`msolve`

A new open source library for algebraic system solving

by

Jérémy Berthomieu (PolSys Team, LIP6, Sorbonne Université),
Christian Eder (TU Kaiserslautern) and
Mohab Safey El-Din (PolSys Team, LIP6, Sorbonne Université)

joint work with

Wolfram Decker (TU Kaiserslautern),
Franz-Josef Pfreundt (Fraunhofer ITWM Kaiserslautern) and
Bernd Sturmfels (Max-Planck-Institute Leipzig)

#0

What do we mean by algebraic solving?

In this talk:

Let's restrict to **zero-dimensional** ideals.

Three main steps

1. Compute the **reduced Gröbner basis**
G for I w.r.t. some monomial order.
(by default **degree reverse lexicographical**)

2. **Convert** G to the reduced Gröbner basis H for I w.r.t. the **lexicographical** order.

3. **Solve** the uniquely defined **univariate polynomial** from H.

   Go on **recursively** substituting variables already solved for.

Recap:

1. Compute the **reduced** Gröbner basis G.
2. **Convert** to lexicographical basis H.
3. **Solve** univariate polynomial in H.

Let's do this from back to front.

#3
An optimized univariate solver

Main method:

Subdivision algorithm based on a variant
of Vincent's theorem and Descartes' method

1. Find $B \in \mathbb{N}$ such that all positive (real) roots are in $[0, B]$.

2. Rescale: $[0, B] \longrightarrow [0, 1]$.

3. Use Descartes' rule of sign to find number of real roots in $[0, 1]$:
   If > 1, split into $\left[0, \frac{1}{2}\right]$ and $\left[\frac{1}{2}, 1\right]$, rescale, recursion.

Main tricks for efficiency:

- Use 2-adic truncations of the coefficients.
- Apply **Taylor shift** using asymptotically fast algorithms:

  cross-over degree: 512,
  crucial degrees $\geq 10,000$.

Note:

We target polynomials from Gröbner bases, i.e.

- **large degrees** and
- even **larger bit size coefficients**.

Henrion-n

Timings in seconds, **ratio** is $\frac{\text{Time xxx}}{\text{Time msolve}}$

deg.	♯ sols	msolve	Mathematica		Maple		SLV		tdescartes	
			time	ratio	time	ratio	time	ratio	time	ratio
6!	12	0.3	1.1	3.6	1.1	3.6	0.5	1.6	0.8	2.6
7!	12	57	533	9.3	296	5.2	339	5.9	112	1.9

Katsura-n

Timings in seconds, **ratio** is $\frac{\text{Time xxx}}{\text{Time msolve}}$

deg.	♯ sols	msolve	Mathematica		Maple		SLV		tdescartes	
			time	ratio	time	ratio	time	ratio	time	ratio
2^9	120	1.3	1.2	0.9	2.3	1.8	1.8	1.4	9.8	7.5
2^{10}	216	12.5	15	1.2	27.7	2.2	26	2.0	74	5.9
2^{11}	326	98.1	156	1.6	333	3.4	359	3.7	1,055	10.8
2^{12}	582	1,010	3,808	3.7	8,931	8.6	7,151	6.9		
2^{13}	900	8,975								
2^{14}	1,606	82,004								

Future plans:

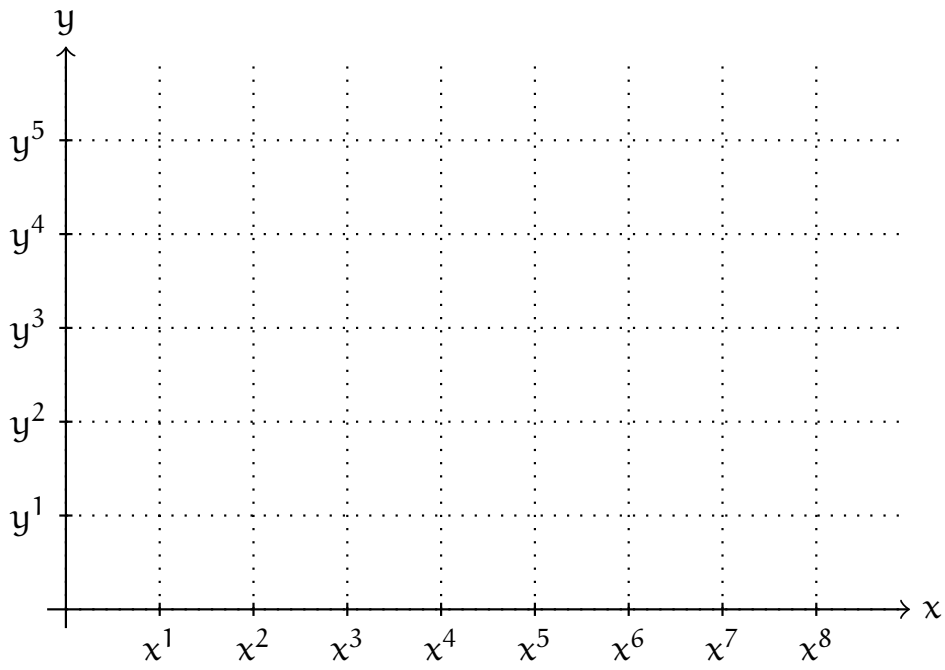
- Exploit FFT precomputations to speed up the asymptotically fast Taylor shift.
- Introduce Newton iterations for better performance on clusters of roots.

