The Pixel Array method for solving nonlinear systems

David I. Spivak
Joint with Magdalen R.C. Dobson, Sapna Kumari, and Lawrence Wu

dspivak@math.mit.edu
Mathematics Department
Massachusetts Institute of Technology

Presented on 2017/07/13

Outline

Outline

Another look at matrix multiplication

Separately plot the solutions to equations: $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in its own bounding box, say in the range $[-1.5, 1.5]$.
- Consider the plots as matrices M, N whose entries are on/off pixels.
- That is, M and N are *boolean matrices* corresponding to f and g.

Another look at matrix multiplication

Separately plot the solutions to equations: $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in its own bounding box, say in the range $[-1.5, 1.5]$.
- Consider the plots as matrices M, N whose entries are on/off pixels.
- That is, M and N are *boolean matrices* corresponding to f and g.

Multiplying these two matrices MN yields...

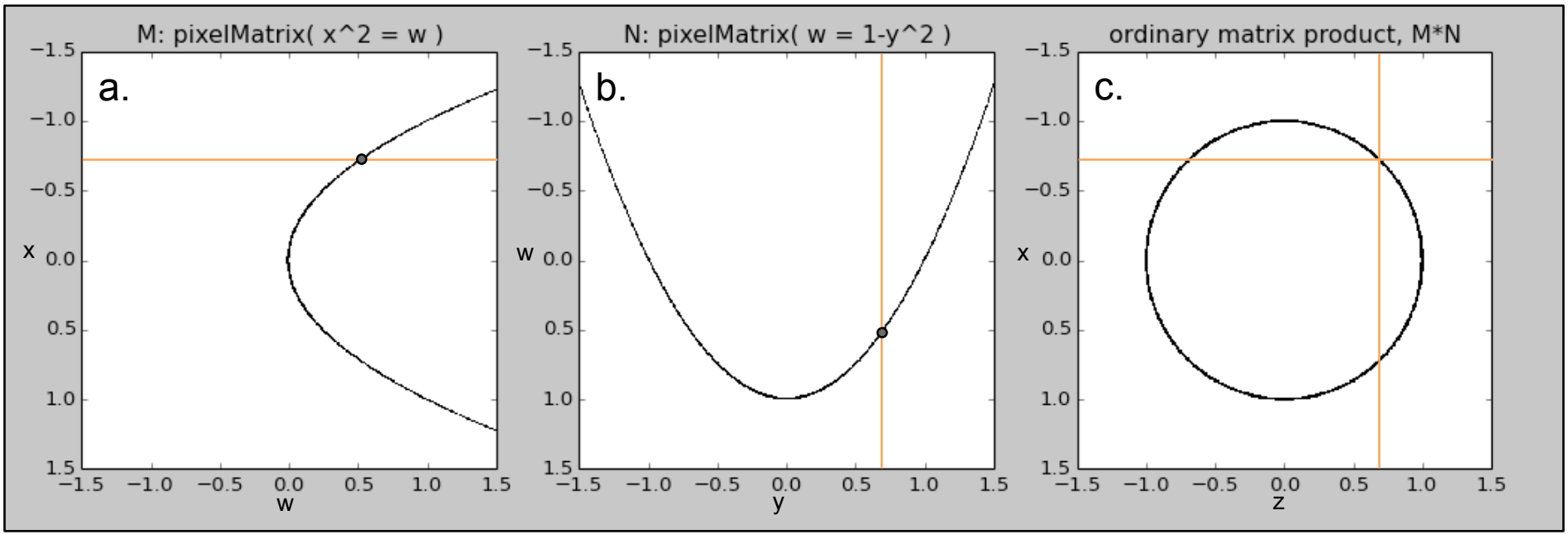
Another look at matrix multiplication

Separately plot the solutions to equations: $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in its own bounding box, say in the range $[-1.5, 1.5]$.
- Consider the plots as matrices M, N whose entries are on/off pixels.
- That is, M and N are *boolean matrices* corresponding to f and g.

Multiplying these two matrices MN yields the simultaneous solution.

- For example, plot equations $x^2 = w$ and $w = 1 - y^2$, and multiply.



