Randomly Surreal (Numbers)

Prof Matthew Roughan, FIEEE, FACM
Director TRC, Uni Adelaide
<matthew.roughan@adelaide.edu.au>

THE UNIVERSITY of ADELAIDE

TELETRAFFIC RESEARCH CENTRE

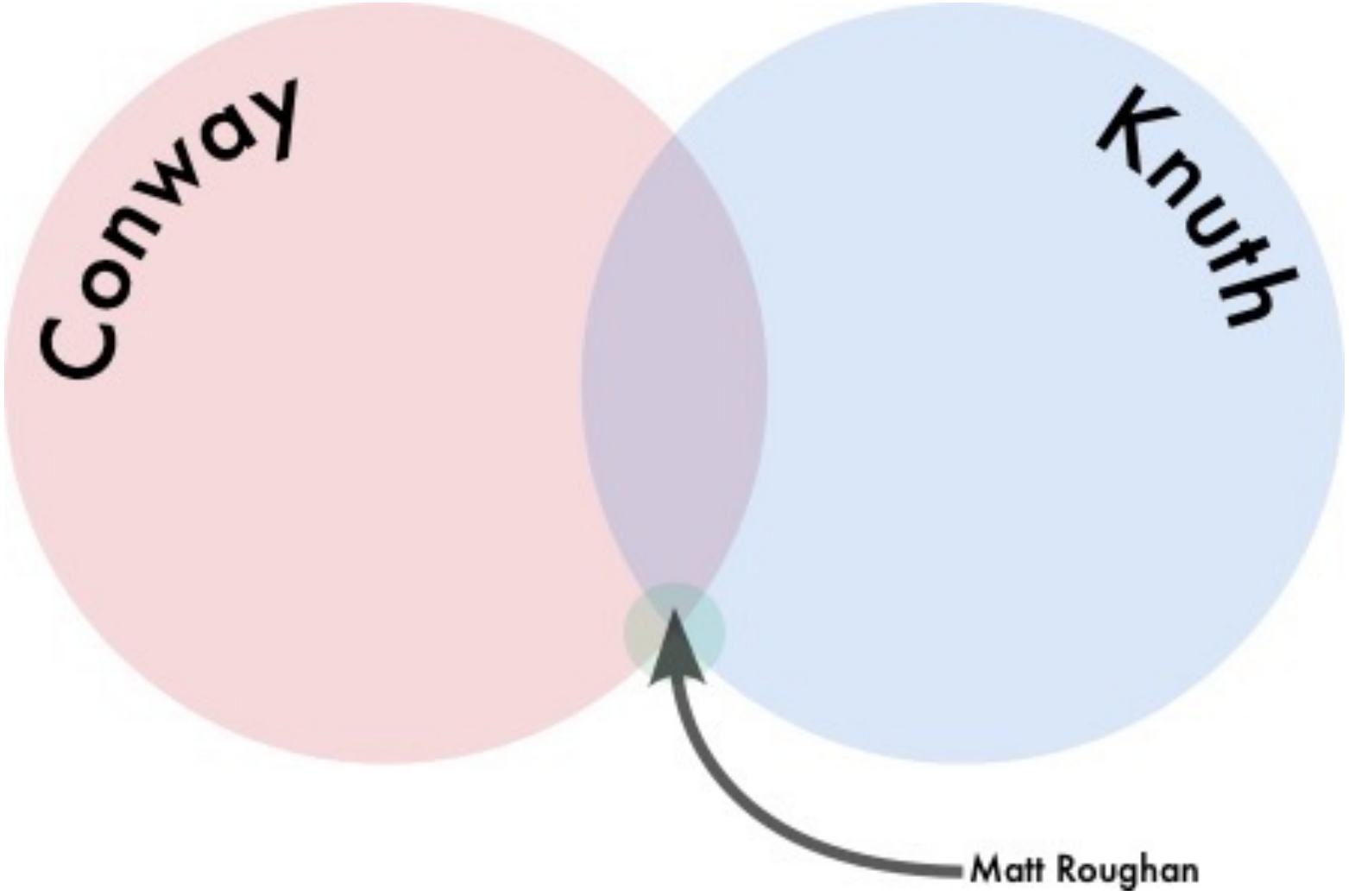
Why?