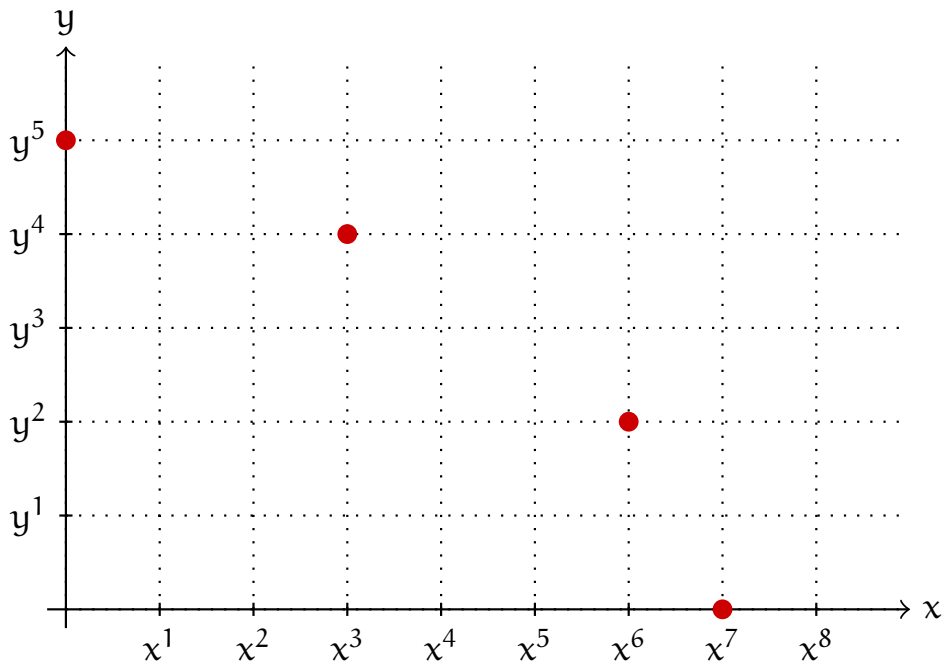
#2

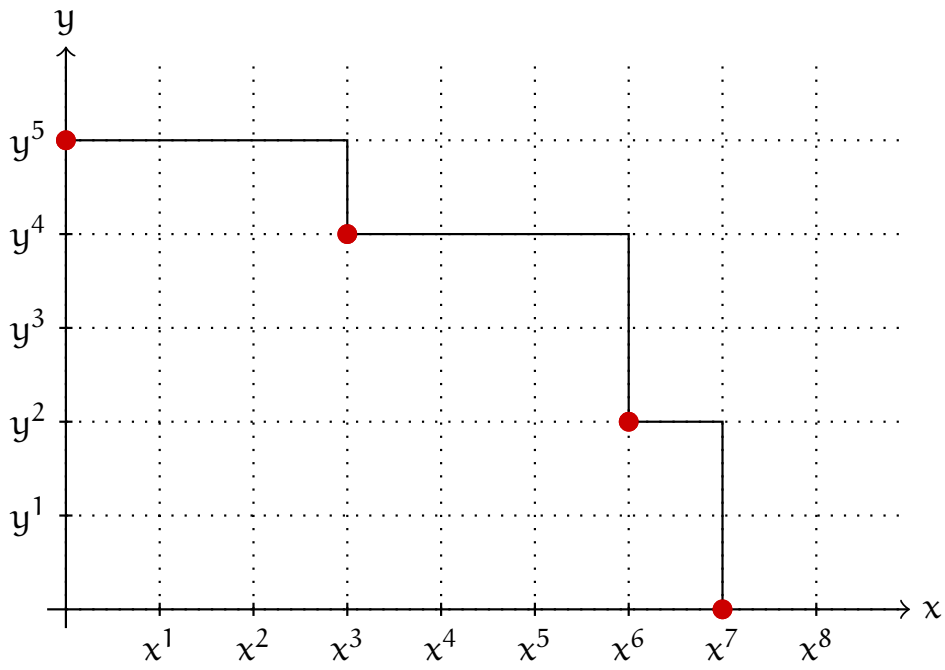
`FGLM`

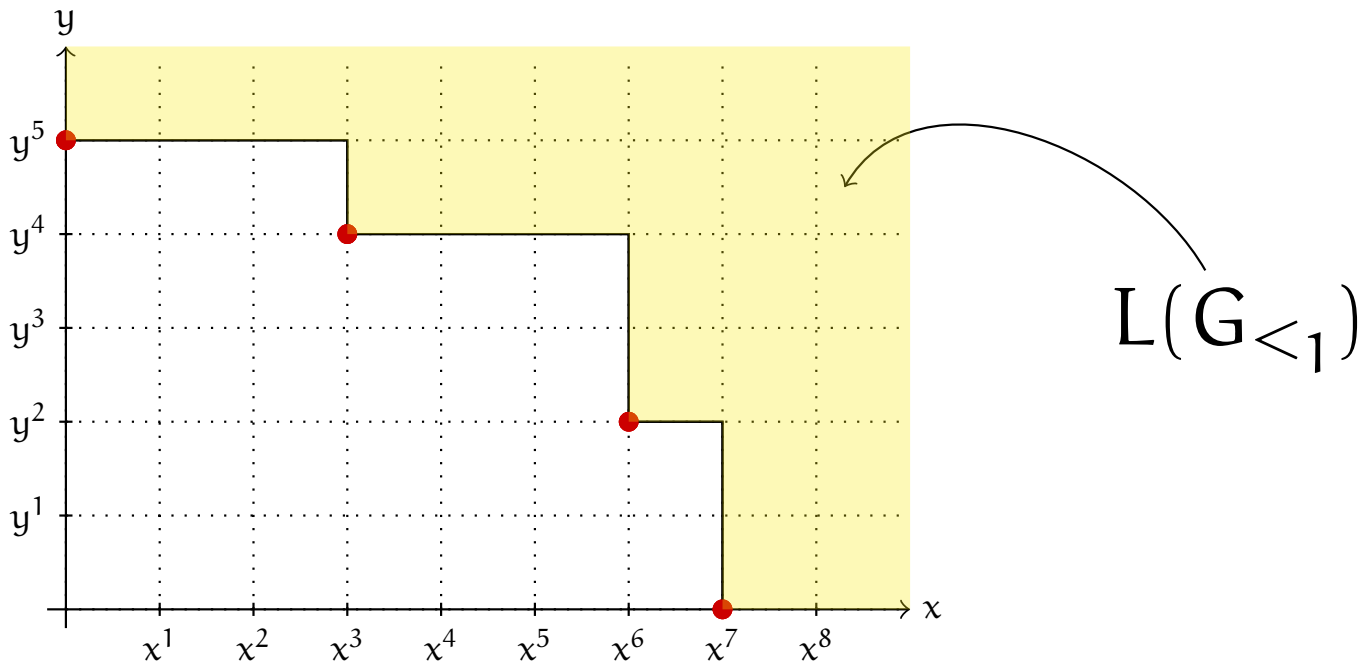
A sparse Gröbner basis conversion algorithm

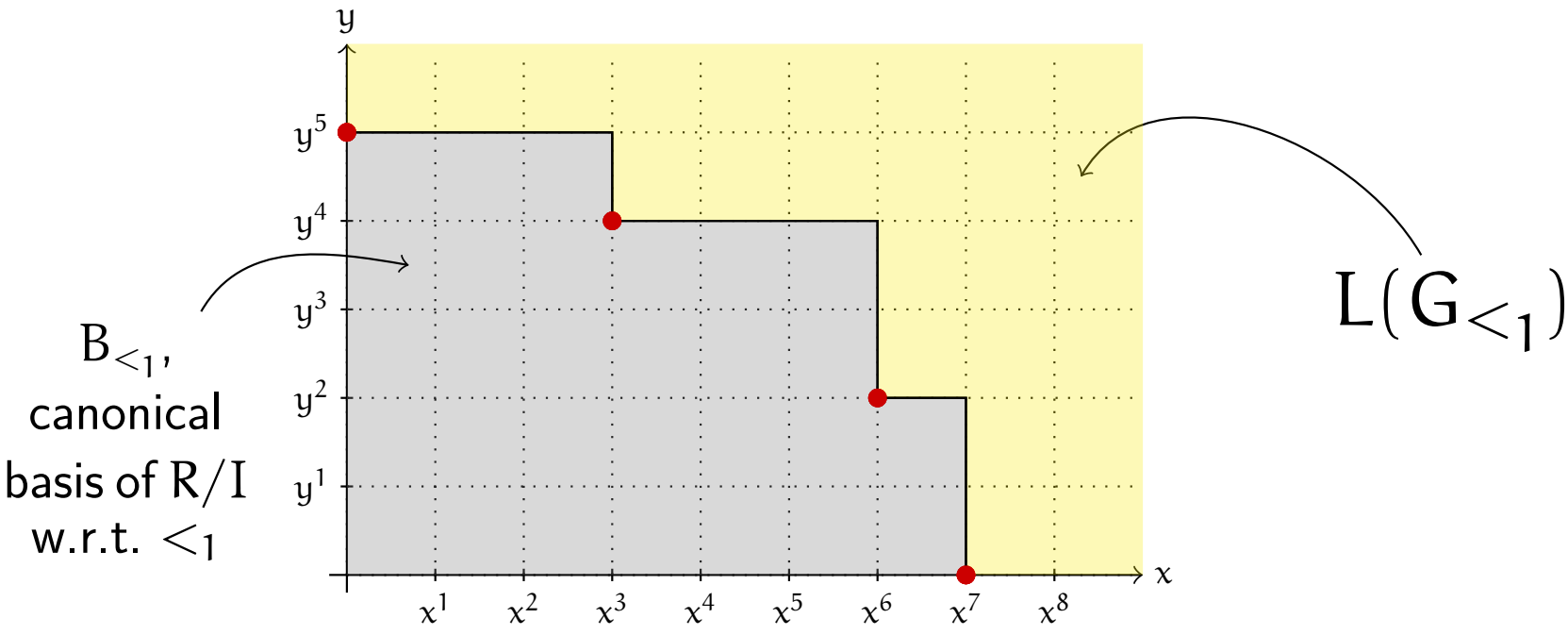
If I is zero-dimensional we can go from $<_1$ to $<_2$ **directly**.


