Subjective Exchangeable Partition Priors via Integer Partitions

ICERM Workshop on Nonparametric Bayesian Inference – Computational Issues

David B. Dahl Richard L. Warr Talmage Hilton

Department of Statistics
Brigham Young University

January 13, 2026

BYU

"OUTSTANDING . . . FUN TO READ."
–THE WALL STREET JOURNAL

COMPLETELY REVISED AND UPDATED

New Ideas From Dead Economists

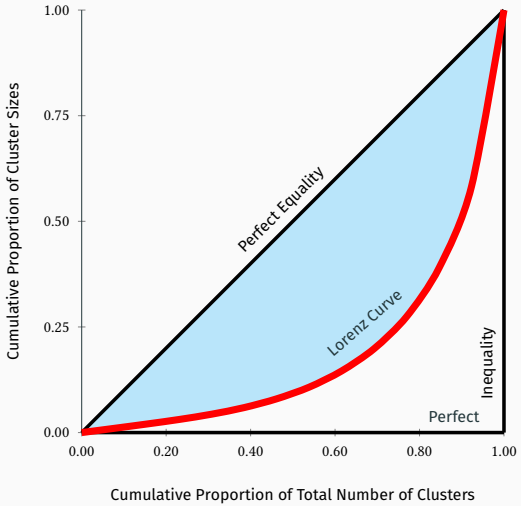
The Introduction to Modern Economic Thought, 4th Edition

Todd G. Buchholz

AUTHOR OF *NEW IDEAS FROM DEAD CEOs*

Roadmap

- Motivation: Only Indirect Control
- Desiderata for Our Partition Prior
- Hierarchical Construction
- Lorenz Integer Partition (Lorenz-IP) Distribution
- Theory
- Exchangeability
- Computation
- Conclusion and Open Questions



Motivation: Only Indirect Control

Prior beliefs about the number of clusters and cluster imbalance

- Bayesian mixture modeling revolves around concerns about the clustering prior:
  - What is the distribution of the number of clusters?
  - Is *this* or *that* prior consistent for the number of clusters?
  - Are we comfortable with the "rich get richer" property?
  - How do we deal with all of these singleton clusters?
- In practice, we often have separate intuition about "how many" clusters there are and "how imbalanced" the clusters are.
- Standard BNP priors entangle these two aspects through a small set of global parameters.
  - E.g., Pitman-Yor has only univariate concentration and discount parameters.
  - Mixtures of finite mixtures (MFMs) allow a prior on the number of clusters, but size profiles are still indirect.

Pitman-Yor / CRP2 code

```
> n_items <- 100
> concentration <- 1.0
> discount <- 0.1
>
> crp <- pumpkin::CRPPartition(n_items, concentration, discount)
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
```

Pitman-Yor / CRP2 code

```
> concentration <- 1.0
> discount <- 0.1
>
> crp <- pumpkin::CRPPartition(n_items, concentration, discount)
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
```

Pitman-Yor / CRP2 code

```
> discount <- 0.1
>
> crp <- pumpkin::CRPPartition(n_items, concentration, discount)
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
```

Pitman-Yor / CRP2 code

```
>
> crp <- pumpkin::CRPPartition(n_items, concentration, discount)
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
```

Pitman-Yor / CRP2 code

```
> crp <- pumpkin::CRPPartition(n_items, concentration, discount)
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
```

Pitman-Yor / CRP2 code

```
> x <- pumpkin::samplePartition(crp, 10000)
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
```

Pitman-Yor / CRP2 code

```
> x[1, ]
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
```

Pitman-Yor / CRP2 code

```
  [1] 1 1 2 1 1 2 1 1 1 1 2 1 2 3 2 2 1 1 1 1 4 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 2
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
```

Pitman-Yor / CRP2 code

```
 [38] 1 1 5 2 1 1 2 1 1 2 4 4 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 6 1
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
```

Pitman-Yor / CRP2 code

```
 [75] 2 6 1 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 1 1 1 1 1 1 6
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
```

Pitman-Yor / CRP2 code

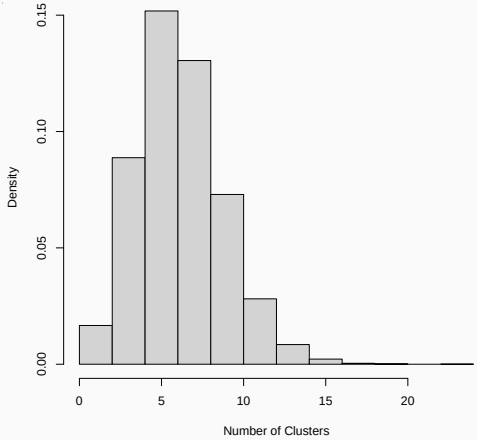
```
> x[2, ]
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Pitman-Yor / CRP2 code

```
  [1]  1  2  1  3  4  5  6  2  7  8  9  2  1  3  2 10  3  4  3  7  1  4  4  1  2
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 12  1 10  2  9 13  7 12  3 10  2
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
```

Pitman-Yor / CRP2 code

```
 [26]  3  4  1 11  4  3  4  3  5  7  7  4  3 12 1
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
```



Pitman-Yor / CRP2 code

```
 [51] 14  5  2  9  1  5  4  3  3  1  2  5 12  3  1  5  5  3  4  9 15 12 10  9  4
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
```

Pitman-Yor / CRP2 code

```
 [76]  3  4 10  7  9  4  1  1  3  2 10 12  3  7  7  4  3  4  1  7  3  3  3  7  2
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
```

Pitman-Yor / CRP2 code

```
> x[3, ]
  [1] 1 2 1 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 1 3 3 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
```

Pitman-Yor / CRP2 code