Julia is the cool new kid on the programming block

- It’s fast
- It’s clean

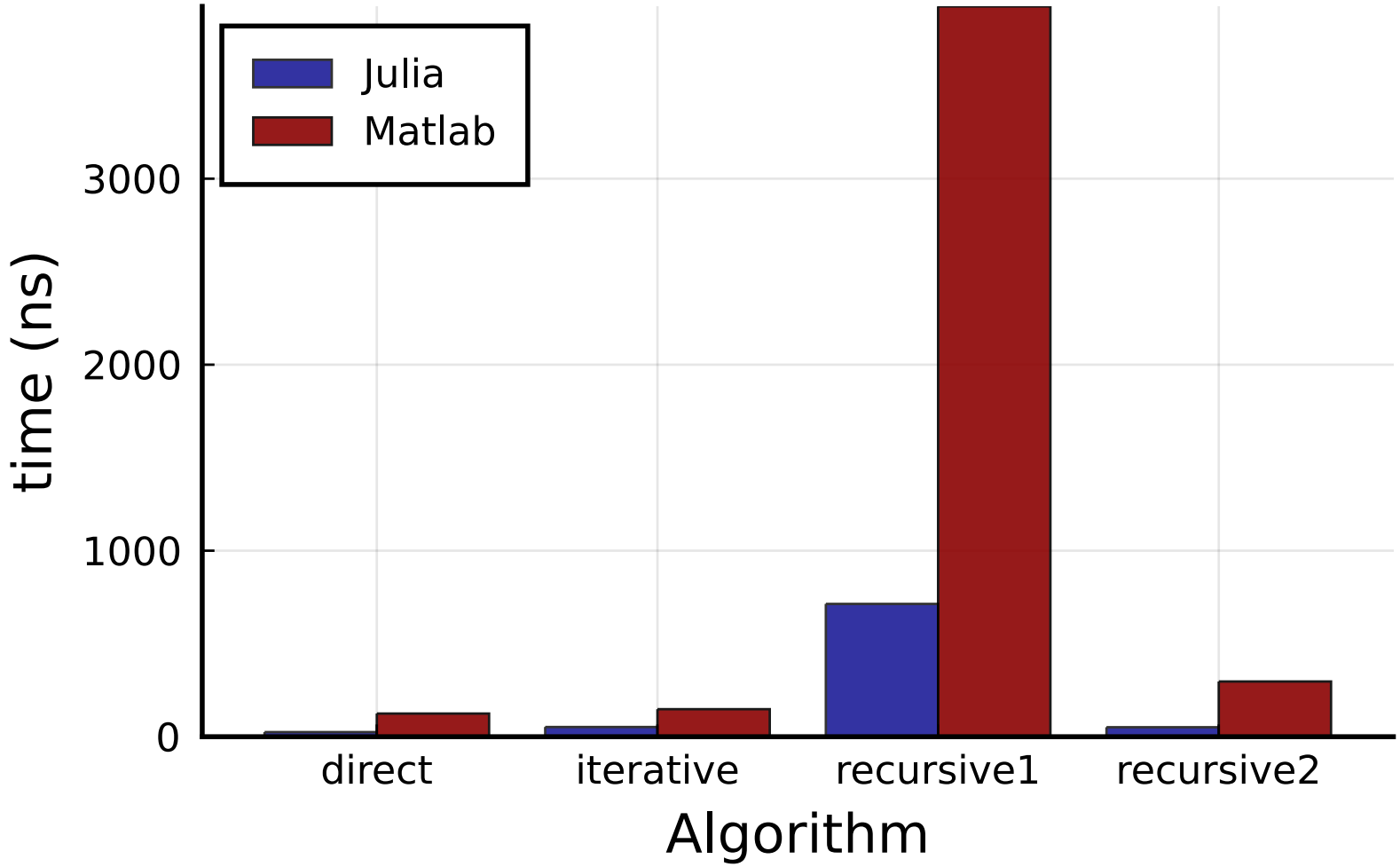
So I should know it, but you need to do, to know

- I wanted a task
- Shouldn’t be easy
- Should make a contribution


Conway
Knuth
Matt Roughan

Recursion

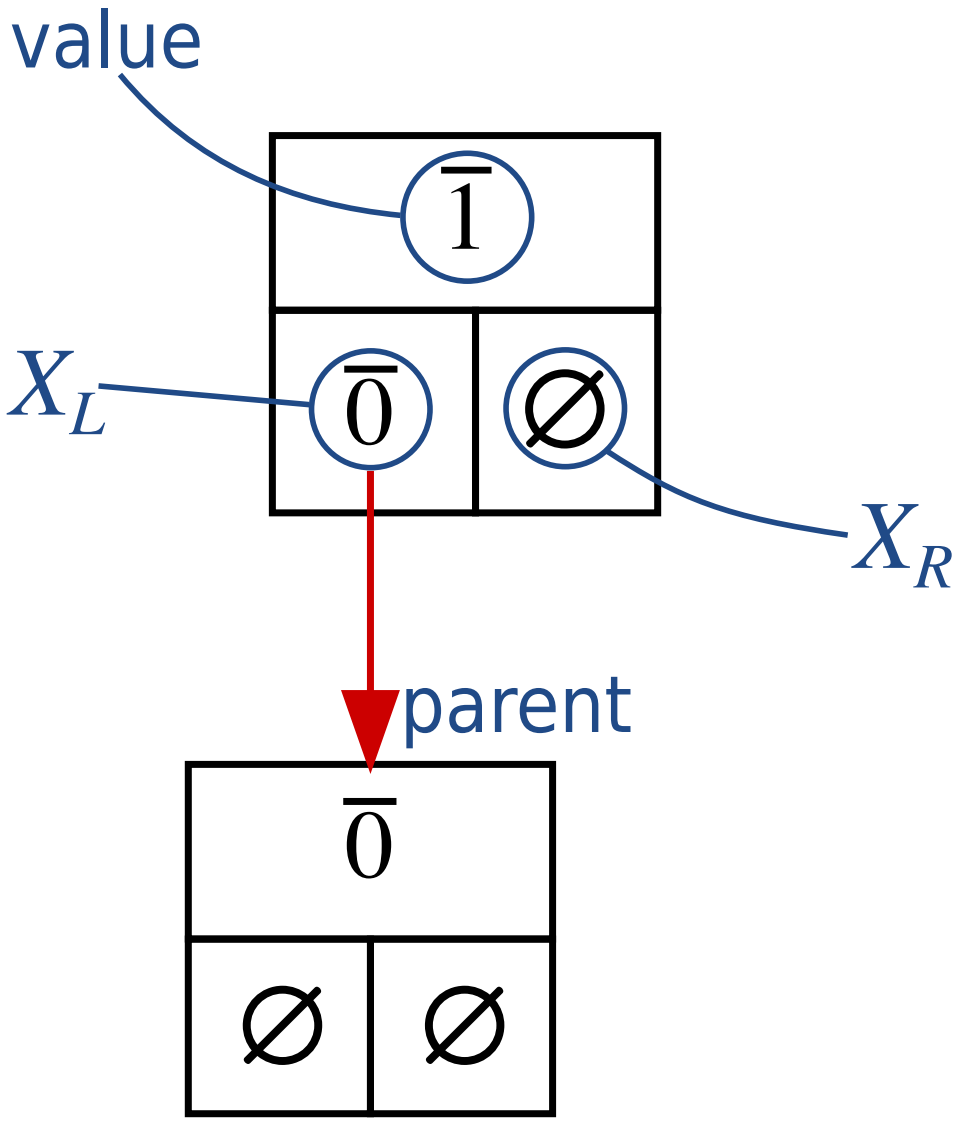
I wanted a job that would be painful in Matlab



