Cumulative Prospect Theory: A Simulative Demonstration

A Historical and Mathematical Overview: Cumulative Prospect Theory

Prospect Theory was first introduced in Kahneman and Tversky (1979), a theory designed to explain irrational behavior under uncertainty. Famous examples of this irrational behavior include the famous *Allais* or *St. Petersburg Paradox*. The theory operates as an amendment to Expected Utility Theory. While Prospect Theory did what it intended, it often violated stochastic dominance and transitivity assumptions. The idea of Rank-Dependent Utility (RDU) was conceptualized in Quiggin (1982) to avoid these violations, and soon followed Kahneman and Tversky's Advances in Prospect Theory: Cumulative Representation of Uncertainty (1992) which incorporated RDU fundamentals.

There are two, defining observations of CPT. One observation is that agents make decisions based on a particular reference point rather than the final outcome. It follows that agents feel differently depending on whether or not they are subject to losses or gains, overreacting to potential losses and possibly underreacting to potential gains. This is typically referred to as *loss aversion*. The other observation is that people assign more weight to extreme events than they would mundane events.

We will start the mathematical overview by considering the gamble:

$$\tilde{x} = (x_{-m},\ p_{-m};\ \dots\ ; x_{-1},\ p_{-1}; x_0,\ p_0; x_1,\ p_1;\ \dots\ ; x_n,\ p_n)$$

where $x_i < x_j$ for $i < j$, $x_0 = 0$, and $\sum_{i=-m}^{n} p_i = 1$. Under CPT, an agent is proposed to assign the preceding lottery the value

$$\sum_{i=-m}^{n} \pi_i v(x_i) \tag{1}$$

where π_i can be thought of as the cumulative weighted probability of achieving some value $v(x_i)$, where the function $v(\cdot)$ is known as the value function. Kahneman and Tversky elaborate on the above with the following functions:

$$\pi_i = \begin{cases} w^+(p_i + \ldots + p_n) - w^+(p_{i+1} + \ldots + p_n), & \text{for } 0 \le i \le n \\ w^-(p_{-m} + \ldots + p_i) - w^-(p_{-m} + \ldots + p_{i-1}), & \text{for } -m \le i < 0 \end{cases}$$

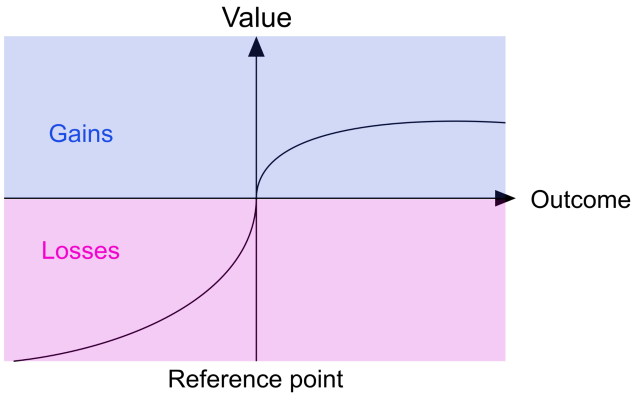
where $w^+(\cdot)$ and $w^-(\cdot)$ are probability weights,

$$v(x) = \begin{cases} x^\alpha, & \text{for } x \ge 0 \\ -\lambda(-x)^\alpha, & \text{for } x < 0 \end{cases}$$

and

$$w^+(p) = \frac{p^\gamma}{[p^\gamma + (1-p^\gamma)]^{1/\gamma}}, \qquad w^-(p) = \frac{p^\delta}{[p^\delta + (1-p^\delta)]^{1/\delta}}$$

where $\alpha, \gamma, \delta \in (0, 1)$ and $\lambda > 1$. It should be noted that $v(0) = 0$, $w^+(0) = w^+(0) = 0$, and $w^+(1) = w^+(1) = 1$. An accompanying sketch of a common value function can be seen below, where the convexity over losses and concavity over gains are displayed. Note the functions non-differentiability at origin.



