082s
`g = expand(f*(f+1));`	1.515	4.995s	26.910s
`divide(g,f);`	1.959	5.344s	32.547s

Vectors and Matrices

Ax uses Iliffe vectors:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

MATRIX 4					
row_0	→	VECTOR 4	a	b	c
row_1	→	VECTOR 4	d	e	f
row_2	→	VECTOR 4	g	h	i

Notice how A_{ij} becomes `A[i][j]` in C.

We also support Matlab's `A'` (transpose) and `A\b` (linear solve).

- vectors simplify to a sparse or dense format automatically
- updating a vector is $O(n)$, updating a matrix is $O(m+n)$
- row operations are fast, e.g. `A[1] += 2*A[2];`

Linear Algebra

Tricky tradeoffs between sparse and dense algorithms.

- floating point matrices have dense double precision code
- integer matrices have dense modular methods; 63-bit primes
- dense machine code parallelized with no extra memory

Benchmarks on a Mac Mini M4 (10 cores):

determinant over $\mathbb{Z}$ where $\lvert A_{ij}\rvert < 10$				
dense	digits	sequential	parallel	speedup
100×100	125	0.035s	0.033s	1.060x
500×500	797	3.787s	2.364s	1.602x
1000×1000	1744	66.743s	24.650s	2.707x

Adding sparse algorithms for matrices and graphs is ongoing work.

User Language

```
function euclid(a,b)
{
    var r;
    while (b != 0) {
        r = a % b;
        a = b, b = r;
    }
    return a;
}
```

```
function collatz(n)
{
    if (n < 2) return 0;
    if (n % 2) return collatz(3*n+1)+1;
    return collatz(n/2)+1;
}
```

- no `i++;` or `i--;` use `i+=1;` or `i-=1;`
- garbage collection only after top-level statements

Benchmark of the sum of the first 10^7 Collatz values.

language	C	Ax	Ax	Maple	Maple
memoization	no	yes	disabled	yes	no
time	0.140s	1.030s	145.441s	20.369s	366.594s

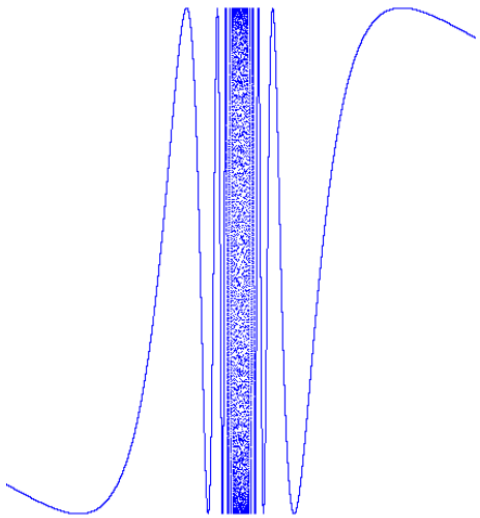
Plotting

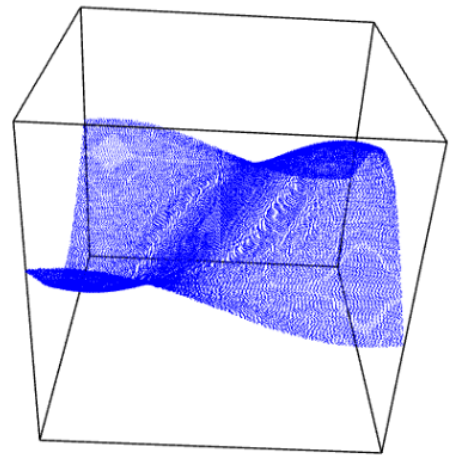
```
plot(sin(1/x),x,-2,2,-2,2);
```

- 2D plots generate a .gif file on the server with html axes
- uses *oversampling* to give the illusion of a line

```
plot3d(sin(x)*cos(y),x,y,-2,2,-2,2,-2,2);
```

- 3D plots generate data viewable in Javascript/WebGL
- draws *voxels* giving the illusion of a surface





Portability

Binaries released for seven 64-bit platforms:

```
ax.exe      Windows x64 (Intel/AMD)
axarm.exe   Windows Arm64 (Qualcomm/Nvidia)
ax.mac      Mac Arm64 (Apple)
ax.macx64   Mac x64 (Intel)
ax          Linux x64 (Intel/AMD)
axarm       Linux Arm64 (Qualcomm/Nvidia)
axrv        Linux Risc-V (SiFive/Sophgo/Ky)
```

No external dependencies, but optional assembly/intrinsics:

```
lo = umul2(a,b,&hi);      for a*b = (lo:hi)
q = udiv2(lo,hi,v,&r);    for (lo:hi)/v = (q,r)
c = uadd2(&lo,&hi,a,b);   for (lo:hi) += (a:b) ret. carry
c = usub2(&lo,&hi,a,b);   for (lo:hi) -= (a:b) ret. borrow
```

- Möller and Granlund's 2/1 reciprocal division used extensively
- New platforms (Windows Arm, Linux Risc-V) took 5 minutes

Gröbner Bases

Ax started as a wrapper for a Gröbner basis engine.

Goal stretched to a polynomial system solver.

Solver requires factorization, gcd, and so on.

grevlex Gröbner bases modulo p (F4)					
$p = 2^{61} - 1$	cyclic-9	katsura-12	eco-12	noon-9	reimer-9
parallel	13.266s	7.426s	0.627s	4.569s	3.548s
sequential	58.936s	34.247s	1.705s	6.795s	13.866s
speedup	4.442x	4.611x	2.719x	1.487x	3.908x

- Gröbner code (F4/FGLM/Walk) is largely standalone
- started releasing that code into the public domain
- model for collaborations

Some Important Books

- Algorithms for Computer Algebra (Geddes, Czapor, Labahn)
- A Course in Computational Algebraic Number Theory (Cohen)
- Modern Computer Algebra (von zur Gathen, Gerhard)
- Numerical Recipies in C (Press et al.)
- Real Computing Made Real (Acton)
- ISSAC/CASC Proceedings (various)
- Symbolic Integration I (Bronstein)
- Hacker's Delight (Warren)
- Wikipedia (various)

Thank you!