Surreal Numbers

A surreal number x is

- A left and right set of surreal numbers $\{X_L \mid X_R\}$
- Such that no element of X_L is ≥ any element of X_R



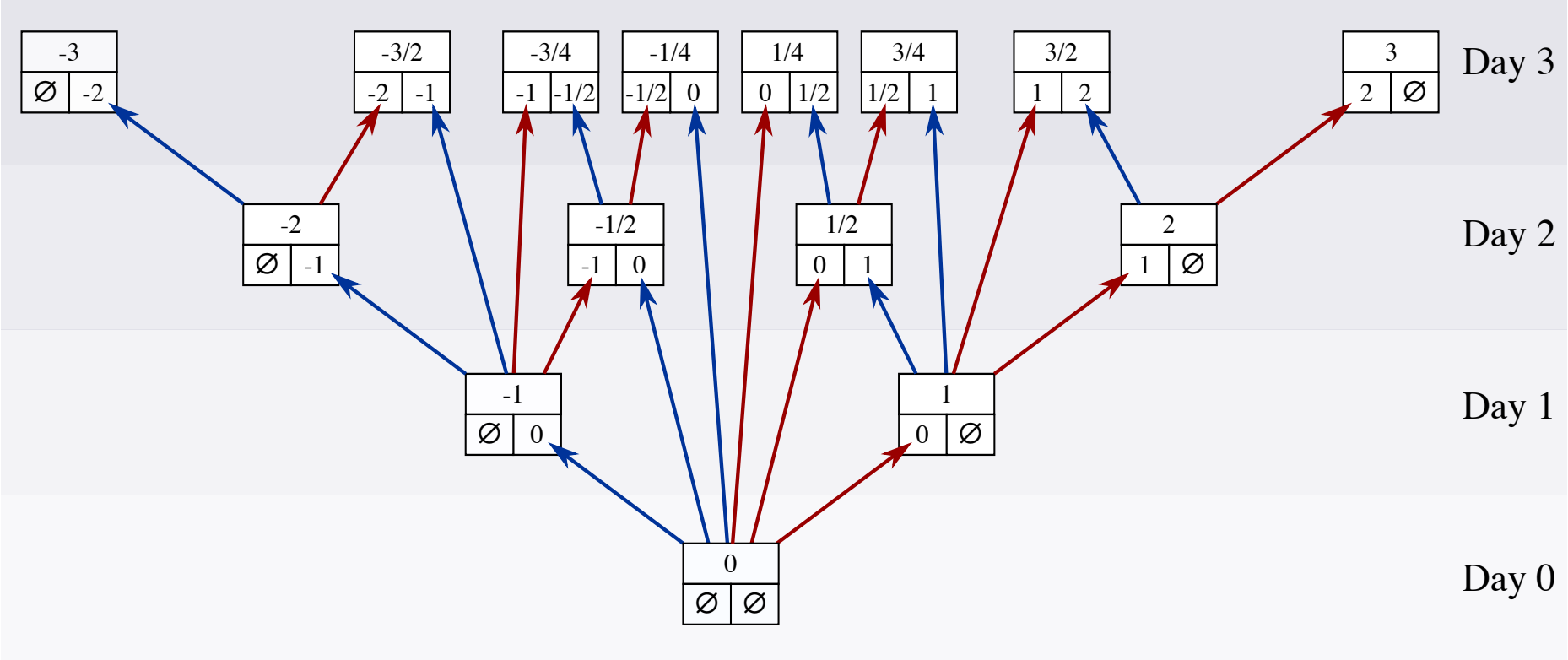
We can represent a surreal number as a **DAG**

Directed
Acyclic
Graph

With left and right labels on the edges.

Note that empty sets are OK, so $0 = \{ \mid \}$ is always the root

The dyadic canonicals



Every dyadic $k/2^n$ can be constructed using the “dyadic tree”

I am only working with dyadic, but …

There are multiple constructions for numbers, but I call this canonical and indicate by and overling

But They Get Complicated Real Fast

Addition

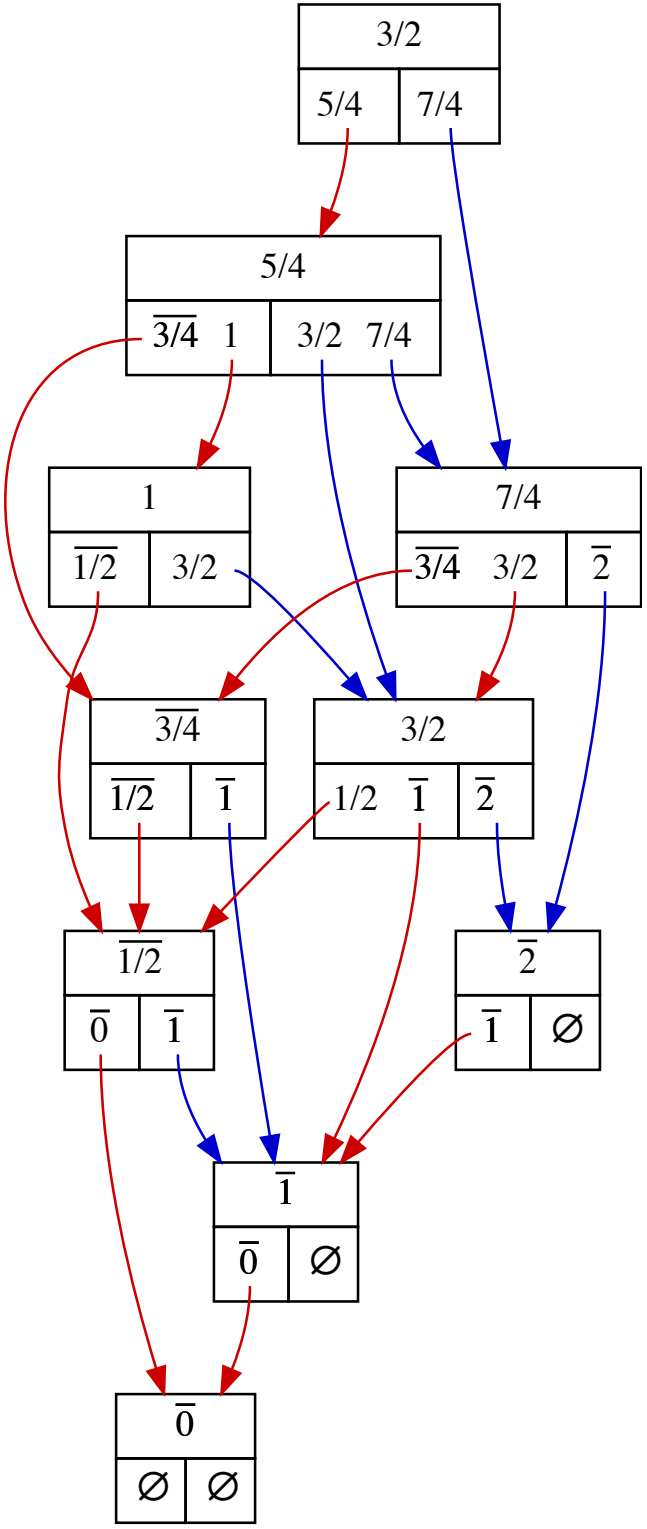
$$x + y = \{X_L + y \cup x + Y_L \mid X_R + y \cup x + Y\_R\}$$