Those familiar with Expected Utility Theory's (EUT) may already see parallels between the expected utility function, (2), and the value function, (1). The expected utility function described in CPT's predecessor, EUT, is

$$\sum_{i=1}^{n} p_i U(w_0 + x_i) \tag{2}$$

where w_0 is the agents initial wealth, $U(\cdot)$ is the agents utility function, and p_i is the probability associated with outcome x_i.

There are a couple of notable differences between (1) and (2). The first is that (1) assesses a gamble with potential gains and losses, while (2) assesses changes in wealth level. Secondly, the utility function in (2), $U(\cdot)$, is differentiable everywhere while the analogous value function in (1), $v(\cdot)$, is non-differentiable at origin to indicate loss aversion (more sensitive to losses than to equivalent gains). The parameter λ is typically used to determine the severity of loss aversion. Thirdly, $U(\cdot)$ is concave throughout its domain, while $v(\cdot)$ is concave over positive x-values (gains) but convex over negative x-values (losses). This concavity is determined by our parameter, α. Lastly, (2) shows the agent acting rationally and using objective probabilities to make decisions, while (1) shows a weighting process in which total probabilities over gains and

losses are distorted, separately. This aspect of our equation ensures that our agent over exaggerates tail-end probabilities; that is, he or she is especially averse to extreme outcomes.

Applications

From decisions being made under uncertain conditions in a hospital, to decisions made under uncertain stock market conditions, Cumulative Prospect Theory (CPT) has several applications that are still being explored today.

According to "Neuroeconomics, health psychology, and the interdisciplinary study of preventative health behavior" by DeStasio, Clithero and Berkman (2019), the notion that decision making is a common denominator in preventable disease is "acutely applicable". Furthermore, the paper discusses some of the main medical contributions of CPT. Firstly, patients will react differently to circumstances phrased as a string of potential gains v.s. potential losses, based on their reference point. Secondly, this reference point may impact patients' preventive health behaviors that are "unpleasant themselves (e.g., vaccinations or invasive screening tests), where the risk of the immediate negative outcome (e.g., pain) is felt higher than the risk of the potential long-term outcome" (Gisbert-Pérez, Martí-Vilar, and González-Sala, 2022). These phenomena are qualitative indicators that framing a decision in a particular way - around either gains or losses - can influence preventive and reactionary health behaviors.

On the quantitative side, the article "Prospect Theory and Stock Returns: An Empirical Test" by Barberis, Mukherjee, and Wang (2016) introduces a mathematical model using CPT that supposedly captures an investor's feelings towards a stock, assuming that he or she was able to gather historical data based on that stock. Given the historical return of a stock over the past 60 months, suppose that m of these returns are negative, while $n = 60 - m$ of these returns are

positive. We can then formulate a gamble in which r_{-m} is the most negative return and r_n is the most positive over the past 60 months as follows:

$$\tilde{x} = (r_{-m}, \frac{1}{60}; \ldots; r_{-1}, \frac{1}{60}; r_0, \frac{1}{60}; r_1, \frac{1}{60}; \ldots; r_n, \frac{1}{60})$$

where $p_i = \frac{1}{60}, \forall i$. From (1) of the CPT overview, it stands to reason that the value of this gamble as assigned by an investor is of the form:

$$\sum_{i=-m}^{n} \pi_i v(x_i)$$

$$= \sum_{i=-m}^{-1} v(r_i)\left[w^-\left(\frac{i+m+1}{60}\right) - w^-\left(\frac{i+m}{60}\right)\right] + \sum_{i=1}^{n} v(r_i)\left[w^+\left(\frac{n-i+1}{60}\right) - w^+\left(\frac{n-i}{60}\right)\right]$$

An important property of this model is that it is time insensitive; that is, the order of returns and in which month they occurred is of no importance to the model. In fact, this model only adequately describes an investor's attitude towards a stock as a whole, with equal probabilities for each return and no probability weighting based on the particular time of a return. There exists modifications of this model that time-weight the returns.