```
  [1] 1 2 1 1 1 3 1 1 1 4 1 1 1 5 1 1 1 1 1 3 3 1 1 1 1 1 1 6 1 1 1 1 1 3
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
```

Pitman-Yor / CRP2 code

```
 [38] 1 1 6 1 1 1 1 1 6 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 3 1 1 1 1 1 1
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
```

Pitman-Yor / CRP2 code

```
 [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
```

Pitman-Yor / CRP2 code

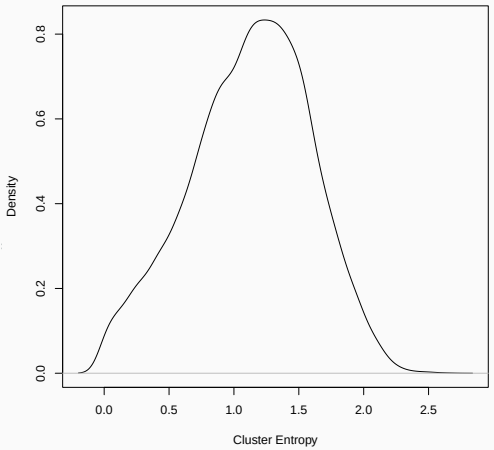
```
>
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Pitman-Yor / CRP2 code

```
> ip <- apply(x, 1, \(y) sort(tabulate(y)))
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
```

Pitman-Yor / CRP2 code

```
> ip[[1]]
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
```



Pitman-Yor / CRP2 code

```
[1]  1  1  3  5 24 66
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
```

Pitman-Yor / CRP2 code

```
> ip[[2]]
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
> gc <- sapply(ip, \(s) {
```

Pitman-Yor / CRP2 code

```
 [1]  1  1  1  1  1  1  6  6  6  7 10 11 13 15 20
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
```

Pitman-Yor / CRP2 code

```
> ip[[3]]
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = ”“, xlab = ”Number of Clusters”)
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = ”“, xlab = ”Cluster Entropy”)
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
+ })
```

Pitman-Yor / CRP2 code

```
[1]  1  1  1  2  3  6 86
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
+ })
> summary(gc)
```

Pitman-Yor / CRP2 code

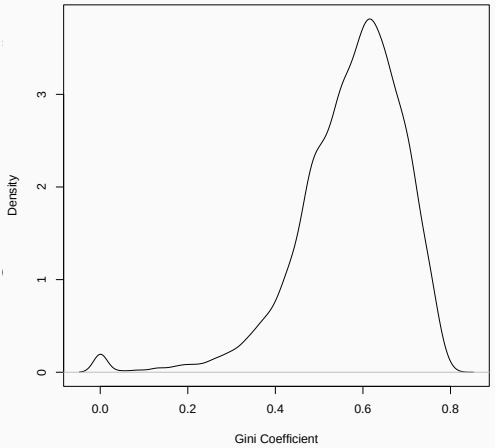
```
>
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
+ })
> summary(gc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

Pitman-Yor / CRP2 code

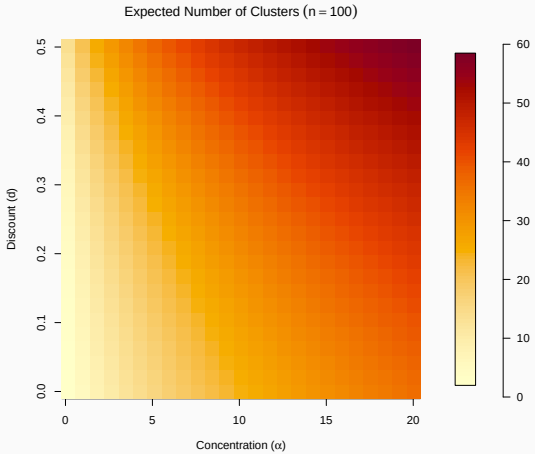
```
> n_clusters <- sapply(ip, \(s) length(s))
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab = "Number of Clusters")
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Cluster Entropy")
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
+ })
> summary(gc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.5086  0.5913  0.5733  0.6575  0.8050
```

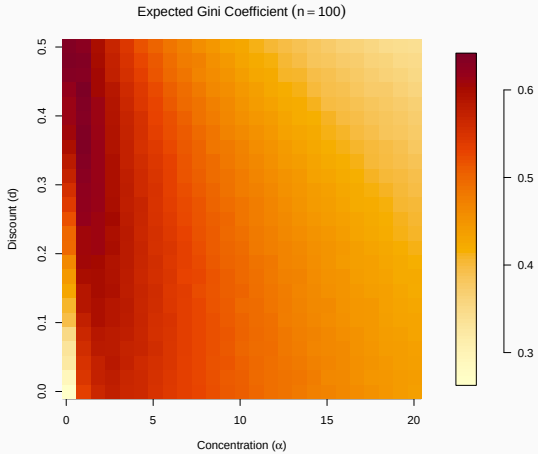
Pitman-Yor / CRP2 code