y
y^5
y^4
y^3
y^2
y^1
x
x^1
x^2
x^3
x^4
x^5
x^6
x^7
x^8


y
y^5
y^4
y^3
y^2
y^1
x
x^1
x^2
x^3
x^4
x^5
x^6
x^7
x^8


y
y^5
y^4
y^3
y^2
y^1
x
x^1
x^2
x^3
x^4
x^5
x^6
x^7
x^8


y
y^5
y^4
y^3
y^2
y^1
x
x^1
x^2
x^3
x^4
x^5
x^6
x^7
x^8
$L(G_{<_1})$


y
y^5
y^4
y^3
y^2
y^1
x
x^1
x^2
x^3
x^4
x^5
x^6
x^7
x^8
$B_{<_1}$, canonical basis of R/I w.r.t. $<_1$
$L(G_{<_1})$

- Use multiplication matrices $T_i \in K^{D \times D}$.
- Multiply all possible terms, sorted by $<_2$.
- Once we find a relation, we have a new element in $G_{<_2}$.

We use a **sparse** variant of `FGLM` due to Faugère and Mou.

Simplify again (for this talk):

Assume I is in **shape position**,
i.e. G_{LEX} is given by

$$\{f_1(x_1), x_2 - f_2(x_1), \ldots, x_n - f_n(x_1)\}$$

where $\deg f_1 = D$.

We first compute $f_1(x_1)$ applying **Wiedemann**'s algorithm:

- Generate a linearly recurring sequence $s = \left[\langle r, T_1^k e\rangle : 0 \leq k \leq 2D - 1\right]$.
- Apply **Berlekamp-Massey** algorithm to get the minimal polynomial $\tilde{f_1}$.

Easiest case: If $\deg \tilde{f_1} = D$ then $\tilde{f_1} = f_1$ and I is in shape position.

- There exists a **deterministic** version of Wiedemann's algorithm.
- We can use **Berlekamp-Massey-Sakata** algorithm if I is not in shape position.

So we have constructed $f_1(x_1)$.

Now generate $f_2(x_1), \ldots, f_n(x_1)$:

$$f_i(x_1) = \sum_{k=0}^{D-1} c_{i,k}\, x_1^k.$$

We want to have

$$x_i - f_i(x_1) = x_i - \sum_{k=0}^{D-1} c_{i,k}\, x_1^k \in I.$$

In other words:

$$\mathrm{NF}\left(x_i - \sum_{k=0}^{D-1} c_{i,k}\, x_1^k\right) = 0.$$

Reinterpreting this in linear algebra:

$$v_i := T_i e = \sum_{k=0}^{D-1} c_{i,k}\, T_1^k e.$$

Try to reuse data already computed in s:

$$\left\langle \mathrm{r}, \mathrm{T}_1^{\mathrm{j}} \mathrm{v}_{\mathrm{i}} \right\rangle = \sum_{\mathrm{k}=0}^{\mathrm{D}-1} \mathrm{c}_{\mathrm{i},\mathrm{k}} \left\langle \mathrm{r}, \mathrm{T}_1^{\mathrm{k}+\mathrm{j}} \mathrm{e} \right\rangle$$

for $0 \leq \mathrm{j} \leq \mathrm{D} - 1$.

Entries $\left\langle \mathrm{r}, \mathrm{T}_1^{\mathrm{k}+\mathrm{j}} \mathrm{e} \right\rangle$ generate a **Hankel matrix**.

There exists an efficient algorithm to solve these linear equations (Brent et al.)

Main trick for efficiency

- Exploit **structure** of multiplication matrices, isolate **dense** parts and apply dense linear algebra using intrinsics (AVX2).

Future plans

- Add Berlekamp-Massey-Sakata algorithm.
- Add efficient sparse resp. hybrid linear algebra.
- Optimize for 8-, 16- and 31-bit coefficients.

#1

Efficient Gröbner basis algorithms

Main algorithm at the moment:

Faugère's F4 Algorithm

$I = \langle f_1, \ldots, f_m \rangle \subset K[x_1, \ldots, x_n]$

$G \leftarrow \{f_1, \ldots, f_m\}$

$P \leftarrow \{(f_i, f_j) \mid 1 \leq i < j \leq m\}$

While $(P \neq \emptyset)$ do

Choose subset $L \subset P$**,** $P \leftarrow P \setminus L$

$L \leftarrow$ **symbolic preprocessing**(L, G)

$L \leftarrow$ **linear algebra**(L)

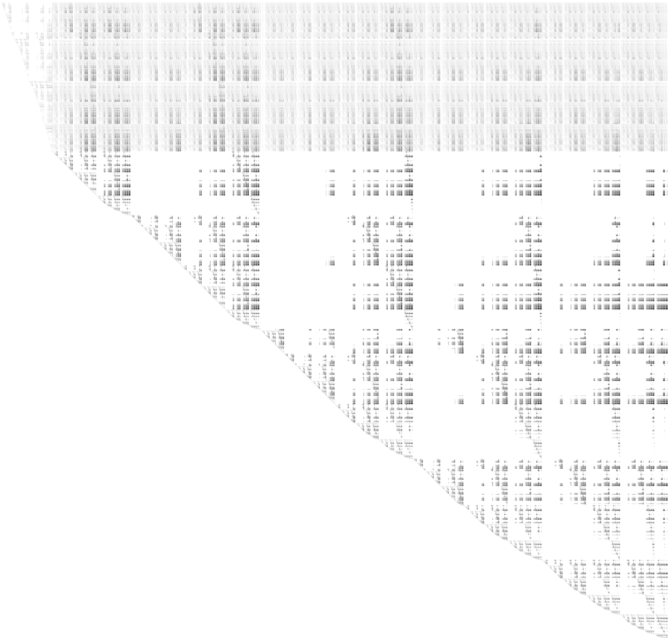
for $h \in L$ **with lm**$(h) \notin L(G)$