The Simulation

Cumulative Prospect Theory (CPT) suggests that behavior under uncertainty can be quantified and thus predicted. I intend to demonstrate this behavior with a simulation. Using the Python language, I will utilize object oriented programming to construct value and probability models according to Kahneman and Tversky.

➔ *Class Initialization*

The script starts with the class "cpt," which upon initialization, takes the parameters appropriate for cumulative prospect theory. That is,

```
class cpt:
    def __init__(self, alpha, gamma, sigma, lam):
        self.alpha = alpha
        self.gamma = gamma
        self.sigma = sigma
        self.lam = lam

        # Parameter restrictions
        if self.lam <= 1:
            raise ValueError("Lambda value must be greater than 1")
        if (self.alpha >= 1 or self.alpha <= 0) or \
           (self.gamma >= 1 or self.gamma <= 0) or \
           (self.sigma >= 1 or self.sigma <= 0):
            raise ValueError("alpha, gamma, and sigma values must be
strictly between 0 and 1")
```

It should be noted that "lam" is restricted to be strictly greater than one, as $\lambda > 1$. Similarly, "alpha," "gamma," and "sigma" are restricted to being strictly in between 0 and 1 as $\alpha, \gamma, \delta \in (0, 1)$.

➔ *The Value Function*

Recall that for nonnegative outcomes, an agent assigns the value x^{α} for $x \geq 0$. For negative outcomes, the same agent assigns the value $-\lambda(-x)^{\alpha}$ for $x < 1$. This is encoded within the "cpt" class as

```
    def value_function(self, x):
        # Is the outcome a gain?
        if x >= 0:
            return x ** self.alpha
        # Is the outcome a loss?
        else:
            return -self.lam * (-x) ** self.alpha
```

where x is the outcome of the lottery. Note that $v(0) = 0$ holds.

➔ *Weighting Separately Over Gains and Losses*

Kahneman and Tversky define a piecewise, cumulative weighting function consisting of two weighting functions designed to weigh probabilities over gains separately than the probabilities over losses. Recall that cumulative probabilities were weighed as $w^{+}(p_i + \ldots + p_n)$ over gains, $w^{-}(p_{-m} + \ldots + p_i)$ over losses. This could be encoded within the "cpt" class as

```
def prob_weight_plus(self, p):
        if p == 1: return 1
        elif p == 0: return 0
          else: return (p ** self.gamma) / ((p ** self.gamma + (1 - p **
self.gamma)) ** (1 / self.gamma))

    def prob_weight_minus(self, p):
        if p == 1: return 1
        elif p == 0: return 0
          else: return (p ** self.sigma) / ((p ** self.sigma + (1 - p **
self.sigma)) ** (1 / self.sigma))
```

where the function "prob_weight_plus" corresponds to $w^{+}(p)$ and "prob_weight_minus" corresponds to $w^{-}(p)$ where either $-m \leq p \leq i$ (losses) or $i \leq p \leq n$ (gains).

➔ *Cumulative Weighted Probabilities and Expected Value*

Given a lottery and an agent's parameters, Kahneman and Tversky assert that it is possible to capture this agent's opinion about a lottery. Representing the lottery as a two dimensional list, we can traverse through the outcomes and probabilities, taking aggregate sums as we move along. We start by checking if the i^{th} outcome is a gain or a loss. If it is a gain, we take the sum from the i^{th} probability to the n^{th} probability, then again from the $i^{\text{th}} + 1$ probability to the n^{th} probability. The difference between the two aggregate sums is then computed, followed by the product of this weighted probability by the corresponding outcome. This is encoded within the "cpt" class as

```
    def expected_value(self, lottery):
        total = 0
        for i, (xi, pi) in enumerate(lottery):
            if xi >= 0:
                # Defines the aggregate sum from the ith probability to
the nth probability
                w_plus_i = self.prob_weight_plus(sum([p for _, p in
lottery[i:]]))
                # Check if we are at the end of the lottery
                if i < len(lottery) - 1:
                    # Defines the aggregate sum from the ith+1 probability
to the nth probability
                    w_plus_i_next = self.prob_weight_plus(sum([p for _, p
in lottery[i+1:]]))
                    total += self.value_function(xi) * (w_plus_i -
w_plus_i_next)
                else:
                    total += self.value_function(xi) * w_plus_i
```