```
> summary(n_clusters)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   5.000   6.000   6.648   8.000  23.000
> hist(n_clusters, freq = FALSE, main = "", xlab
>
> entropy <- sapply(ip, \(s) {
+   p <- s / sum(s)
+   -sum(p * log(p))
+ })
> summary(entropy)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.8123  1.1563  1.1231  1.4581  2.6402
> plot(density(entropy), main = "", xlab = "Clust
>
> gc <- sapply(ip, \(s) {
+   lorenzpartition::gini_coefficient(s)
+ })
> summary(gc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.5086  0.5913  0.5733  0.6575  0.8050
> plot(density(gc), main = "", xlab = "Gini Coefficient")
```



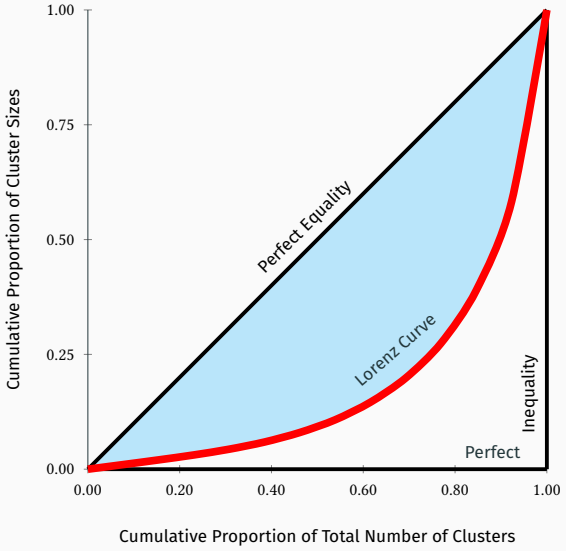
CRP heatmaps



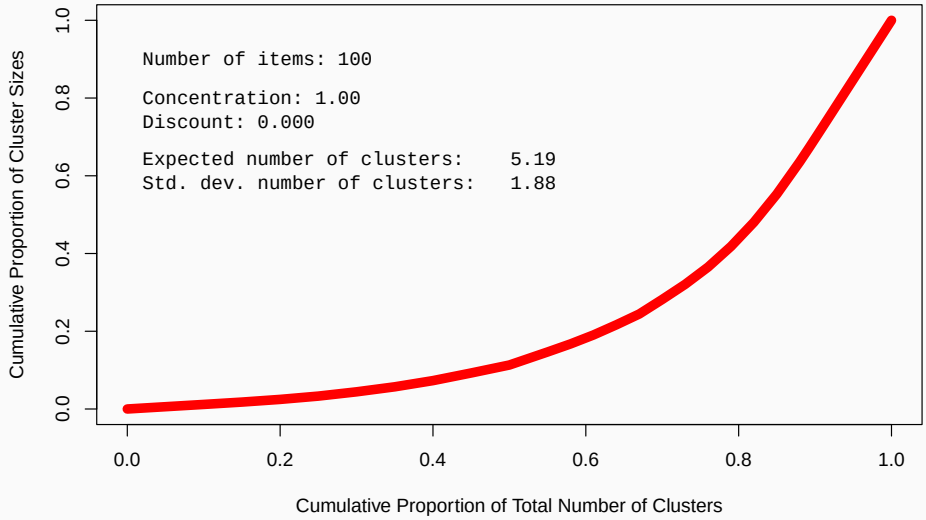


Lorenz curve: intuition for cluster sizes

- Lorenz (1905) introduced his curve to describe wealth concentration among individuals in a population.
- Our idea: Use it to parametrize beliefs about relative cluster sizes.
- Construction:
  - Order clusters, smallest to largest.
  - X-axis: cumulative fraction of clusters. Y-axis: cumulative fraction of items contained in those clusters.
- The 45° line means equal sizes; more bowed curve means more imbalance.
- Twice the blue shaded area is the Gini coefficient (Gini 1912).



CRP2 with $n = 100$ items and fixed concentration at 1.0



Desiderata for Our Partition Prior

Desiderata for our subjective prior on partitions

- **Interpretability**: Direct specification of the distribution of:
  - Number of clusters... Whatever you want!
  - Relative cluster sizes... Through an elicited Lorenz curve!
- **Controllable variability**: Concentrate sharply when strong beliefs are available, but high variability otherwise.
- **Flexibility**: Full support over all cluster size configurations.
- **Theoretically sound**: Exchangeable at fixed n, with bias and stability bounds for the induced mean profile.
- **Computability**: Evaluation and sampling should be fast and stable for very large numbers of items and clusters.

Hierarchical Construction

Hierarchical construction

$$
\begin{aligned}
k &\sim p(k), & k &\in \{1, \dots, n\} \\
\boldsymbol{x} \mid k &\sim p(\boldsymbol{x} \mid k), & \boldsymbol{x} &\in \mathscr{X}_{n,k} \\
\boldsymbol{\pi} \mid \boldsymbol{x} &\sim \mathsf{Unif}(\mathscr{P}_{n,\boldsymbol{x}}), & \boldsymbol{\pi} &\in \mathscr{P}_{n,\boldsymbol{x}}
\end{aligned}
$$

- Three-stage hierarchy: $k \to \boldsymbol{x} \to \boldsymbol{\pi}$.
- k is the number of clusters.
- $\boldsymbol{x}$ is the sorted cluster-size profile.
  Equivalently, $\boldsymbol{x}$ is an integer partition of n into k parts.
- $\boldsymbol{\pi}$ is the partition of $[n]$ with size profile $\boldsymbol{x}$.

$$
p(\boldsymbol{\pi}) = p(k)\, p(\boldsymbol{x} \mid k)\, \frac{1}{|\mathscr{P}_{n,\boldsymbol{x}}|}.
$$

- Depends on $\boldsymbol{\pi}$ only through the multiset of cluster sizes.
- Thus the induced distribution is exchangeable and admits a fixed-n EPPF.

Connection to Casella et al. (2014)

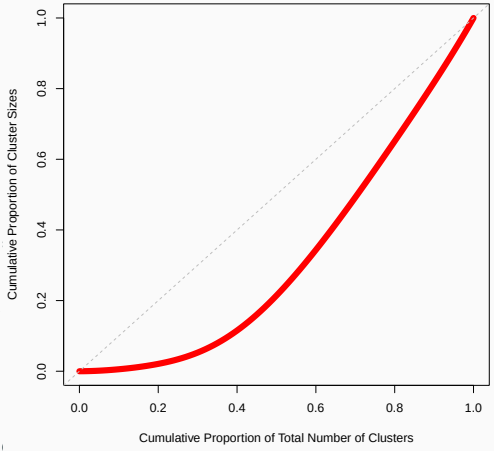
- Casella, Moreno, and Girón (2014) proposed:

$$k \sim p(k), \qquad \boldsymbol{x} \mid k \propto 1 \text{ on } \mathscr{X}_{n,k}, \qquad \boldsymbol{\pi} \mid \boldsymbol{x} \sim \text{Unif}(\mathscr{P}_{n,\boldsymbol{x}}).$$

- They emphasize objective choices for $p(k)$, e.g., truncated Poisson with Jeffreys/intrinsic prior on rate parameter.
- Our differences: Keep the uniform allocation over $\mathscr{P}_{n,\boldsymbol{x}}$ but replace the uniform $\boldsymbol{x} \mid k$ with Lorenz-IP and encourage subjective $p(k)$.
- Result: Direct control of size profiles (and tail behavior) while staying exchangeable at fixed n.

Hierarchical construction in R code