So

1 + ½ =

{{{∅ | ∅} | {{∅ | ∅} | ∅}}, {{∅ | ∅} | ∅} | {{{∅ | ∅} | ∅} | ∅}}

And $\frac{3}{4} + \frac{3}{4} \Rightarrow$



So I Wrote a Package

`https://github.com/mroughan/SurrealNumbers.jl`

But how do you check a package?

- **There isn't an existing one to compare to**
- **The existing examples are either**
  - Too simple
  - Too complex*
- **Goal:** Create random ensemble with controlled complexity

* = complexity ≈ generation

Synthesis Algorithm (abbrev.)

for i = 1 to m do ◂ create a new clade C_i

  for j = 1 to n do ◂ create a new surreal $x_{\{ij\}}$

    Generate a number of parents $n_p \sim D_p(\lambda)$

    Select a set of n_p surreals from $C_{\{i-1\}}$

      according to the weighting w(g(x))

    Sort the parents into a list P

    Choose a split point $s \sim D_s(n_p)$

    $X_L \leftarrow \{ P[1, \ldots, s] \}$

    $X_R \leftarrow \{ P[s+1, \ldots, n_p] \}$

    $x_{\{ij\}} \leftarrow \{ X_L | X_R \}$

  end for

end for

Generation Distribution

Naively you might expect generation to grow with l

- Injecting more “depth” into each generation

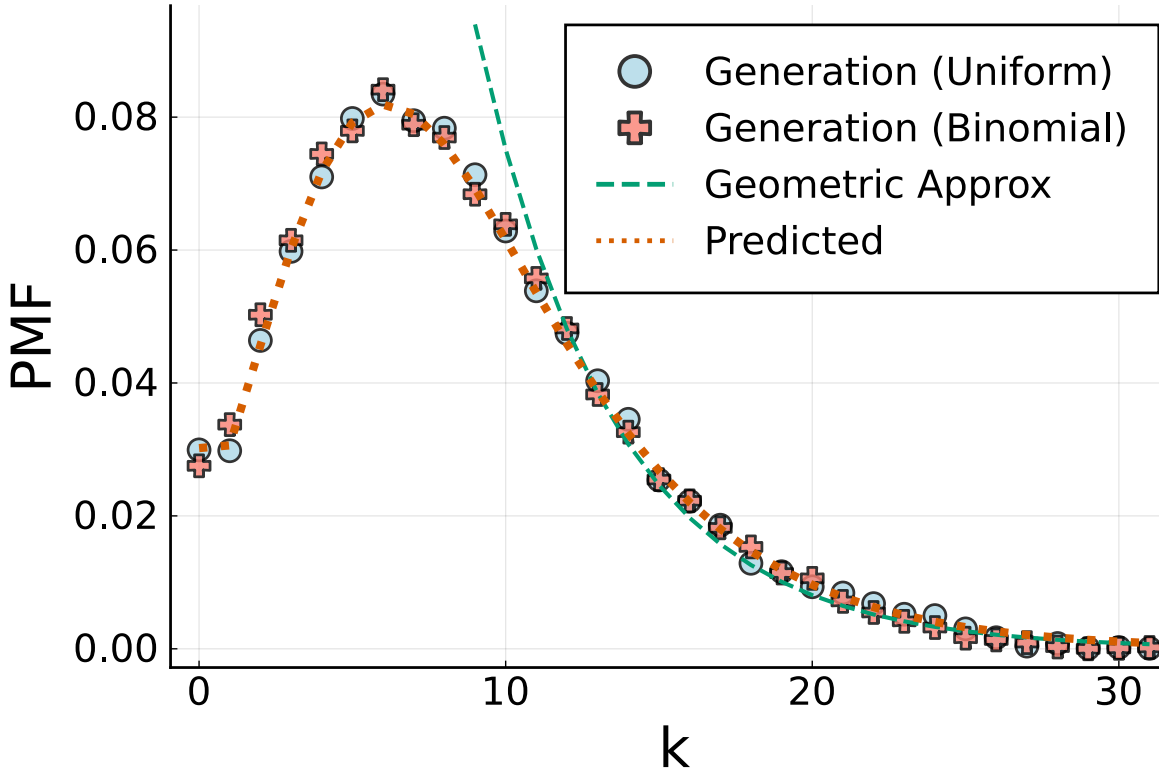
With weighting distribution $w(g) \propto \alpha^g$ there is a push back