However, if the i^{th} outcome is a loss we similarly take the sum from the $-\ m^{\text{th}}$ probability to the $i^{\text{th}}+\ 1$ probability, then again from the $-\ m^{\text{th}}$ probability to the i^{th} probability. We then compute the difference between the two aggregate sums and multiply this difference by the corresponding outcome. This is similarly encoded within the "expected_value" function as,

```
            else:
                # Defines the aggregate sum from the -mth probability to
the ith+1 probability
                w_minus_i = self.prob_weight_minus(sum([p for _, p in
lottery[:i+1]]))
                # Check if we are at the end of the lottery
                if i > 0:
                    # Defines the aggregate sum from the -mth probability
to the ith probability
                    w_minus_i_prev = self.prob_weight_minus(sum([p for _,
p in lottery[:i]]))
                    total += self.value_function(xi) * (w_minus_i -
w_minus_i_prev)
                else:
                    total += self.value_function(xi) * w_minus_i
        return total
```

Demonstrations and Conclusions

Our newly constructed simulation enables us to conduct a hypothetical study with a hypothetical subject. For this subject, we assign the parameters the average values obtained from

Kahneman and Tversky's study in 1992. That is $\alpha = 0.88, \gamma = 0.61, \delta = 0.69, \lambda = 2.25$. We do this with the following chunk of code:

```
alpha = 0.88
gamma = 0.61
sigma = 0.69
lam = 2.25
```

With these parameters, we can create an instance of the class "cpt" class, which will act as our agent. We can do this with the following line:

```
agent = cpt(alpha, gamma, sigma, lam)
```

➔ *A Simulative Demonstration of Loss Aversion*

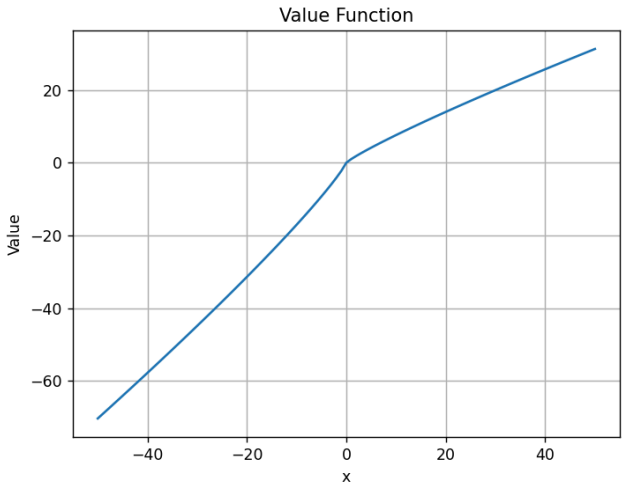
Using the simulation we've constructed, it is possible to demonstrate loss aversion. I will be doing so in two ways:

1) *Graphing the Value function*

Using the python library `matplotlib`, we can graph our value function after importing it as "plt."

```
# Plotting the value function on the x-value axis
x_values = list(range(-50, 51))
y_values = []
for x in x_values:
    y_values.append(agent.value_function(x))
plt.plot(x_values, y_values)
plt.grid(True)
plt.show()
```

With a domain and range of -50 to 51, we pass x-values (arithmetically increasing by 1) to our value function, which returns y-values. We then use "plt" to utilize `matplotlib` and graph our Value Function:



Notice the kink at origin and how much more drastic the convexity over losses is juxtaposed to the concavity over gains. This is numerically exposed by the next demonstration.