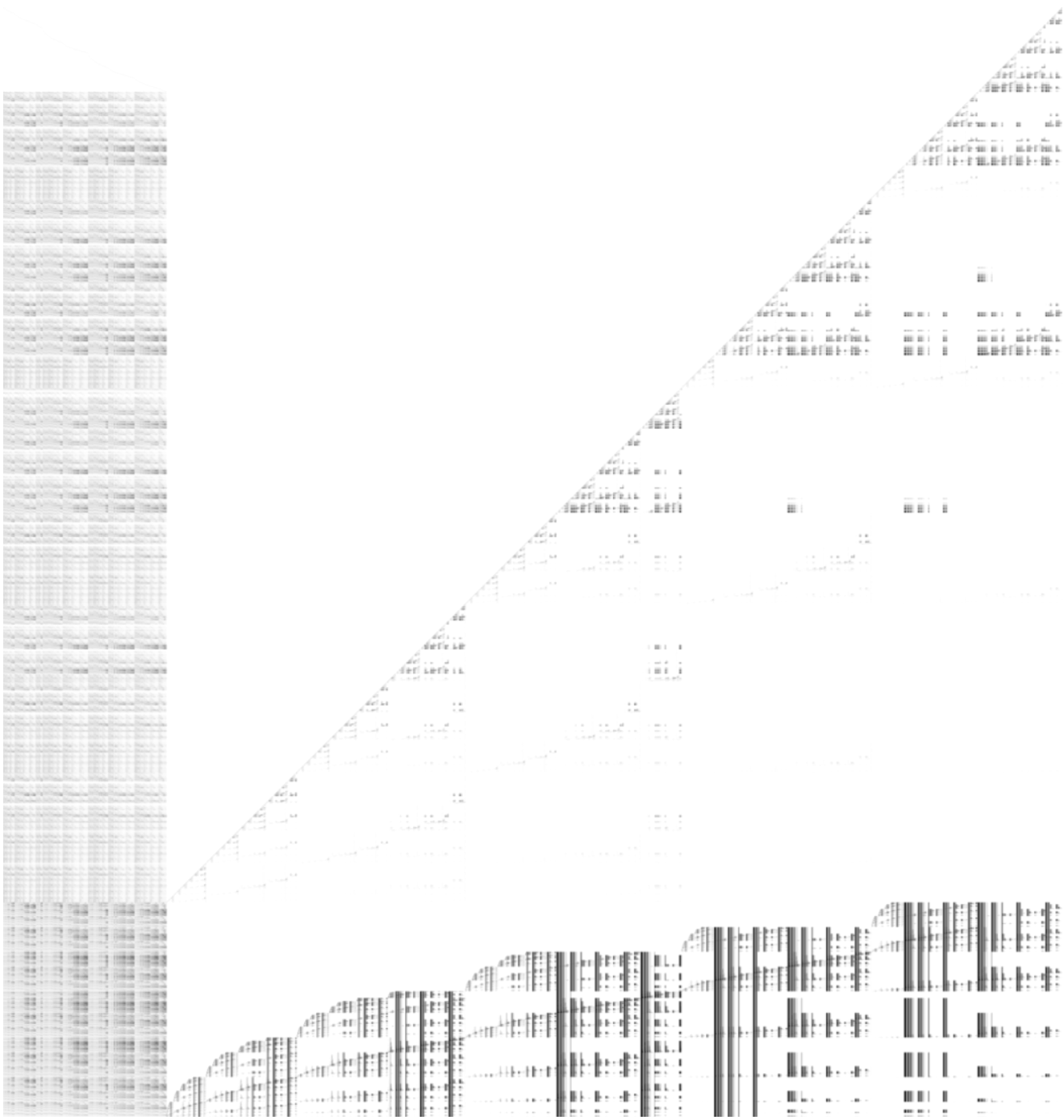
$P \leftarrow P \cup \{(h, g) \mid g \in G\}$

