When Philosophy Becomes Code: Distributional Justice Metrics in ErisML

A discussion document for the Philosophy Club

Introduction

What if you could *run* Rawls? Not just read him, not just argue about him—but execute his theory of justice on actual data and compare the results to Bentham's utilitarian calculus?

That's essentially what this code does.

The ErisML library now includes six distributional fairness metrics that translate major philosophical positions on justice into executable algorithms. For a philosophy club that discusses Rawls, this is an unusual opportunity: the centuries-old debate between competing theories of distributive justice has been rendered *testable*.

The Metrics and Their Philosophical Roots

1. Rawlsian Maximin

The Philosophy: In *A Theory of Justice* (1971), John Rawls argued that principles of justice should be chosen from behind a "veil of ignorance"—not knowing what position you'd occupy in society. From this perspective, rational agents would adopt the **Difference Principle**: inequalities are justified only if they benefit the worst-off members of society.

The **maximin rule** follows: maximize the minimum. When evaluating distributions, look at whoever is worst off and optimize for them.

The Code:

```python
def rawlsian_maximin(moral_tensor) -> float:
    """Identify the welfare of the worst-off party across all dimensions."""
    return min(minimum_welfare_per_party)
```

The function looks at all parties in a distribution, finds whoever has the lowest welfare, and returns that value. When comparing two policies, the Rawlsian metric prefers whichever policy makes the worst-off party better off—even if total welfare is lower.

Philosophical Implications: This is Rawls' core insight made computational. You can now:

- Compare policies by their treatment of the worst-off
- See exactly where Rawlsian and utilitarian judgments diverge
- Test edge cases Rawls never imagined

---

2. Utilitarian Aggregation (Sum and Average)

The Philosophy: Jeremy Bentham and John Stuart Mill argued that the right action is whatever maximizes total utility—"the greatest good for the greatest number." Welfare is *aggregated*: individual utilities are summed together.

The Code:

```python
def utilitarian_sum(moral_tensor) -> float:
    """Total welfare: sum across all parties."""
    return total_welfare

def utilitarian_average(moral_tensor) -> float:
    """Mean welfare: total divided by number of parties."""
    return total_welfare / n_parties
```

Simple addition. Every person's welfare counts equally, and we maximize the total.

Philosophical Implications: The utilitarian metrics reveal the classic tension:

- A policy that makes one person extremely well off while slightly harming many others can score high on utilitarian_sum but poorly on maximin
- This is the "utility monster" problem, now quantifiable
- You can calculate exactly how much inequality utilitarianism tolerates

---

3. Prioritarian Weighted Welfare

The Philosophy: Prioritarianism (associated with Derek Parfit and Thomas Nagel) is a middle ground between Rawls and the utilitarians. It says:

- Benefits to the worse-off matter *more* than benefits to the better-off
- But not infinitely more (unlike Rawls' lexical priority)
- We weight welfare by vulnerability

This captures the intuition that a dollar means more to a poor person than a rich person—mathematically.

The Code:

```python
def prioritarian_weighted_welfare(moral_tensor, alpha=0.5) -> float:
    """Aggregate welfare with priority weighting for worse-off parties."""
    # Lower welfare gets higher weight
    # alpha controls how much priority the worse-off receive
```

The `alpha` parameter is philosophically significant: it controls how much we prioritize the disadvantaged. α=0 collapses to utilitarianism; higher α approaches Rawlsian maximin.

Philosophical Implications: Prioritarianism has always been somewhat vague about "how much" priority to give. The code forces precision:

- What value of α best captures your intuitions?
- At what α do prioritarian and maximin judgments converge?
- Is there a natural or defensible value for α?

---

4. Gini Coefficient

The Philosophy: The Gini coefficient comes from welfare economics, not philosophy departments—but it operationalizes the egalitarian intuition that inequality itself is bad, independent of aggregate welfare.

Gini = 0 means perfect equality (everyone has the same).
Gini = 1 means perfect inequality (one person has everything).

The Code:

```python
def gini_coefficient(moral_tensor) -> float:
    """Measure inequality in distribution. [0=equal, 1=unequal]"""
```

Philosophical Implications: For luck egalitarians and relational egalitarians, the Gini provides a pure measure of inequality. Combined with the other metrics, you can ask:

- How much inequality does maximizing utility require?
- Does Rawlsian maximin actually reduce inequality, or just protect the floor?
- What's the efficiency-equality tradeoff curve?