```
> set.seed(sum(utf8ToInt("Brown")))
>
> n_items <- 1000
> curve <- lorenz_ispline(c(4, 6, 25, 30, 35))
> plot(curve, lwd = 8, col = "red")
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration
[1] 9
>
> ## Sample integer partition given the number of
> (int_part <- rlorenzip(1, n_items, curve, 0.01,
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the int
> partition <- sample(rep(seq_along(int_part), int_part))
```



Hierarchical construction in R code

```
>
> n_items <- 1000
> curve <- lorenz_ispline(c(4, 6, 25, 30, 35))
> plot(curve, lwd = 8, col = "red")
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
```

Hierarchical construction in R code

```
> n_items <- 1000
> curve <- lorenz_ispline(c(4, 6, 25, 30, 35))
> plot(curve, lwd = 8, col = "red")
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
```

Hierarchical construction in R code

```
> curve <- lorenz_ispline(c(4, 6, 25, 30, 35))
> plot(curve, lwd = 8, col = "red")
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
```

Hierarchical construction in R code

```
> plot(curve, lwd = 8, col = "red")
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
```

Hierarchical construction in R code

```
>
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
```

Hierarchical construction in R code

```
> #
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
```

Hierarchical construction in R code

```
> # First sample
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
```

Hierarchical construction in R code

```
> #
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
```

Hierarchical construction in R code

```
>
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
```

Hierarchical construction in R code

```
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
```

Hierarchical construction in R code

```
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
```

Hierarchical construction in R code

```
[1] 9
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
```

Hierarchical construction in R code

```
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
```

Hierarchical construction in R code

```
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
```

Hierarchical construction in R code

```
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
```

Hierarchical construction in R code

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
```

Hierarchical construction in R code

```
[1,]    1   35   61  130  143  152  158  160  160
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

Hierarchical construction in R code

```
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
```

Hierarchical construction in R code

```
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
```

Hierarchical construction in R code

```
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
```

Hierarchical construction in R code

```
> length(partition)
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
```

Hierarchical construction in R code

```
[1] 1000
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
```

Hierarchical construction in R code

```
> tabulate(partition)
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
```

Hierarchical construction in R code

```
[1]   1  35  61 130 143 152 158 160 160
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
```

Hierarchical construction in R code