We can show

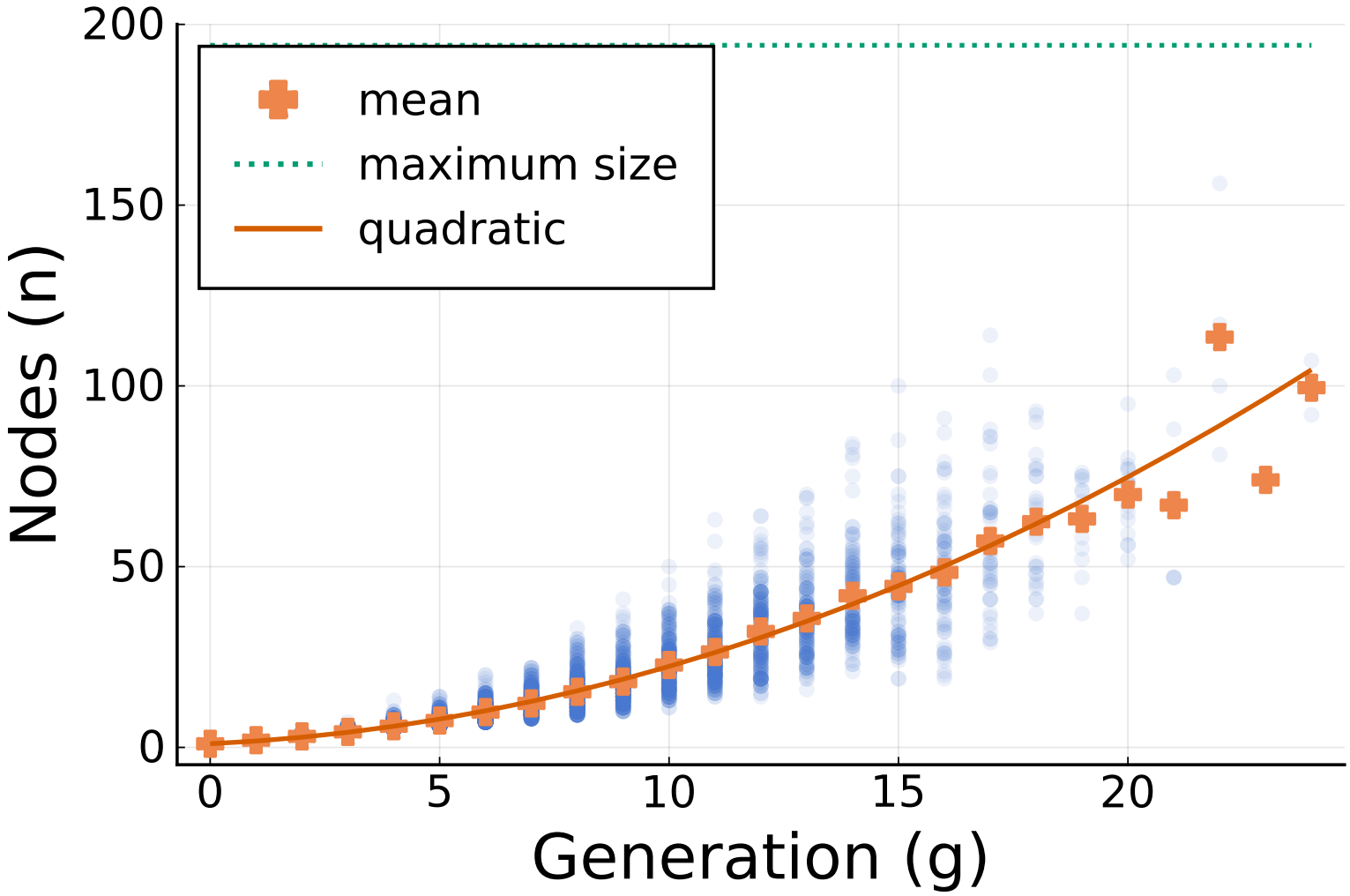
$$P\{ g(x) \leq k\} = e^{-\lambda \alpha^k}$$