A more complex example

The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist? [1]

[1] Spivak, DI; Dobson, MRC; Kumari, S. (2016) "Pixel Arrays: A fast and elementary method for solving nonlinear systems". http://arxiv.org/pdf/1609.00061v1.pdf

A more complex example

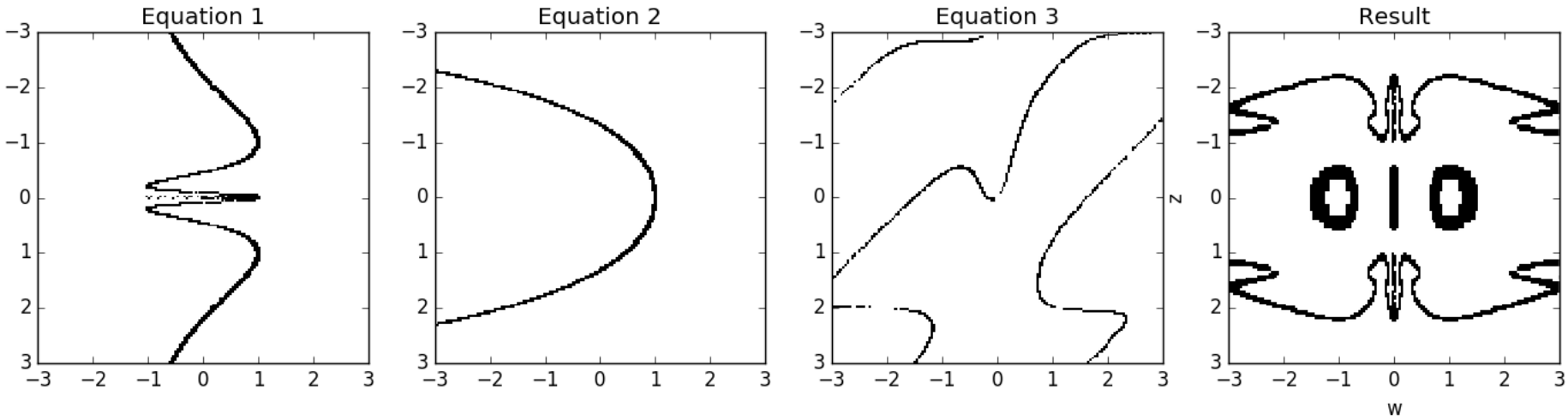
The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist? [1]

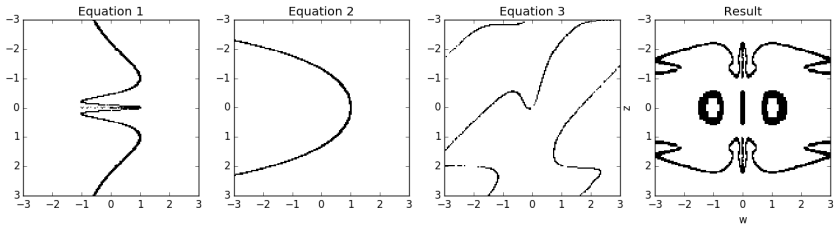


[1] Spivak, DI; Dobson, MRC; Kumari, S. (2016) "Pixel Arrays: A fast and elementary method for solving nonlinear systems". http://arxiv.org/pdf/1609.00061v1.pdf

Selling points

The Pixel Array method has the following features:

- it returns *all solutions in a given bounding box*;
- it's *much faster* than quasi-Newton methods for finding "all solutions" to partially decomposable systems;
- it introduces *no false negatives*;
- it works for *non-differentiable* or even *discontinuous* functions;
- it's *not iterative* and requires no initial guess, in contrast with quasi-Newton methods;
- it's *elementary*—just clustered tensor multiplication—hence has low barrier to entry; and
- it *provides insights*, by showing the whole solution set at once.



So where are the limitations hiding?

Limitation: PA method takes original plots as input.

- The Pixel Array (PA) method requires a plot for each equation.
- The plot of $f(x_1, \ldots, x_n) = 0$ is n-dimensional; can be costly.
  - But there are methods: sampling, zero-crossing, "interval arithmetic".
  - Nice aside: you can even use raw data as your plots.
- PA method is fairly robust, as we saw in butterfly picture.