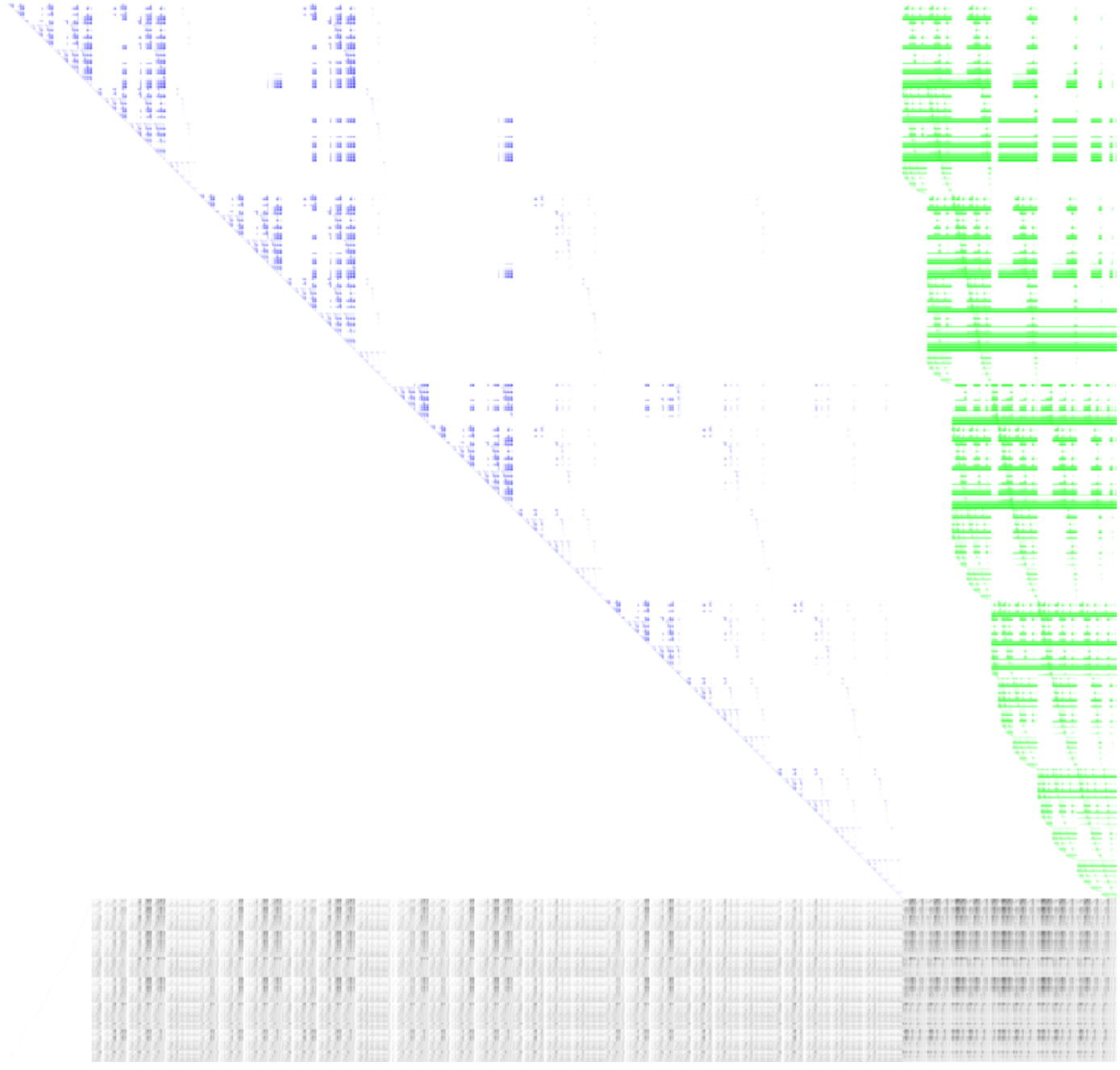
$G \leftarrow G \cup \{h\}$

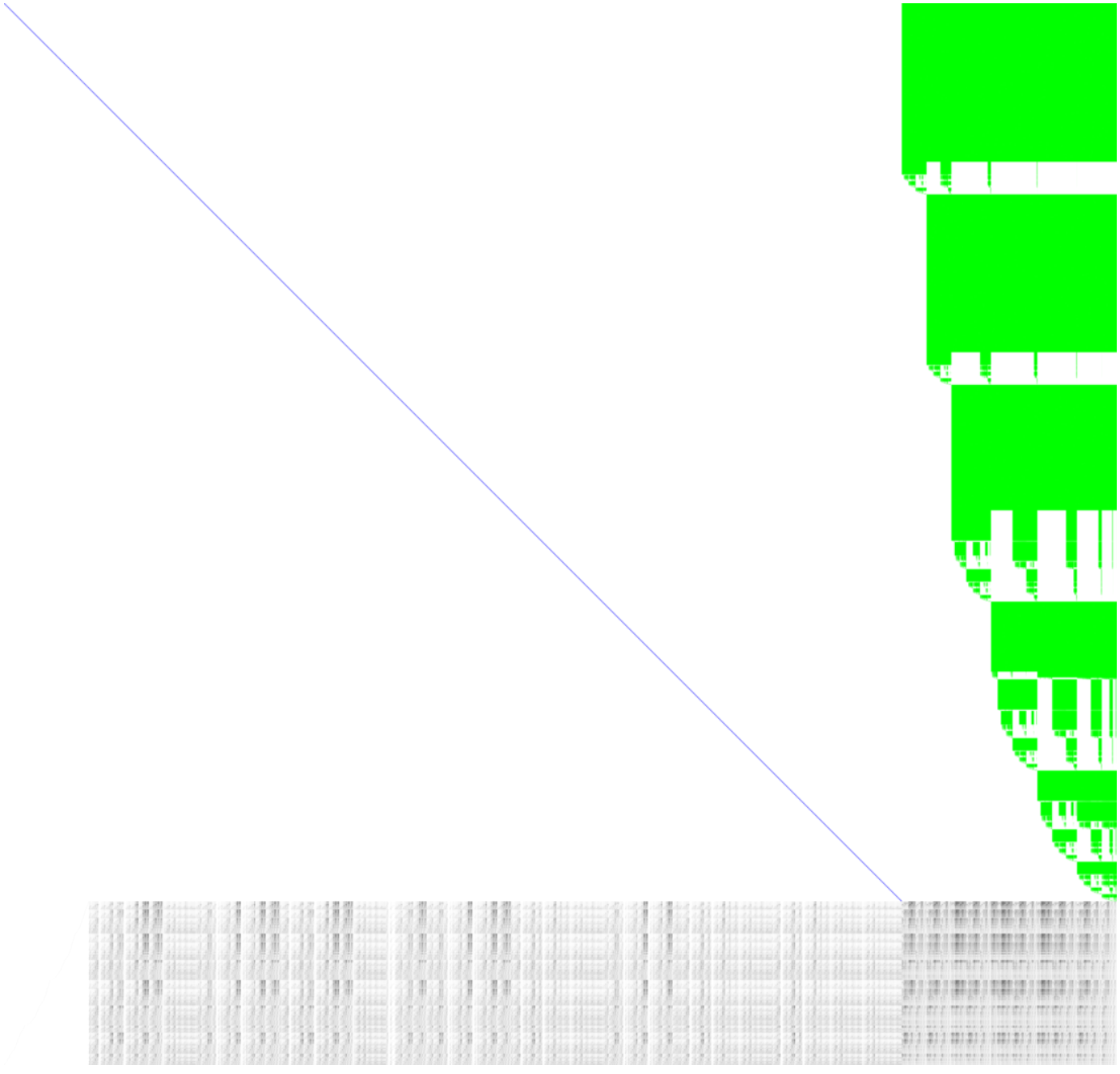
Return G

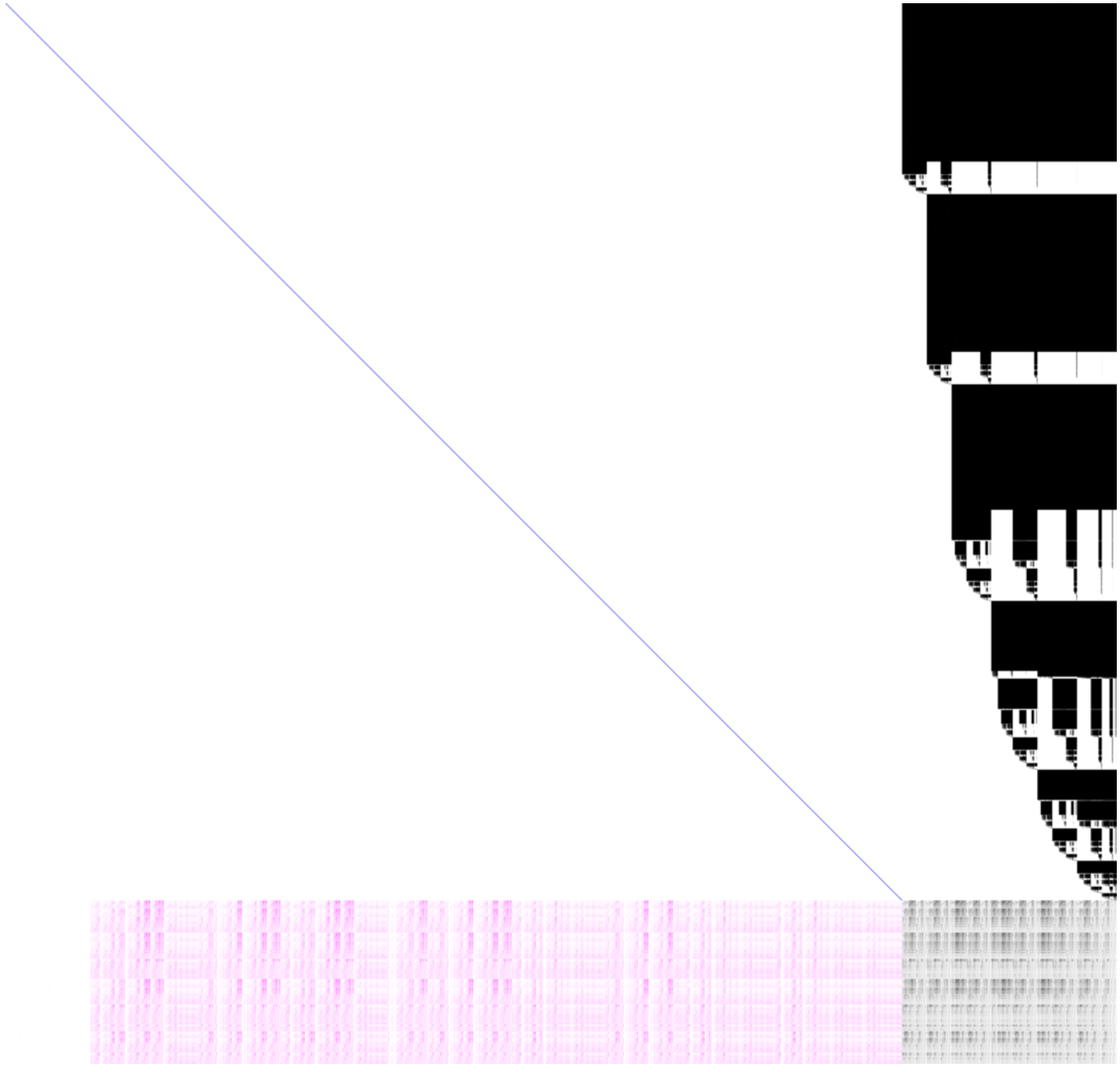
Special linear algebra (GBLA)