```
>
>
> #
> # Second sample
> #
> #
> ## Sample number of clusters from the CRP
> (n_clusters <- rcrpk(1, n_items, concentration = 1.0, discount = 0.0))
[1] 7
>
> ## Sample integer partition given the number of clusters from LorenzIP distribution
> (int_part <- rlorenzip(1, n_items, curve, 0.01, n_clusters = n_clusters))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    4   24  148  191  206  208  219
>
> ## Uniformly sample set partition given the integer partition
> partition <- sample(rep(seq_along(int_part), int_part))
> length(partition)
[1] 1000
> tabulate(partition)
[1]   4  24 148 191 206 208 219
```

Hierarchical construction

$$
\begin{aligned}
k &\sim p(k), & k &\in \{1, \dots, n\} \\
\boldsymbol{x} \mid k &\sim p(\boldsymbol{x} \mid k), & \boldsymbol{x} &\in \mathscr{X}_{n,k} \\
\boldsymbol{\pi} \mid \boldsymbol{x} &\sim \mathsf{Unif}(\mathscr{P}_{n,\boldsymbol{x}}), & \boldsymbol{\pi} &\in \mathscr{P}_{n,\boldsymbol{x}}
\end{aligned}
$$

- Three-stage hierarchy: $k \to \boldsymbol{x} \to \boldsymbol{\pi}$.
- k is the number of clusters.
- $\boldsymbol{x}$ is the sorted cluster-size profile.
  Equivalently, $\boldsymbol{x}$ is an integer partition of n into k parts.
- $\boldsymbol{\pi}$ is the partition of $[n]$ with size profile $\boldsymbol{x}$.

$$
p(\boldsymbol{\pi}) = p(k)\, p(\boldsymbol{x} \mid k)\, \frac{1}{|\mathscr{P}_{n,\boldsymbol{x}}|}.
$$

- Depends on $\boldsymbol{\pi}$ only through the multiset of cluster sizes.
- Thus the induced distribution is exchangeable and admits a fixed-n EPPF.

Space of set partitions with a fixed size profile

$$\mathscr{P}_{n,\boldsymbol{x}} \equiv \Big\{ \boldsymbol{\pi} \in \mathscr{P}_n \ : \ |\boldsymbol{\pi}| = k \text{ and } \boldsymbol{x}_{\boldsymbol{\pi}} = \boldsymbol{x} \Big\}.$$

- $\mathscr{P}_n$ is the set of partitions of $[n]$ and $\boldsymbol{x}_{\boldsymbol{\pi}}$ is the sorted cluster-size vector.
- $\mathscr{P}_{4,(2,2)} = \{\{\{1,2\},\{3,4\}\}, \ \{\{1,3\},\{2,4\}\}, \ \{\{1,4\},\{2,3\}\}\}$.
- $\mathscr{P}_{4,(1,3)} = \{\{\{1\},\{2,3,4\}\}, \ \{\{2\},\{1,3,4\}\}, \ \{\{3\},\{1,2,4\}\}, \ \{\{4\},\{1,2,3\}\}\}$.
- $\mathscr{P}_{4,(4)} = \{\{\{1,2,3,4\}\}\}$ and $\mathscr{P}_{4,(1,1,1,1)} = \{\{\{1\},\{2\},\{3\},\{4\}\}\}$.
- Uniform allocation over $\mathscr{P}_{n,\boldsymbol{x}}$ yields exchangeability at fixed n.

Hierarchical construction

$$
\begin{aligned}
k &\sim p(k), & k &\in \{1, \dots, n\} \\
\boldsymbol{x} \mid k &\sim p(\boldsymbol{x} \mid k), & \boldsymbol{x} &\in \mathscr{X}_{n,k} \\
\boldsymbol{\pi} \mid \boldsymbol{x} &\sim \mathsf{Unif}(\mathscr{P}_{n,\boldsymbol{x}}), & \boldsymbol{\pi} &\in \mathscr{P}_{n,\boldsymbol{x}}
\end{aligned}
$$

- Three-stage hierarchy: $k \to \boldsymbol{x} \to \boldsymbol{\pi}$.
- k is the number of clusters.
- $\boldsymbol{x}$ is the sorted cluster-size profile.
  Equivalently, $\boldsymbol{x}$ is an integer partition of n into k parts.
- $\boldsymbol{\pi}$ is the partition of $[n]$ with size profile $\boldsymbol{x}$.

$$
p(\boldsymbol{\pi}) = p(k)\, p(\boldsymbol{x} \mid k)\, \frac{1}{|\mathscr{P}_{n,\boldsymbol{x}}|}.
$$

- Depends on $\boldsymbol{\pi}$ only through the multiset of cluster sizes.
- Thus the induced distribution is exchangeable and admits a fixed-n EPPF.

Integer partition space

$$\mathscr{X}_{n,k} \equiv \left\{ \boldsymbol{x} \in \mathbb{Z}^k \,:\, 1 \le x_1 \le \cdots \le x_k,\ \sum_{j=1}^{k} x_j = n \right\}.$$

- Each $\boldsymbol{x}$ is a nondecreasing vector of cluster sizes.
- $\mathscr{X}_{7,2} = \{(1,6),(2,5),(3,4)\}$.
- $\mathscr{X}_{8,4} = \{(1,1,1,5),(1,1,2,4),(1,1,3,3),(1,2,2,3),(2,2,2,2)\}$.
- $\mathscr{X}_{5,1} = \{(5)\}$ and $\mathscr{X}_{5,5} = \{(1,1,1,1,1)\}$.
- $\mathscr{X}_{10,3} = \{(1,1,8),(1,2,7),(1,3,6),(1,4,5),(2,2,6),(2,3,5),(2,4,4),(3,3,4)\}$.
- A key contribution is the Lorenz-IP distribution, which indexes a distribution on $\mathscr{X}_{n,k}$ by a Lorenz curve and a concentration curve.

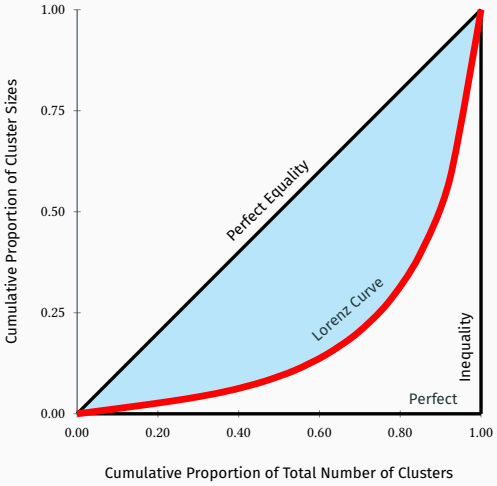
Lorenz Integer Partition (Lorenz-IP) Distribution

From a Lorenz curve to target cluster sizes for fixed k

- After sampling k, it is fixed and we work on $\mathcal{X}_{n,k}$ (ordered size profiles).
- Choose a Lorenz curve $\mathscr{L}(u)$ on $u \in [0, 1]$.
- Discretize at $u_j = j/k$:

$$\omega_j = \mathscr{L}(u_j) - \mathscr{L}(u_{j-1}), \quad j = 1, \dots, k.$$

- $\boldsymbol{\omega} = (\omega_1, \dots, \omega_k)$ sums to 1 and encodes target relative cluster sizes.
- We aim for $\mathbb{E}[X_j]/n \approx \omega_j$.



Sequential construction given k and ω

- Build sizes from smallest to largest such that: 1. ordering is automatic and 2. the sum-to-n constraint is automatic.
- Start with X_1 on $\{1, 2, \dots, \lfloor n/k \rfloor\}$ and mean $\mu_1 = n\omega_1$.

Sequential construction given k and ω

- Build sizes from smallest to largest such that: 1. ordering is automatic and 2. the sum-to-n constraint is automatic.