So where are the limitations hiding?

Limitation: PA method takes original plots as input.

- The Pixel Array (PA) method requires a plot for each equation.
- The plot of $f(x_1, \ldots, x_n) = 0$ is n-dimensional; can be costly.
  - But there are methods: sampling, zero-crossing, "interval arithmetic".
  - Nice aside: you can even use raw data as your plots.
- PA method is fairly robust, as we saw in butterfly picture.

Limitation: PA method introduces false positives.

- Error is roughly bounded by (mesh size) * (max derivative).
- Refining the mesh gives the correct answer in the limit.

So where are the limitations hiding?

Limitation: PA method takes original plots as input.

- The Pixel Array (PA) method requires a plot for each equation.
- The plot of $f(x_1, \ldots, x_n) = 0$ is n-dimensional; can be costly.
  - But there are methods: sampling, zero-crossing, "interval arithmetic".
  - Nice aside: you can even use raw data as your plots.
- PA method is fairly robust, as we saw in butterfly picture.

Limitation: PA method introduces false positives.

- Error is roughly bounded by (mesh size) * (max derivative).
- Refining the mesh gives the correct answer in the limit.

Limitation: its speed requires unexposed variables, partial decomposability.

- I will discuss exposed/unexposed variables a bit more.
- For now, recall butterfly picture: we exposed w and z, but not x or y.
- Speed comparison with quasi-Newton is not straightforward.

Outline

Equations and wiring diagrams

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

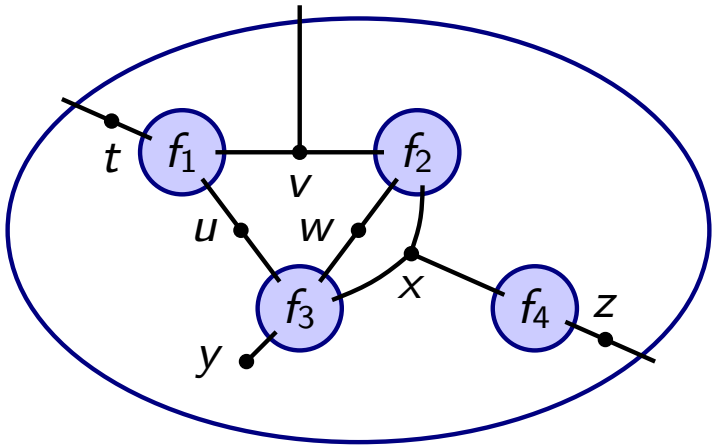
Bold variables are those we want to *expose*; others are latent or *unexposed*.

Equations and wiring diagrams

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

Bold variables are those we want to *expose*; others are latent or *unexposed*.



Said another way, we want $\{(t, v, z) \mid \exists u, w, x, y : f_1 = f_2 = f_3 = f_4 = 0\}$.

Using array multiplication to solve systems

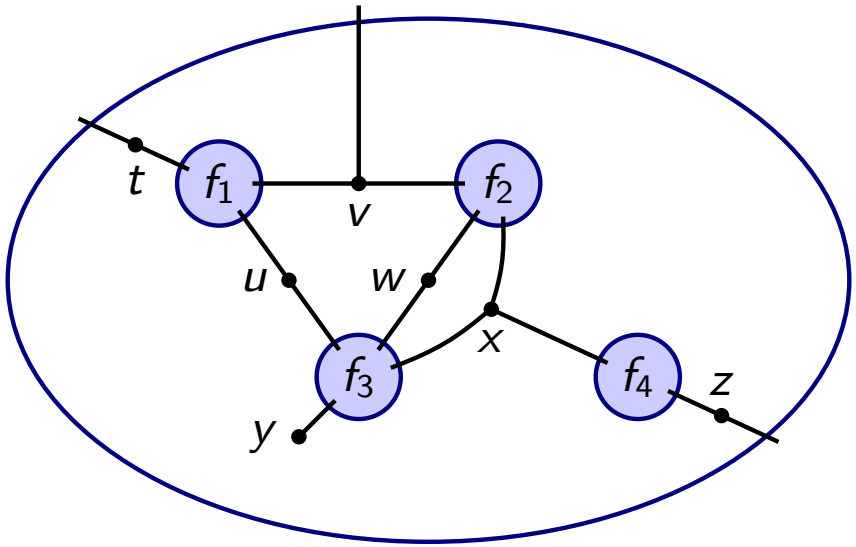
$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

- To solve the system for t, v, z, we plot each equation as an array.