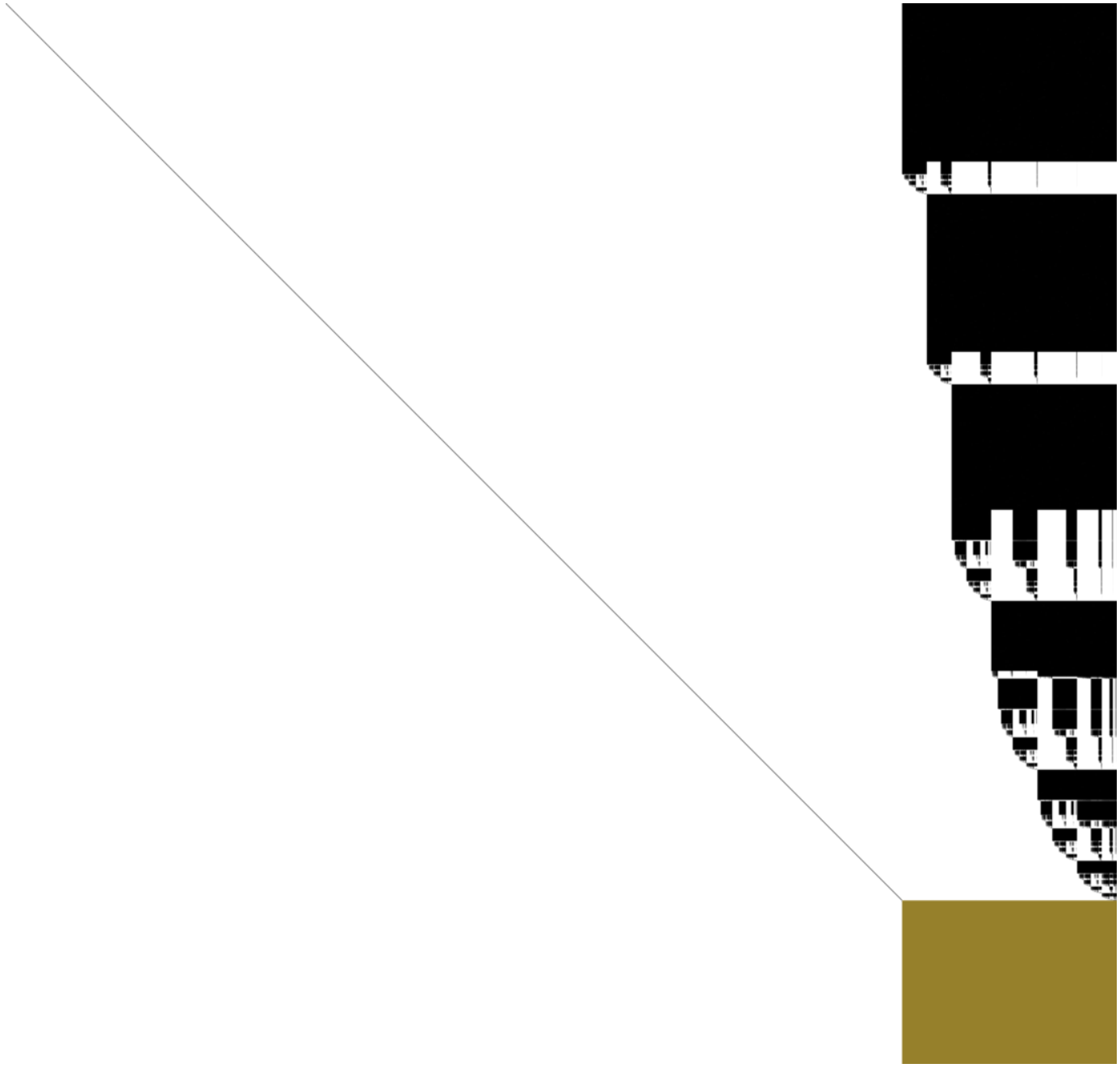


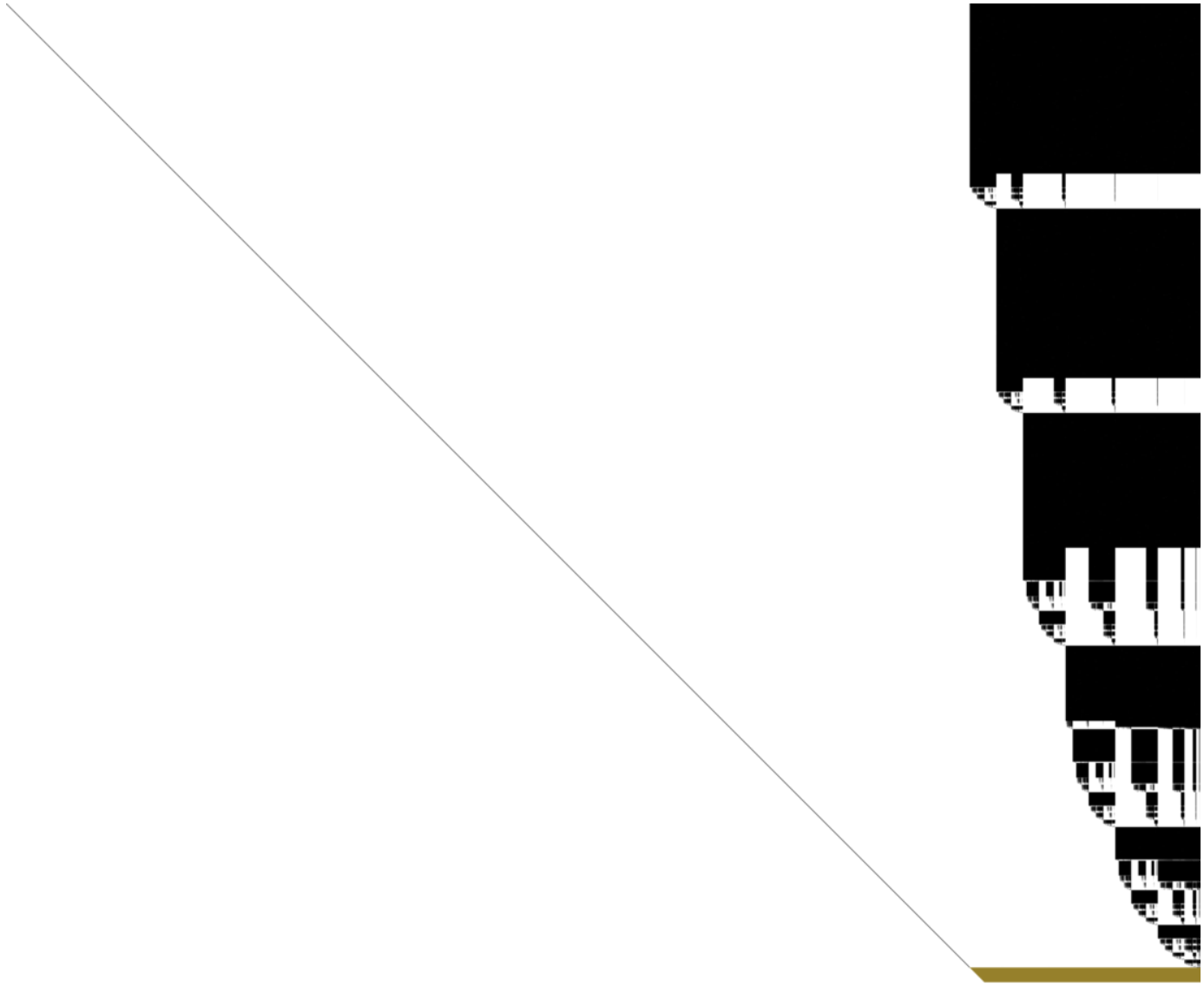






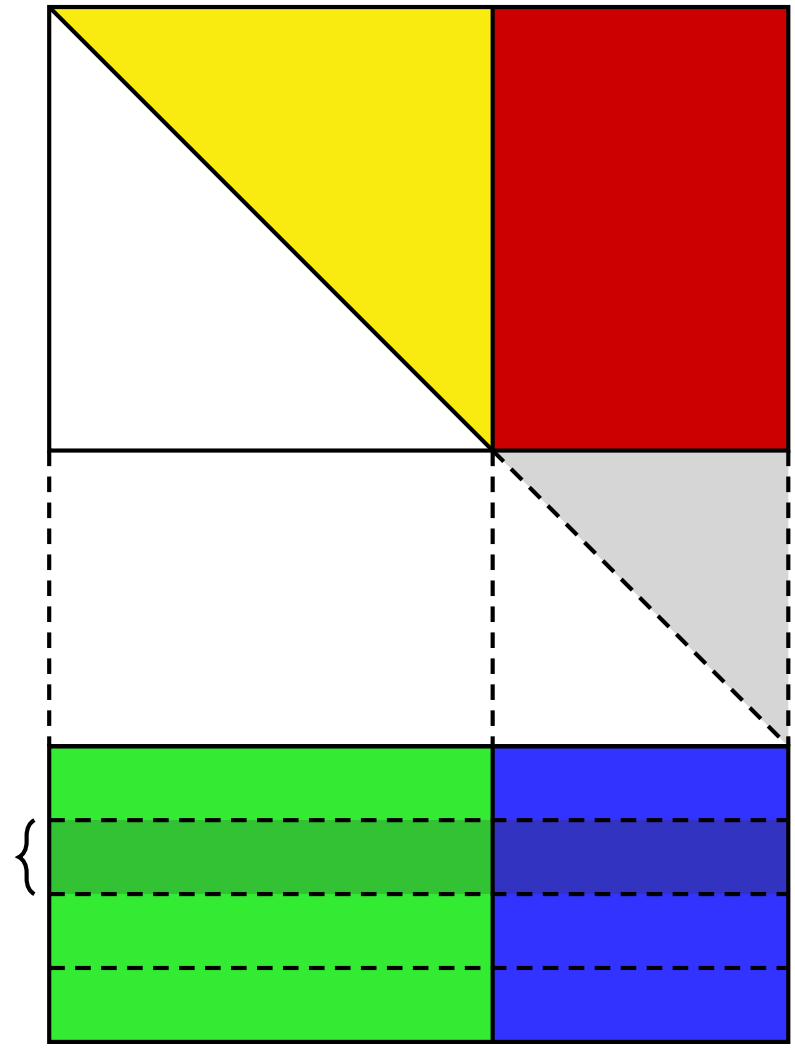




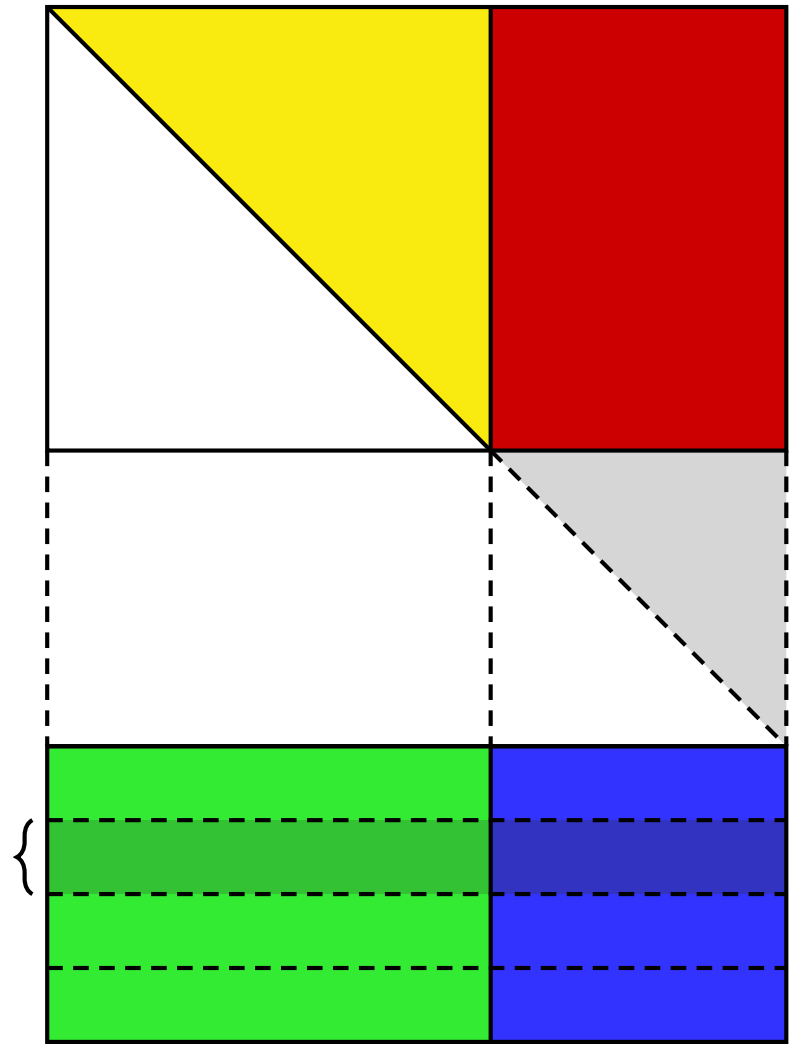


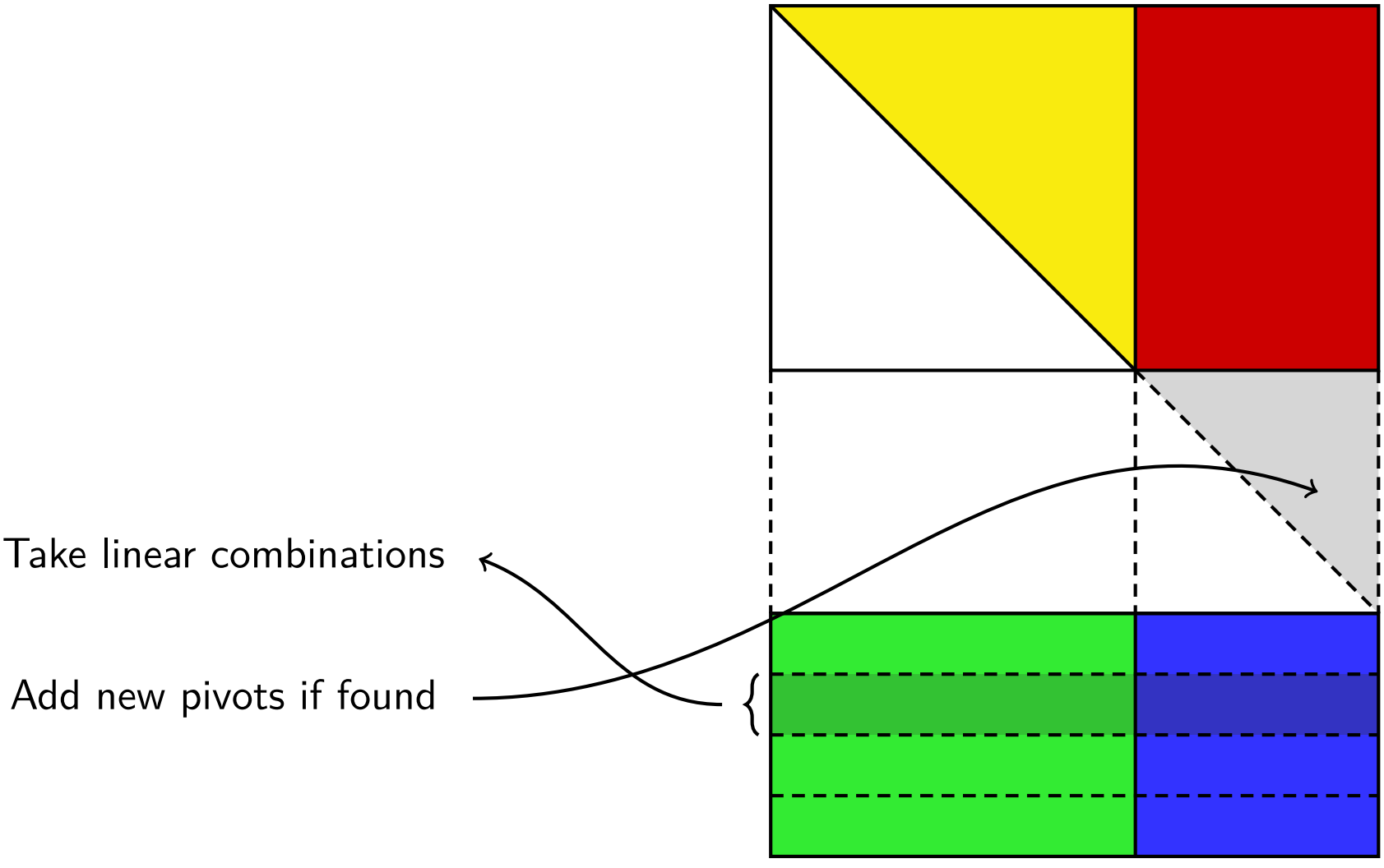
`magma/maple/msolve` **optimization**

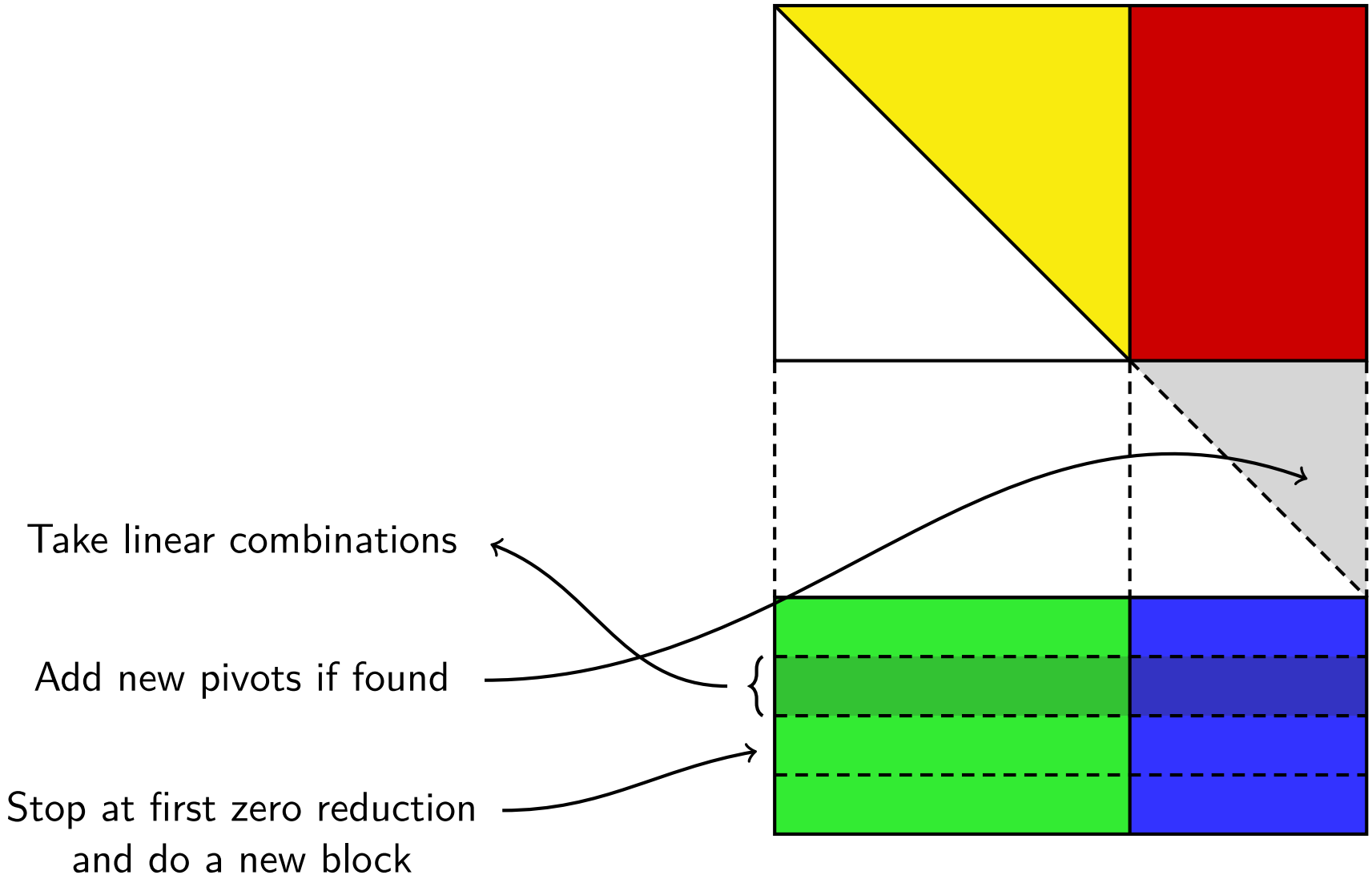
Probabilistic linear algebra over finite fields



Take linear combinations




Take linear combinations
Add new pivots if found


Take linear combinations
Add new pivots if found
Stop at first zero reduction
and do a new block

Compare finite field DRL computations.

Benchmark	msolve		maple		magma	
	F4	FGLM	F4	FGLM	F4	FGLM
phuoc	2.99	3.59	3.71	4.09	3.42	182
katsura-15	1,688	1,460	2,891	2,112	2,088	7,138
katsura-16	13,511	12,041	21,912	15,895	16,838	56,314
cyclic-10	9,108	3,954	16,475	171	8,968	37,927
eco-14	54.3	17.1	65.3	36.5	56.1	105
eco-15	343	172	407	275	346	788
robotics	14,562	39,842	22,152	53,634	10,989	154,602
crit-d4-n9	754	1,297	991	–	505	5,967
crit-d4-n10	17,455	41,787	22,137	–	10,645	153,745
5-conics	1,372	24.1	1,526	35.1	871	964
bethe-15-7	913	1,002	1,115	1,462	1,305	3,406