- **Has a geometric tail**
- **Expectation**

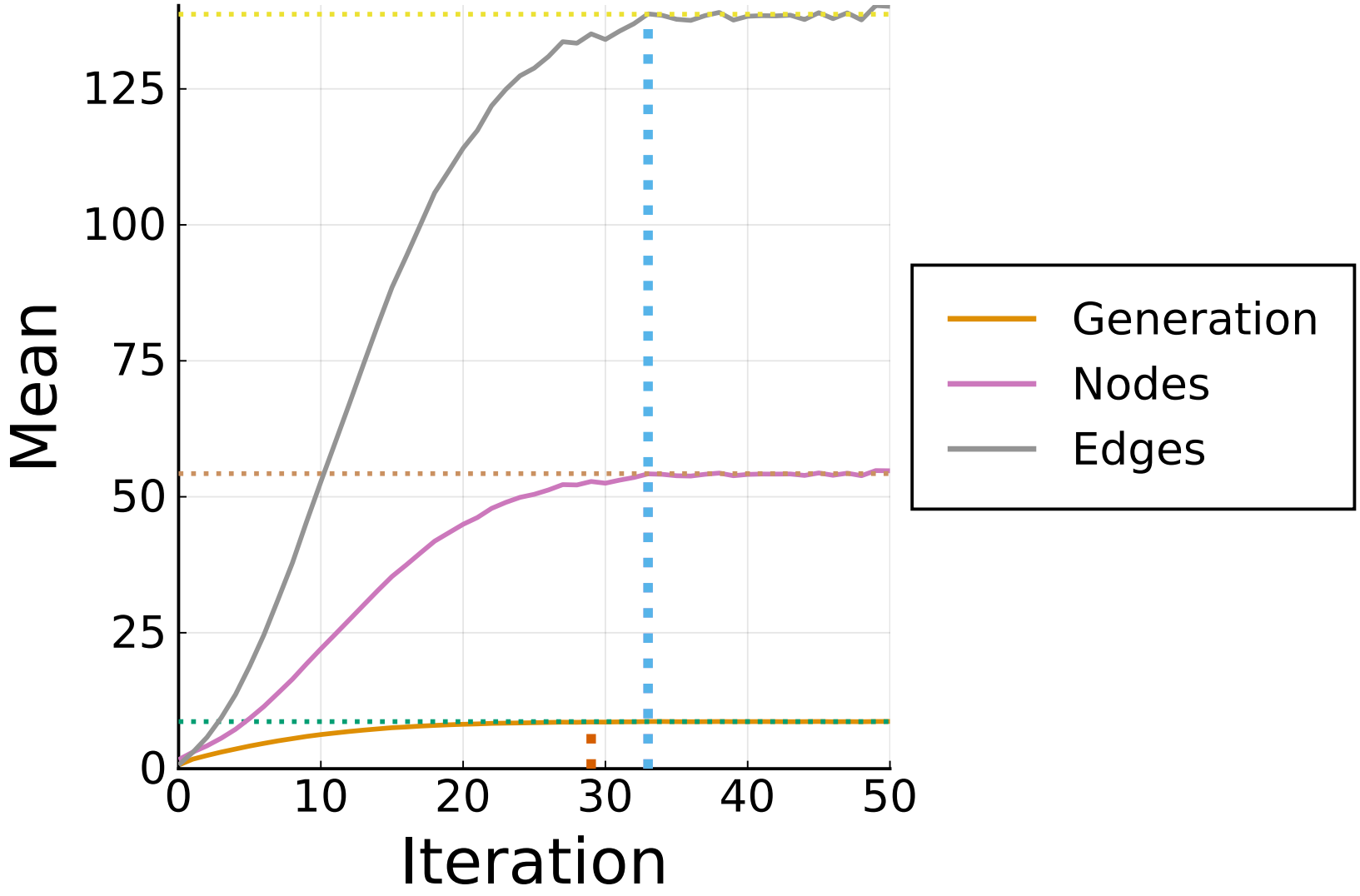
$$\approx \frac{1}{\log \alpha}(\log(\lambda) - E_1(\lambda) + \gamma)$$



Node-size


200
150
100
50
0
Nodes (n)
mean
maximum size
quadratic
0
5
10
15
20
Generation (g)

Convergence (in a weak sense)



https://github.com/mroughan/SurrealNumbers.jl

Conclusion

We did it to debug

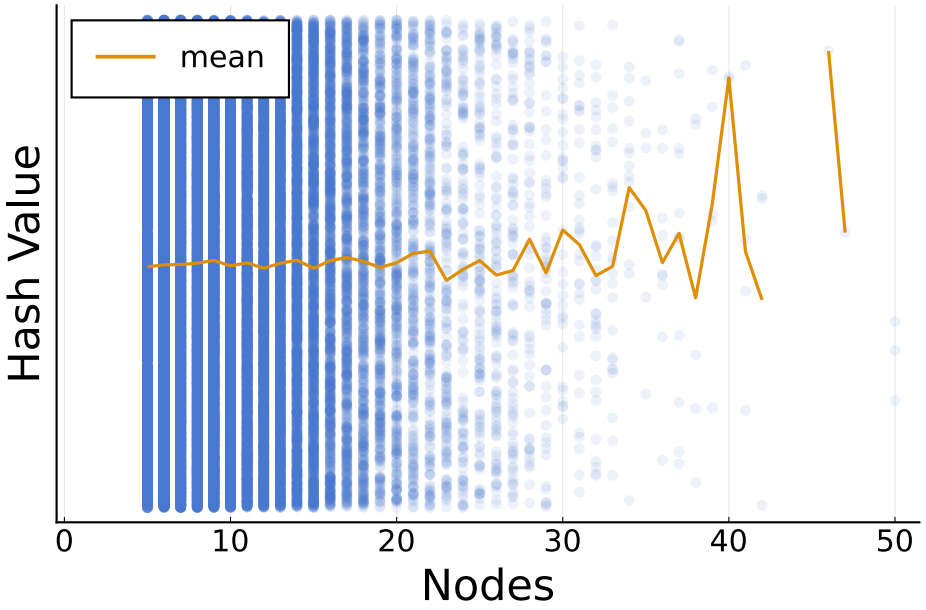
- Found 1 bug (1 in a 10 billion cases triggered the bug)
- Hash function in package has a bad feature (will fix)

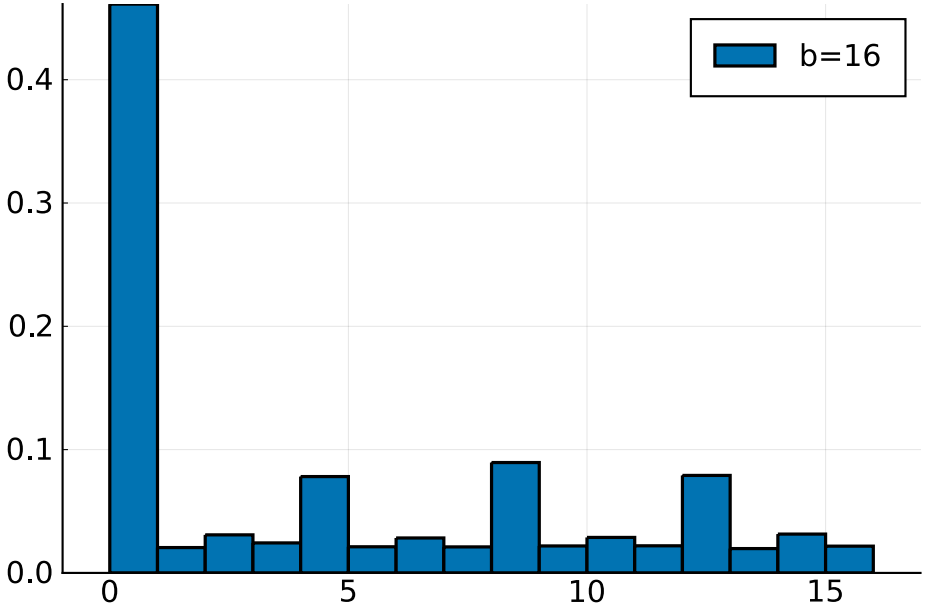
What next

- Seems interesting
  - Generalize
  - More maths to come, e.g., distributional properties
  - Maybe useful in generating hypotheses about the surreals