Using array multiplication to solve systems

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

- To solve the system for t, v, z, we plot each equation as an array.
- PA says: multiply the arrays according to "variable sharing" diagram:



Partial decomposability: not every equation shares every variable

Why and how

There are now two important directions to go from here:

- Explain general array multiplication as dictated by a wiring diagram.
- Demystify relationship between multiplying arrays and solving systems.

Let's start by demystifying why the Pixel Array method works.

- As we'll see, it just comes down to logic: AND, OR, TRUE, FALSE.

To keep things simple, let's restrict our attention to matrix multiplication.

Multiplying Boolean matrices

The matrix multiplication formula works well in any semiring.

$$(MN)_{i,k} = \sum_j M_{i,j} * N_{j,k}.$$

- Roughly, a semiring is a set with $0, 1, +, *$ that act reasonably.
- It's like a ring, but you don't need negatives (e.g. $\mathbb{N}$).

Multiplying Boolean matrices

The matrix multiplication formula works well in any semiring.

$$(MN)_{i,k} = \sum_j M_{i,j} * N_{j,k}.$$

- Roughly, a semiring is a set with $0, 1, +, *$ that act reasonably.
- It's like a ring, but you don't need negatives (e.g. $\mathbb{N}$).

Today we'll focus on the Boolean semiring, $\mathbb{B}$.

- It has two elements $\mathbb{B} = \{0, 1\}$.
- 0 means FALSE, and 1 means TRUE.
- Multiplication is given by boolean AND (denoted $\wedge$).
- Addition is given by boolean OR (denoted $\vee$).
- The only slightly unexpected thing is that $1 + 1 = 1$.

The logic of matrix multiplication

In general, the PA method is to multiply boolean tensors (arrays).

- But for now, let's suppose A, B are boolean matrices.
- Say that A and B are plots of $f(x, y) = 0$ and $g(y, z) = 0$, resp.
- So $A_{i,j} = 1$ means $f(x, y) = 0$ in corresponding pixel (else $A_{i,j} = 0$).

The logic of matrix multiplication

In general, the PA method is to multiply boolean tensors (arrays).

- But for now, let's suppose A, B are boolean matrices.
- Say that A and B are plots of $f(x, y) = 0$ and $g(y, z) = 0$, resp.
- So $A_{i,j} = 1$ means $f(x, y) = 0$ in corresponding pixel (else $A_{i,j} = 0$).
- The (i, k)-entry of their product AB is given by the formula:

$$\begin{aligned}(AB)_{i,k} &= \sum_j A_{i,j} * B_{j,k} \\ &= \bigvee_j A_{i,j} \wedge B_{j,k} \\ &= \exists_j \left(A_{i,j} \wedge B_{j,k}\right)\end{aligned}$$

The logic of matrix multiplication

In general, the PA method is to multiply boolean tensors (arrays).

- But for now, let's suppose A, B are boolean matrices.
- Say that A and B are plots of $f(x, y) = 0$ and $g(y, z) = 0$, resp.
- So $A_{i,j} = 1$ means $f(x, y) = 0$ in corresponding pixel (else $A_{i,j} = 0$).
- The (i, k)-entry of their product AB is given by the formula:

$$\begin{aligned}(AB)_{i,k} &= \sum_j A_{i,j} * B_{j,k} \\ &= \bigvee_j A_{i,j} \wedge B_{j,k} \\ &= \exists_j \left(A_{i,j} \wedge B_{j,k}\right)\end{aligned}$$

- " There exists some y such that $f(x, y) = 0$ and $g(y, z) = 0$. "

Multiplying larger-order arrays

When two arrays share a common dimension, they can be multiplied.