- Start with X_1 on $\{1, 2, \dots, \lfloor n/k \rfloor\}$ and mean $\mu_1 = n\omega_1$.
- Given X_1, draw X_2 on

$$L_2 = X_1, \qquad U_2 = \left\lfloor \frac{n - X_1}{k - 1} \right\rfloor,$$

with mean μ_2 set by the Lorenz target.

Sequential construction given k and ω

- Build sizes from smallest to largest such that: 1. ordering is automatic and 2. the sum-to-n constraint is automatic.
- Start with X_1 on $\{1, 2, \dots, \lfloor n/k \rfloor\}$ and mean $\mu_1 = n\omega_1$.
- Given X_1, draw X_2 on

$$L_2 = X_1, \qquad U_2 = \left\lfloor \frac{n - X_1}{k - 1} \right\rfloor,$$

  with mean μ_2 set by the Lorenz target.
- In general, after drawing $X_{1:i-1}$, the feasible interval for X_i is

$$L_i = X_{i-1}, \qquad U_i = \left\lfloor \frac{n - \sum_{j<i} X_j}{k - i + 1} \right\rfloor.$$

- Draw X_i from a bounded-support kernel on $\{L_i, \dots, U_i\}$ with mean μ_i.
  - Determining μ_i and choosing the mean-parameterized bounded-support kernel are the linchpins of the method.
- Set $X_k = n - \sum_{j<k} X_j$.

Obtain μ_i from κ_i and the realized bounds L_i and U_i

- Interpolation coefficient and conditional mean:

$$\kappa_i = \frac{\omega_i - \omega_{i-1}}{\bar{u}_i(\boldsymbol{\omega}) - \omega_{i-1}}, \qquad \mu_i = L_i + \kappa_i(U_i - L_i).$$

  where

$$\bar{u}_i(\boldsymbol{\omega}) = \frac{1 - \sum_{j<i} \omega_j}{k - i + 1}.$$

  is the remaining-average share from the Lorenz target.
- Note: κ_i is computed once and can be used many times.
- Intuition: κ_i places the *conditional* mean between the smallest and largest feasible sizes.
- With a mean-parameterized kernel, $\mathbb{E}[X_i \mid X_{1:i-1}] = \mu_i$; by iterated expectations, $\mathbb{E}[X_i] = \mathbb{E}[\mu_i]$, so marginal means follow the target (up to rounding in U_i).
- $\kappa_i \approx 0$ keeps X_i near L_i (more imbalance); $\kappa_i \approx 1$ pushes toward U_i (more balanced).

Base kernel: mean + concentration

- At each step we need a distribution on $\{L_i, \dots, U_i\}$ with mean μ_i.
- A concentration parameter γ_i controls how tightly X_i concentrates around μ_i.
- Next: two concrete kernels (exponential-like vs power-law) and the theory that guarantees feasibility, support, and bias bounds.

Two concrete kernels

TiDaL
(truncated discrete Laplace)

$$p_{[L,U]}(x \mid M, \gamma) \propto \exp\Big(-\frac{\gamma}{U-L}\,|x-M|\Big), \quad x \in \{L, \dots, U\}.$$

- Exponential-like tails on $[L, U]$.
- Closed-form normalizer and moments.
- Mean calibration via a one-dimensional solve for M.

TaDPoLe
(truncated discrete power law)

$$p_{[L,U]}(x \mid M, \gamma, \alpha) \propto \left(|x-M| + \frac{U-L}{\gamma}\right)^{-(1+\alpha\gamma)}, \quad x \in \{L, \dots, U\}.$$

- Heavier tails — useful for extreme size profiles — controlled by α.
- Normalizers and moments via standard special-function evaluations (Hurwitz zeta).
- Mean calibration via a one-dimensional solve for M.

Prior elicitation workflow

1. Choose $p(k)$ to encode beliefs about the number of clusters.
2. Choose a target Lorenz curve.
3. Choose tail behavior: TiDaL (exponential) vs TaDPoLe (power-law).
4. Choose concentrations $\gamma_{1:k-1}$ to control tightness.

Theory

Base Kernel Properties 1-3

Property (Existence region)

For each $[L, U]$ and $\gamma > 0$, there is a feasible-mean interval $\mathscr{M}_{[L,U]}(\gamma) \subseteq [L, U]$ such that the base kernel $p_{[L,U]}(z \mid \mu, \gamma)$ exists iff $\mu \in \mathscr{M}_{[L,U]}(\gamma)$. When $L = U$, we take $\mathscr{M}_{[L,U]}(\gamma) = \{L\}$.

Property (Mean parameterization)

For any (μ, γ) for which the base kernel exists, it has expected value $\mathbb{E}[Z]$ equal to μ.

Property (Full support)

For any (μ, γ) in which the base kernel exists, it has full support on $\{L, \dots, U\}$, i.e., $\mathbb{P}(Z = z) > 0$ for all $z \in \{L, \dots, U\}$.

Base Kernel Properties 4-6

Property (Interior feasibility with uniform threshold)

For every $\eta \in (0, 1/2]$, there exists a finite uniform feasibility threshold $\gamma_{\text{feas}}(\eta)$ such that whenever $W := U - L \geq 1$ and $\mu \in [L + \eta W,\, U - \eta W]$, the base kernel exists for all $\gamma \geq \gamma_{\text{feas}}(\eta)$.

Property (Eventual feasibility)

Fix $[L, U]$ with $L < U$ and any $\mu \in (L, U)$. There exists a finite $\gamma_0(L, U, \mu)$ such that $\mu \in \mathscr{M}_{[L,U]}(\gamma)$ for all $\gamma \geq \gamma_0(L, U, \mu)$.

Property (Two-point concentration)

Fix $[L, U]$ with $L < U$ and any $\mu \in (L, U)$, and fix any $\gamma_0(L, U, \mu)$ such that $\mu \in \mathscr{M}_{[L,U]}(\gamma)$ for all $\gamma \geq \gamma_0(L, U, \mu)$. For every $\varepsilon \in (0, 1)$ there exists $\gamma^\star(\varepsilon; L, U, \mu) \geq \gamma_0(L, U, \mu)$ such that for $Z \sim \mathrm{Base}_{[L,U]}(\mu, \gamma)$ and all $\gamma \geq \gamma^\star(\varepsilon; L, U, \mu)$,

$$\mathbb{P}\big(Z \in \{\lfloor \mu \rfloor, \lceil \mu \rceil\}\big) \geq 1 - \varepsilon. \tag{1}$$