Extra Result

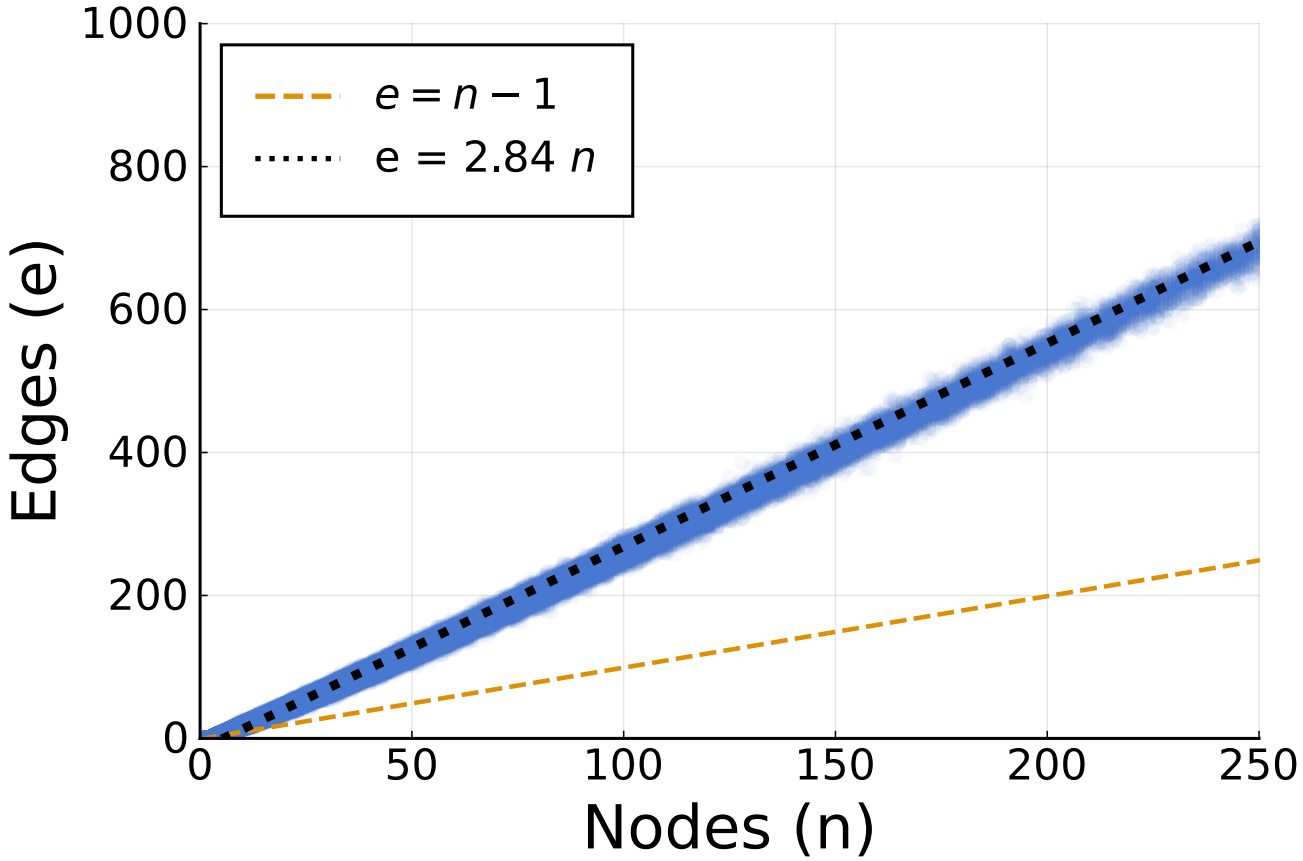
Hashes





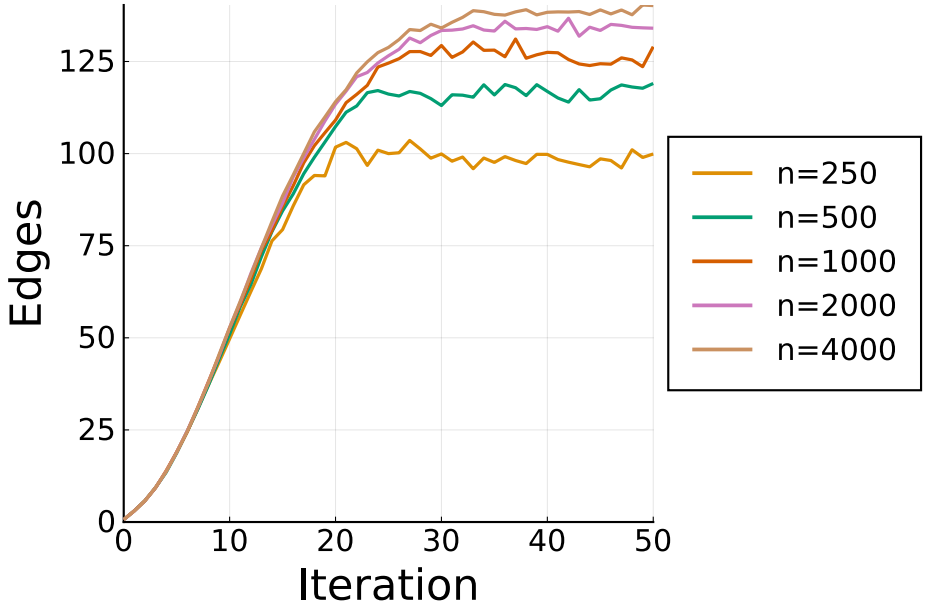
Extra Result

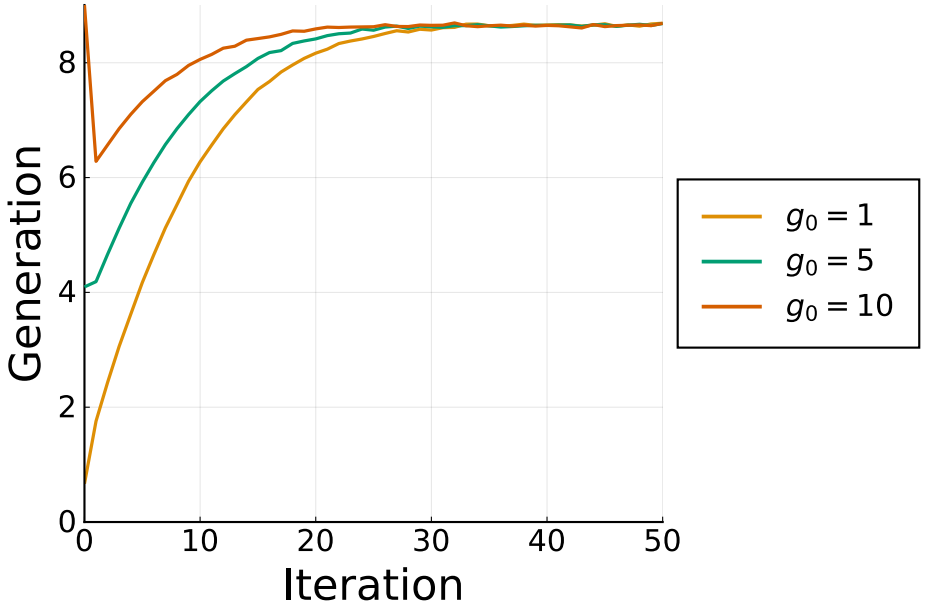
Edges



Extra Result