---

5. Atkinson Index

The Philosophy: The Atkinson index (developed by economist Anthony Atkinson) measures inequality with an explicit **inequality aversion parameter** ε (epsilon):

- ε = 0: We don't care about inequality at all
- ε = 1: Moderate inequality aversion
- ε → ∞: Infinite aversion (approaches Rawlsian focus on the minimum)

This makes your egalitarian intuitions precise: *how much* do you care about inequality versus total size?

The Code:

```python
def atkinson_index(moral_tensor, epsilon=0.5) -> float:
    """Parametric inequality with epsilon sensitivity. [0, 1]"""
```

Philosophical Implications: The Atkinson index forces a question philosophers often avoid: what's your inequality aversion *number*? This is uncomfortable but clarifying. Different values of ε correspond to different moral philosophies:

- ε = 0.5: Mild egalitarianism
- ε = 1.0: Moderate (roughly prioritarian)
- ε = 2.0: Strong egalitarianism

---

6. Theil Index

The Philosophy: The Theil index comes from information theory (entropy). Its philosophical significance is *decomposability*: you can break down total inequality into between-group and within-group components.

This matters for debates about intersectionality and group-based justice. Is inequality primarily between groups (race, gender, class) or within them?

The Code:

```python
def theil_index(moral_tensor) -> float:
    """Generalized entropy. [0, ∞)"""

def theil_decomposition(moral_tensor, groups) -> dict:
    """Decompose inequality into between-group and within-group components."""
```

Philosophical Implications: The decomposition is powerful for political philosophy:

- If between-group inequality dominates, group-targeted policies may be justified
- If within-group inequality dominates, individual-level interventions matter more
- You can quantify how much each form contributes

The Deeper Significance: Philosophy Made Computational

These six metrics aren't just academic exercises. They're being implemented in an ethics library for AI systems. Here's why that matters:

1. Forcing Precision

Philosophy often thrives on productive ambiguity. But when you code a theory, you can't be vague. Every edge case needs handling:

- What if there are zero parties?
- What if someone has negative welfare?
- What if everyone has identical welfare?

The 74 tests in this PR force these decisions.

2. Making Debates Empirical

Consider the utilitarian vs. Rawlsian debate. Traditionally, this is settled by intuition pumps and thought experiments. Now you can:

- Run both metrics on real distributions
- Identify exactly where they disagree
- Quantify how often they disagree
- Find the distributions where the choice matters most

3. The Parameters Are Moral Commitments

The `alpha` in prioritarianism, the `epsilon` in Atkinson—these aren't arbitrary technical choices. They're moral parameters. Adjusting them is doing ethics.

This opens new questions:

- Can we elicit people's implicit parameters from their judgments?
- Do people have consistent parameters across contexts?
- What parameters should an AI system use?

Discussion Questions for the Club

1. **The Specification Problem**: When you translate Rawls into code, you make choices he never made. Is the resulting algorithm still "Rawlsian"? What would Rawls think of having his theory executed by a computer?
2. **The Parameter Problem**: Prioritarianism and the Atkinson index require choosing numerical parameters. Is this a feature (forcing precision) or a bug (false precision about inherently vague concepts)?
3. **Computational Reflective Equilibrium**: Rawls proposed reflective equilibrium—going back and forth between principles and intuitions until they cohere. Could you run this process computationally? Compare metric outputs to human judgments, adjust parameters, repeat?
4. **AI Ethics Implications**: If an AI system uses these metrics to make allocation decisions, which metric should it use? Should it be configurable? Should users choose their own inequality aversion parameter?
5. **The Limits of Formalization**: Are there aspects of distributive justice that *can't* be captured in code? What's lost in translation?

Further Reading

- Rawls, J. (1971). *A Theory of Justice*. Harvard University Press.
- Sen, A. (1973). *On Economic Inequality*. Oxford University Press.
- Parfit, D. (1997). "Equality and Priority." *Ratio*.
- Atkinson, A.B. (1970). "On the Measurement of Inequality." *Journal of Economic Theory*.

PR Link: https://github.com/ahb-sjsu/erisml-lib/pull/49

This document was prepared for the Philosophy Club discussion on distributional justice and computational ethics.