Timings in seconds (finite field, DRL $\rightarrow$ LEX, single core computation)

Up next:

Multi-modular computations over $\mathbb{Q}$.

General idea

- Run F4 modulo several primes.
- Decide on good prime / bad prime computations.
- Lift results using rational reconstruction.
- Go on until reconstruction stabilizes.

First step can be done in two different approaches:

Approach 1

Run all F4 computations **independently** (using probabilistic linear algebra).

Approach 2

Learn from first F4 computation, **trace** the run, **remove useless** data.

Apply only **optimized** linear algebra in the following runs.

Drawback of tracing approach:

We cannot use probabilistic linear algebra in first run since we need to **recover zero reductions**.

Benefits of tracing approach:

- Way more flexible in terms of reconstruction.
- Concurrent multi-modular computations possible.
- Way less memory consumption after first run.

Experimental results so far.

Examples	`msolve` (tracer)	`msolve` (prob.)	`maple`	`magma`
Katsura-10	37.1	61.8	1,278	82,540
Katsura-11	370	635	7,812	> 72h
Katsura-12	4,817	8,215	120,804	–
Katsura-13	81,295	128,242	> 120h	–
Katsura-14	1,286,602	2,135,227	–	–
Eco-12	831	1,400	4,287	–
Eco-13	11,935	19,032	66,115	–
Eco-14	161,537	248,272	> 240h	–
Noon-7	1,055	1,305	432	> 36h
Noon-8	65,125	69,980	5,997	> 72h
Phuoc-1	8,347	9,092	> 120h	–
Henrion-6	130.585	150.284	1,470.080	–
Henrion-7	124,850.102	139,195.781	> 240h	–
CritPts(3,6,2)	318	411	23,440	> 48h
CritPts(3,7,2)	8,136	10,667	> 240h	–
CritPts(4,5,3)	18,712	21,564	> 240h	–
CritPts(4,6,6)	54,746	72,552	> 240h	–
CritPts(3,8,2)	270,352	322,868	–	–

Single core computations, timings in seconds (if not otherwise stated)

Roadmap

- Finish multi-modular implementation, hunt bugs, analyse efficiency: **right now**.
- Public beta phase: **early December 2020**.
- Open repository: **early 2021**.

Upcoming features for `msolve`

- Julia interface to `OSCAR` (others, too)
- Better parallelization on CPUs
- F5 / signature-based algorithms
- New version of `GBLA`
- Dense linear algebra on GPUs
- You may have some other requests.

Thank you for your attention.

Questions? Remarks?