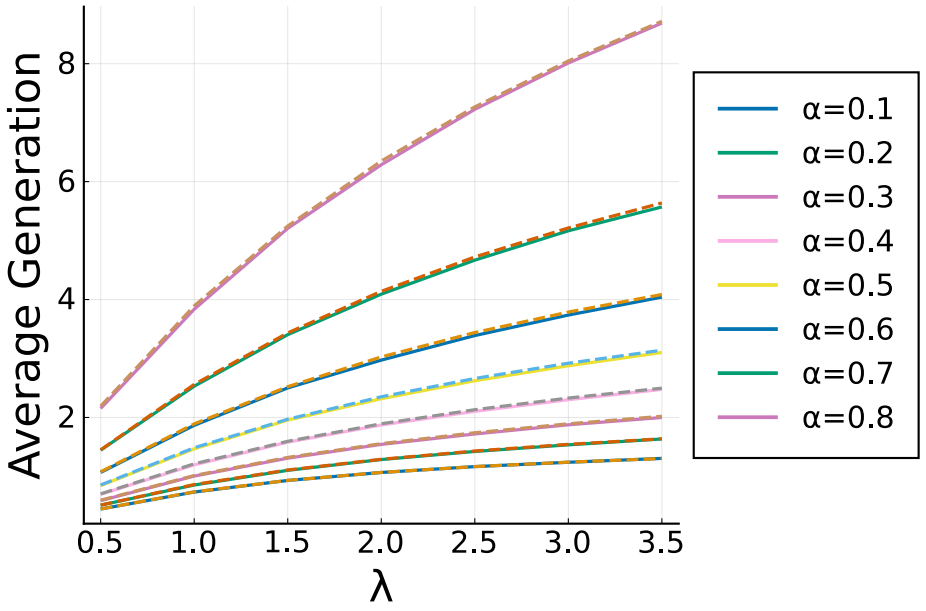
Convergence (n=population size, g_0= starting max gen)

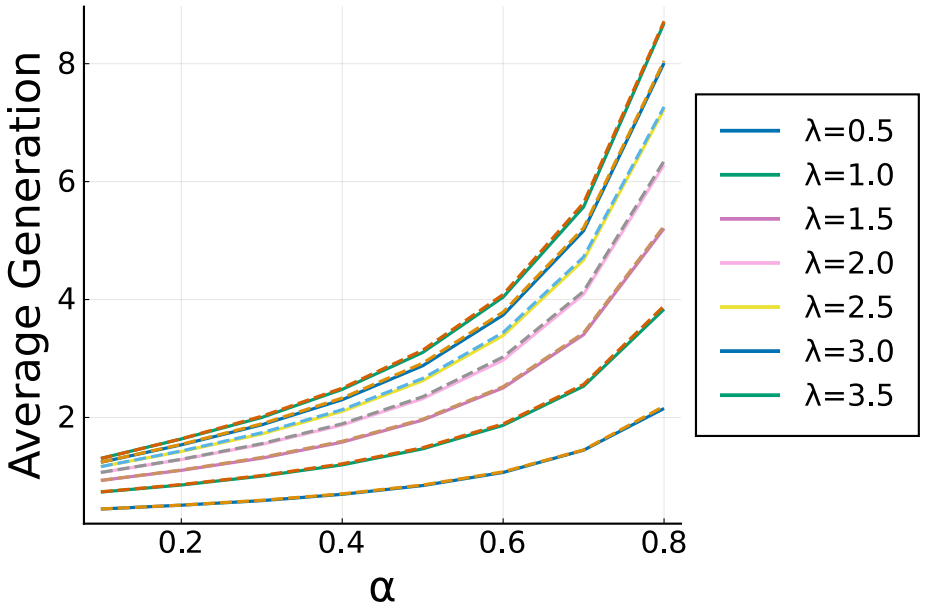




Extra Results

Expectation





Extra Results

Convergence time