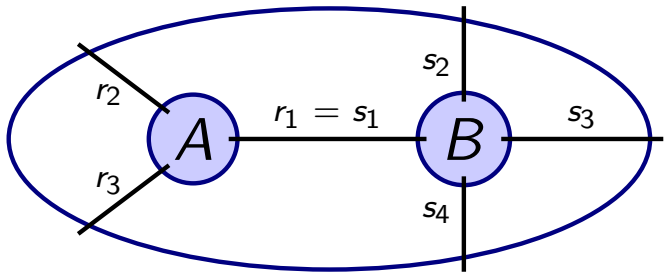
- For example, say that A is $r_1 \times \cdots \times r_m$ and that B is $s_1 \times \cdots \times s_n$.
- If $r_1 = s_1$, the product is an $r_2 \times \cdots \times r_m \times s_2 \times \cdots \times s_n$ array.

Multiplying larger-order arrays

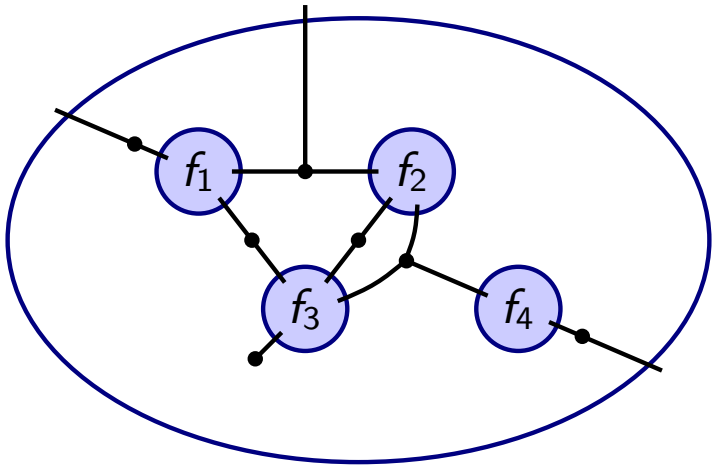
When two arrays share a common dimension, they can be multiplied.

- For example, say that A is $r_1 \times \cdots \times r_m$ and that B is $s_1 \times \cdots \times s_n$.
- If $r_1 = s_1$, the product is an $r_2 \times \cdots \times r_m \times s_2 \times \cdots \times s_n$ array.

We will be drawing these situations using wiring diagrams.



The general array multiplication algorithm



A single formula exists to multiply arrays according to any wiring diagram.

- Basically: iterate over all the links, multiply array entries, and sum.

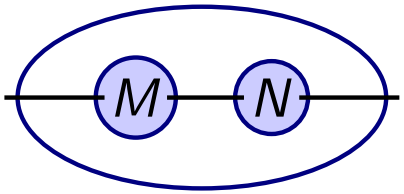
$$\text{Result}(\text{exposed\_vars}) = \sum_{\text{unexposed\_vars}} \prod_{\text{tensors}} \text{tensor}(\text{relevant\_vars})$$

- But this is very naive: $O(\text{resolution}^{\#\text{vars}})$.
  - Plots of equations are sparse, boolean matrices (entries are bits, 0,1).
  - Both of sparsity and booleanness can speed up matrix multiplication.
- But to be efficient, we will need to "cluster" the multiplication.

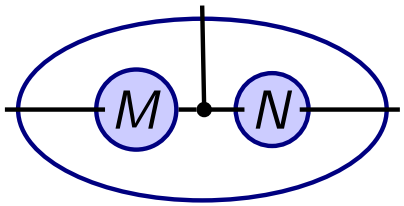
Example wiring diagrams for named operations

The same general array multiplication formula[2] returns famous matrix products for the following wiring diagrams:

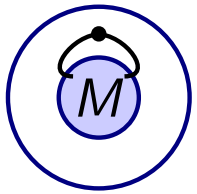
Multiplication: MN



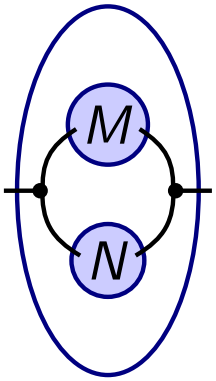
Khatri-Rao: $M \odot N$



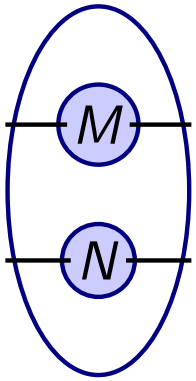
Trace: $\mathrm{Tr}(M)$



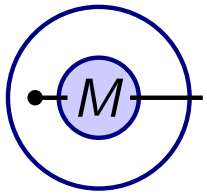
Hadamard: $M \circ N$



Kronecker: $M \otimes N$



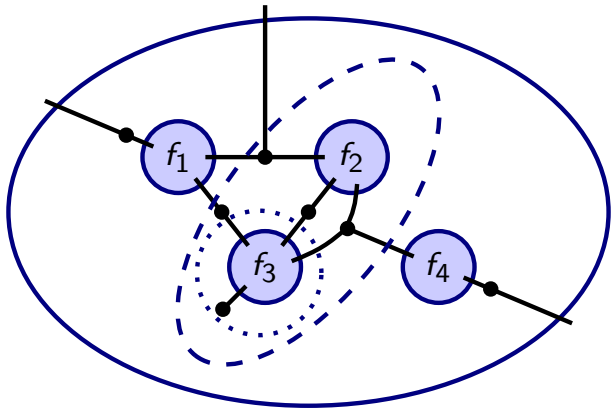
Marginalize: $\sum_i M_{i,j}$



[2] $\text{Result}(\text{exposed\_vars}) = \sum_{\text{unexposed\_vars}} \prod_{\text{tensors}} \text{tensor}(\text{relevant\_vars})$

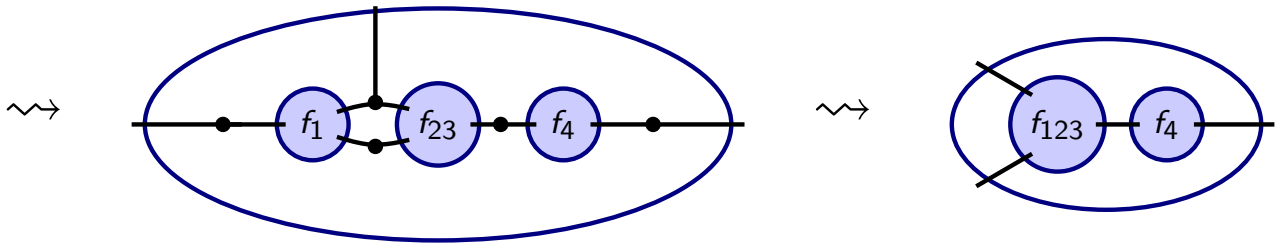
Clustering for speed

Order of array multiplication doesn't affect solution, but does affect speed.



Naive cost for multiplying arrays: $O(\text{resolution}^{\#\text{vars}})$

- Even if we can use sparsity, etc. solving big systems is expensive.
- Instead, we should use the associative law: cluster.



- Naive cost goes from $O(\text{resolution}^7)$ to $O(\text{resolution}^4)$.

Outline

Apples and oranges

The inputs and outputs of the PA method are different than Newton's.