When $\mu \in \mathbb{Z}$, the set $\{\lfloor \mu \rfloor, \lceil \mu \rceil\}$ is the singleton $\{\mu\}$.

Target mean profile and feasibility

- Technical conditions used in the theory:

$$\omega_{j-1} < \omega_j < \frac{1 - \sum_{i<j} \omega_i}{k - j + 1}, \qquad j = 1, \ldots, k-1,$$

$$1 < n\omega_1 < \left\lfloor \frac{n}{k} \right\rfloor,$$

- Here $\omega_0 := 0$ and $j = 1, \ldots, k-1$.
- These keep the sequential construction feasible.
- In the theory, we assume $n \geq 2k$ so that $\lfloor n/k \rfloor \geq 2$.
- In the software, we are robust to violations of what's needed for the theory.

Feasibility guard for the base kernel

$$\tilde{\gamma}_i = \max\{\gamma_i, \gamma_{\text{feas}}(\eta_i)\}, \quad \eta_i = \min\{\kappa_i, 1 - \kappa_i\}.$$

- Ensures the kernel exists for the chosen mean.
- If baseline γ_i is already large enough, the guard is inactive.

Definition (Lorenz-IP)

Given $(n, k, \boldsymbol{\omega})$ with strictly interior target means $\omega_{j-1} < \omega_j < \frac{1-\sum_{i<j}\omega_i}{k-j+1}$ for $j = 1, \dots, k-1$ and $1 < n\omega_1 < \lfloor n/k \rfloor$, baseline concentration parameters $\gamma_1, \dots, \gamma_{k-1} > 0$, and (for TaDPoLe) shape parameters $\alpha_1, \dots, \alpha_{k-1} > 0$, the Lorenz-IP distribution is the joint distribution of $\boldsymbol{X} = (X_1, \dots, X_k)$ produced by the following sequential algorithm:

1. Draw $X_1 \sim \mathrm{Base}_{[L_1, U_1]}(\mu_1, \tilde{\gamma}_1)$, with $L_1 = 1$, $U_1 = \lfloor n/k \rfloor$, and $\mu_1 = n\omega_1$.
2. For $i = 2, \dots, k-1$, set $L_i = X_{i-1}$ and $U_i = \left\lfloor \frac{n - \sum_{j=1}^{i-1} X_j}{k-i+1} \right\rfloor$, then draw
$$X_i \sim \mathrm{Base}_{[L_i, U_i]}\big(\mu_i(X_{1:i-1}), \tilde{\gamma}_i\big).$$
3. Set $X_k := n - \sum_{j=1}^{k-1} X_j$.

When the base kernel is TaDPoLe, each draw uses its coordinate-specific shape parameter α_i.

Valid distribution with full support and ability to concentrate

Proposition (Well-definedness)

Under the base-kernel properties and the strict interior target-mean conditions on ω, the Lorenz-IP algorithm indeed defines a valid distribution on $\mathcal{X}_{n,k}$ almost surely.

Theorem (Full support)

The Lorenz-IP distribution has full support on $\mathcal{X}_{n,k}$: every nondecreasing integer partition receives positive probability.

Theorem (Any integer partition can be made dominant)

Assume $n \geq 2k$. For any $\boldsymbol{x} \in \mathcal{X}_{n,k}$ and $\varepsilon \in (0,1)$, there exist target means $\boldsymbol{\omega}$ with $\omega_{j-1} < \omega_j < \frac{1-\sum_{i<j}\omega_i}{k-j+1}$ for $j = 1, \ldots, k-1$ and $1 < n\omega_1 < \lfloor n/k \rfloor$, and concentrations $\gamma_1, \ldots, \gamma_{k-1}$ such that the Lorenz-IP distribution satisfies $\mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) \geq 1 - \varepsilon$.

Mean accuracy guarantee

Theorem (Uniform $O(1/n)$ bias)

Under mean-parameterized base kernels (so $\mathbb{E}[Z] = \mu$) and strictly interior target means $\omega_{j-1} < \omega_j < \frac{1-\sum_{i<j}\omega_i}{k-j+1}$ for $j = 1, \dots, k-1$ with $1 < n\omega_1 < \lfloor n/k \rfloor$,

$$\max_{1 \le j \le k} \left| \frac{\mathbb{E}[X_j]}{n} - \omega_j \right| \le \frac{C_k(\kappa)}{n},$$

where $C_k(\kappa)$ depends only on k and $\kappa = (\kappa_1, \dots, \kappa_{k-1})$. Thus, for fixed (k, ω), Lorenz-IP marginal means converge uniformly to ω at rate $O(1/n)$.

- Bias arises only from integer rounding in the sequential upper bounds.
- A deterministic recursion bounds cumulative rounding error, yielding $C_k(\kappa)$ and the $O(1/n)$ rate.

Stability to target perturbations

Corollary (Lipschitz stability)

If two target vectors ω and ω' both satisfy $\omega_{j-1} < \omega_j < \frac{1-\sum_{i<j}\omega_i}{k-j+1}$ for $j = 1, \dots, k-1$ with the same (n, k), the corresponding Lorenz-IP mean profiles differ by at most $\|\omega - \omega'\|_1 + 2C_{\max}/n$, where $C_{\max}$ is the larger of the two bias constants.

- Small changes in the target Lorenz curve induce small changes in mean size profiles.
- Elicitation is robust: near-by targets yield near-by priors, especially for larger n.
- Provides a simple sensitivity bound for tuning ω.

Exchangeability

Fixed-n focus vs. projective consistency

- Kingman paintbox consistency is not imposed by design.
- We treat n as fixed and elicit priors directly on $\mathscr{P}_n$.
- This yields direct control over k and the size profile at the observed sample size.

Computation

Computation at a glance

- Sequential evaluation and sampling: one bounded-support pmf per coordinate.
- Base kernels have closed-form normalizers and exact sampling schemes.
- Complexity is $O(k)$ per draw of $\boldsymbol{x}$; uniform allocation uses a standard partition representation.
- MCMC updates can reuse familiar item-allocation moves (collapsed or uncollapsed).

Benchmarks