2) *Computing the Value of Equivalent Gains/Losses*

With our new agent, when can access its value function and pass a gain of 5 and an equivalent loss of 5 with the following lines:

```
agent.value_function(5))
agent.value_function(-5)
```

When the program is executed, our output is the following:

```
The value assigned to a gain of 5 is  4.121863483573453
The value assigned to a loss of 5 is  -9.274192838040268
       Loss aversion holds here, as v(5) > v(-5)
```

➔ *A Simulative Demonstration of Exaggerating Tail-End Probabilities*

We have allowed our agent to consider gambles in the form of two dimensional arrays, we will now have our agent consider a sure loss, and a highly unlikely, but way more drastic loss:

```
lotteryA = [(-50, 0.001), (0, 0.999)]  # Example lottery [(xi, pi), ...]
lotteryB = [(-5, 1)]
```

We also do this with an equivalent gain:

```
lotteryC = [(50, 0.001), (0, 0.999)]
lotteryD = [(5, 1)]
```

When the program is executed, we get the following output for the expected values of the previous lotteries:

Preferences

$$(E[v(A)] = -0.5988) > (E[v(B)] = -9.2742)$$
$$(E[v(C)] = 0.0191) < (E[v(D)] = 4.1219)$$

It should be noted that when given this gamble, our agent prefers to take a risk over losses, with faith in the probability of losing nothing. Over gains, our agent prefers a certain gain over an unlikely, larger gain. We will now alter the gambles to contain more drastic outcomes. That is,

```
lotteryA = [(-5000, 0.001), (0, 0.999)]  # Example lottery [(xi, pi), ...]
lotteryB = [(-5, 1)]
lotteryC = [(5000, 0.001), (0, 0.999)]
lotteryD = [(5, 1)]
```

When the program is executed, we get the following output for the expected values of the previous lotteries:

Preferences

$$(E[v(A)] = -34.4569) < (E[v(B)] = -9.2742)$$
$$(E[v(C)] = 1.0978) < (E[v(D)] = 4.1219)$$

Notice how the preference is the same over gains, but our agent is no longer comfortable taking a risk over losses, regardless of how unlikely that loss is. This is not only consistent with loss aversion, but it also shows how decision makers tend to exaggerate tail-end probabilities, as the chance of losing money in lottery A is small.

References

- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9601776/
    - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7138240/
- https://journals.sagepub.com/doi/abs/10.1177/0149206310394863?casa_token=JRXqv65MS5gAAAAA:ejU3usri_kBhUPU3Vw4rVHmTTMTWzAACZmv08RM_65zCbGOI3_6Ujn8EfxQ_LCpWaeyHQBlCZ5T1
- https://cear.gsu.edu/files/2020/01/WP_2016_05_Cumulative-Prospect-Theory-in-the-Laboratory-A-Reconsideration-JANUARY-2020.pdf
- https://www.researchgate.net/profile/Baolian-Wang/publication/309476055_Prospect_Theory_and_Stock_Returns_An_Empirical_Test/links/63bd883a03aad5368e7d978a/Prospect-Theory-and-Stock-Returns-An-Empirical-Test.pdf
- https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3706942
- https://repub.eur.nl/pub/23089/OriginalandCumulative.pdf
- https://engineering.purdue.edu/DELP/education/decision_making_slides/Module_12___Cumulative_Prospect_Theory.pdf
    - Slides 24-32
- https://www.jstor.org/stable/41755005?searchText=cumulative+prospect+theory&searchUri=%2Faction%2FdoBasicSearch%3FQuery%3Dcumulative%2Bprospect%2Btheory%26so%3Drel&ab_segments=0%2Fbasic_search_gsv2%2Fcontrol&refreqid=fastly-default%3A4a50380dd7604bfa43b24bf6c34ed910&seq=13