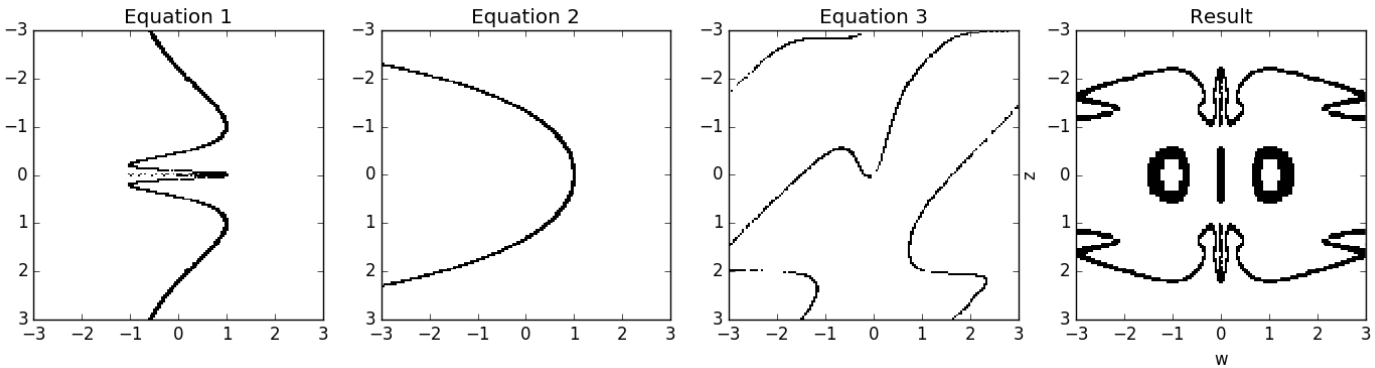
	Pixel Array method	**Newton's method**
Inputs:	a range for each variable	a good initial guess
Outputs:	all solutions for some variables	one solution in all variables.

Apples and oranges

The inputs and outputs of the PA method are different than Newton's.

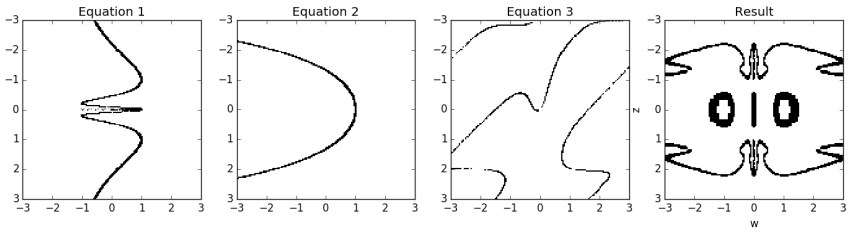
	Pixel Array method	**Newton's method**
Inputs:	a range for each variable	a good initial guess
Outputs:	all solutions for some variables	one solution in all variables.

Our speed test assumes you want to produce "all solutions" in range.



How might we use Newton to find all solutions??

How to compare speed?



Our comparison with Newton assumes you want "all solutions" in range.

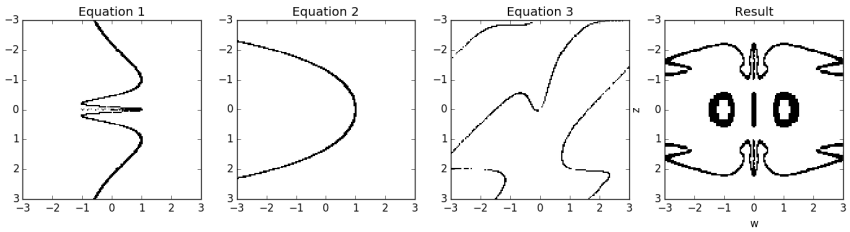
- We iterated over all boxes in the grid, each as an initial guess.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

How to compare speed?



Our comparison with Newton assumes you want "all solutions" in range.

- We iterated over all boxes in the grid, each as an initial guess.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

For the above case, PA was over 7200x faster in finding all solutions.

- PA: 1.5 seconds; Newton: we stopped Julia's NLsolve after 3 hours.
- PA was faster in every partially-decomposable system we tried.

Outline

Summary

In this talk, I discussed the Pixel Array method for solving systems.

- We have an array (tensor) for each equation.
- We know which variables are shared (captured by a wiring diagram).
- We then cluster the arrays to minimize the cost of multiplying.
- Multiplying the arrays gives the simultaneous solution set.

The PA method is quite fast if you want to find all solutions to a partially-decomposable system.

Summary

In this talk, I discussed the Pixel Array method for solving systems.

- We have an array (tensor) for each equation.
- We know which variables are shared (captured by a wiring diagram).
- We then cluster the arrays to minimize the cost of multiplying.
- Multiplying the arrays gives the simultaneous solution set.

The PA method is quite fast if you want to find all solutions to a partially-decomposable system.

Thanks for your time;
Questions and comments welcome!