```
> options(width = 120)
>
> partition <- c(4, 14, 32, 50)
> curve <- lorenz_ispline(partition)
>
> microbenchmark(times = 1000,
+   "k = 4, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
```

Benchmarks

```
>
> partition <- c(4, 14, 32, 50)
> curve <- lorenz_ispline(partition)
>
> microbenchmark(times = 1000,
+   "k = 4, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
```

Benchmarks

```
> partition <- c(4, 14, 32, 50)
> curve <- lorenz_ispline(partition)
>
> microbenchmark(times = 1000,
+   ”k = 4, n = 100” = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 1,000” = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 10,000” = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 100,000” = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 1,000,000” = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   ”k = 8, n = 100” = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 1,000” = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
```

Benchmarks

```
> curve <- lorenz_ispline(partition)
>
> microbenchmark(times = 1000,
+   "k = 4, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
```

Benchmarks

```
>
> microbenchmark(times = 1000,
+   ”k = 4, n = 100” = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 1,000” = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 10,000” = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 100,000” = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   ”k = 4, n = 1,000,000” = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   ”k = 8, n = 100” = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 1,000” = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 10,000” = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 100,000” = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
```

Benchmarks

```
> microbenchmark(times = 1000,
+   "k = 4, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
      k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
```

Benchmarks

```
+   "k = 4, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                  expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
```

Benchmarks

```
+   "k = 4, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
```

Benchmarks

```
+   "k = 4, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                 expr     min       lq     mean   median       uq      max neval
```

Benchmarks

```
+   "k = 4, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 4),
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                 expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
```

Benchmarks

```
+   "k = 4, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 4)
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                 expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
     k = 8, n = 1,000 103.485 109.4515 110.9147 110.1730 111.5350  181.432  1000
```

Benchmarks

```
+ )
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                 expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
     k = 8, n = 1,000 103.485 109.4515 110.9147 110.1730 111.5350  181.432  1000
    k = 8, n = 10,000 107.503 111.2700 113.3332 112.3415 114.0800  286.470  1000
```

Benchmarks

```
Unit: microseconds
                 expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   ”k = 8, n = 100” = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 1,000” = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 10,000” = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 100,000” = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   ”k = 8, n = 1,000,000” = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                 expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
     k = 8, n = 1,000 103.485 109.4515 110.9147 110.1730 111.5350  181.432  1000
    k = 8, n = 10,000 107.503 111.2700 113.3332 112.3415 114.0800  286.470  1000
   k = 8, n = 100,000 163.719 269.1170 276.4181 273.0200 278.6550 1920.910  1000
```

Benchmarks

```
                  expr     min       lq      mean  median       uq      max neval
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                  expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
     k = 8, n = 1,000 103.485 109.4515 110.9147 110.1730 111.5350  181.432  1000
    k = 8, n = 10,000 107.503 111.2700 113.3332 112.3415 114.0800  286.470  1000
   k = 8, n = 100,000 163.719 269.1170 276.4181 273.0200 278.6550 1920.910  1000
 k = 8, n = 1,000,000 220.175 368.6900 380.5347 377.3510 420.7930 1991.483  1000
```

Benchmarks

```
       k = 4, n = 100  93.406  97.2935  99.37528  97.935  99.0770  976.681  1000
     k = 4, n = 1,000  95.530  98.0450  99.48815  98.691 100.0130  201.079  1000
    k = 4, n = 10,000  96.792  99.2920 101.08566 100.229 101.8170  121.298  1000
   k = 4, n = 100,000 100.309 157.3060 189.89491 205.322 207.1605  778.948  1000
 k = 4, n = 1,000,000 101.712 206.1390 201.11273 207.726 209.3095 1613.340  1000
>
> microbenchmark(times = 1000,
+   "k = 8, n = 100" = rlorenzip(1, 100, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000" = rlorenzip(1, 1000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 10,000" = rlorenzip(1, 10000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 100,000" = rlorenzip(1, 100000, curve, 0.1, n_clus = 8),
+   "k = 8, n = 1,000,000" = rlorenzip(1, 1000000, curve, 0.1, n_clus = 8)
+ )
Unit: microseconds
                expr     min       lq     mean   median       uq      max neval
       k = 8, n = 100  98.044 109.0350 110.6104 109.8170 111.0295  198.454  1000
     k = 8, n = 1,000 103.485 109.4515 110.9147 110.1730 111.5350  181.432  1000
    k = 8, n = 10,000 107.503 111.2700 113.3332 112.3415 114.0800  286.470  1000
   k = 8, n = 100,000 163.719 269.1170 276.4181 273.0200 278.6550 1920.910  1000
 k = 8, n = 1,000,000 220.175 368.6900 380.5347 377.3510 420.7930 1991.483  1000
>
```

Conclusion and Open Questions

Conclusion and open questions

- **Control**: Separate priors on k and relative cluster sizes; can concentrate on any $\boldsymbol{x} \in \mathscr{X}_{n,k}$.
- **Tools**: Lorenz-IP plus TiDaL/TaDPoLe give bounded discrete kernels with tunable tails.
- **Fixed-n tradeoff**: No marginal invariance; priors are tailored to the observed n.
- **Beyond clustering**: Lorenz-IP as a prior for other integer-partition problems.
- **Open questions**:
  - Posterior consistency for k under suitable conditions?
  - Should the Lorenz curve depend on n or k?
  - The Lorenz curve is cumulative, like a CDF; is there a density-style summary that reveals subtle differences?

Thanks!