Examples

Shayan Tohidi

2025-06-21

Introduction

The `RSD` package provides an open-source and integrated framework for stochastic dominance (SD) (Hadar and Russell 1969), (Hanoch and Levy 1969), (Rothschild and Stiglitz 1970), (Whitmore 1970), and almost stochastic dominance (ASD) (Leshno and Levy 2002), (Tzeng, Huang, and Shih 2013), (Guo et al. 2013) calculations. They can be used for comparing two probability distributions, considering all information included inside the data to address both outcome and risk. The novelties of this package are:

- implementing ASD algorithms, which allows eliminating extreme utilities and performing with more flexibility than SD rules.
- implementing exact algorithms of SD and ASD instead of numerical estimation, which makes the result more accurate, valid, and reliable.

The `RSD` package focuses on providing the most utilized degrees of SD and ASD, meaning the first- and second-order SD and ASD (AFSD and ASSD, respectively). This package has been developed in `R` environment, and can be accessed and downloaded from CRAN and GitHub.

For general information about stochastic dominance, please read (Levy 2015).

Illustrative Examples

In this section, simple examples are going to be used to explain and understand the concepts and functions' performance.

Class

The backbone of the package is the class `StochasticDominance`. Users can create instances directly with `new` command, or call the implemented constructor `createStochasticDominance`. The latter is recommended because it handles all the calculations needed to populate attributes.

Constructor's input parameters

The input parameters of the constructor correspond to each probability mass function (PMF).

- `outcome1`, `outcome2`: Numerical vectors, include output values of the corresponding distribution.
- `prob1`, `prob2`: Numerical vectors, include probability values of the corresponding distribution.

Instance attributes

The process of creating an instance includes combining both PMFs, calculating CDFs, and SSDs.

- `output`: Numerical vector, created by combining two outcome parameters.
- `prob1`, `prob2`: Numerical vectors. Modified versions of two probability parameters.
- `cdf1`, `cdf2`: Numerical vectors, include CDF values.
- `ssd1`, `ssd2`: Numerical vectors, include SSD values.

```
val1 = c(1,3,5,7)
val2 = c(2,4,6,8)
pr = rep(1/4,4)
sd = createStochasticDominance(val1,val2,pr,pr)
data.frame(outcome = sd@outcome, prob1 = sd@prob1, prob2 = sd@prob2,
           cdf1 = sd@cdf1, cdf2 = sd@cdf2, ssd1 = sd@ssd1, ssd2 = sd@ssd2)
```

```
##   outcome prob1 prob2 cdf1 cdf2 ssd1 ssd2
## 1       1  0.25  0.00 0.25 0.00 0.00 0.00
## 2       2  0.00  0.25 0.25 0.25 0.25 0.00
## 3       3  0.25  0.00 0.50 0.25 0.50 0.25
## 4       4  0.00  0.25 0.50 0.50 1.00 0.50
## 5       5  0.25  0.00 0.75 0.50 1.50 1.00
## 6       6  0.00  0.25 0.75 0.75 2.25 1.50
## 7       7  0.25  0.00 1.00 0.75 3.00 2.25
## 8       8  0.00  0.25 1.00 1.00 4.00 3.00
```

FSD

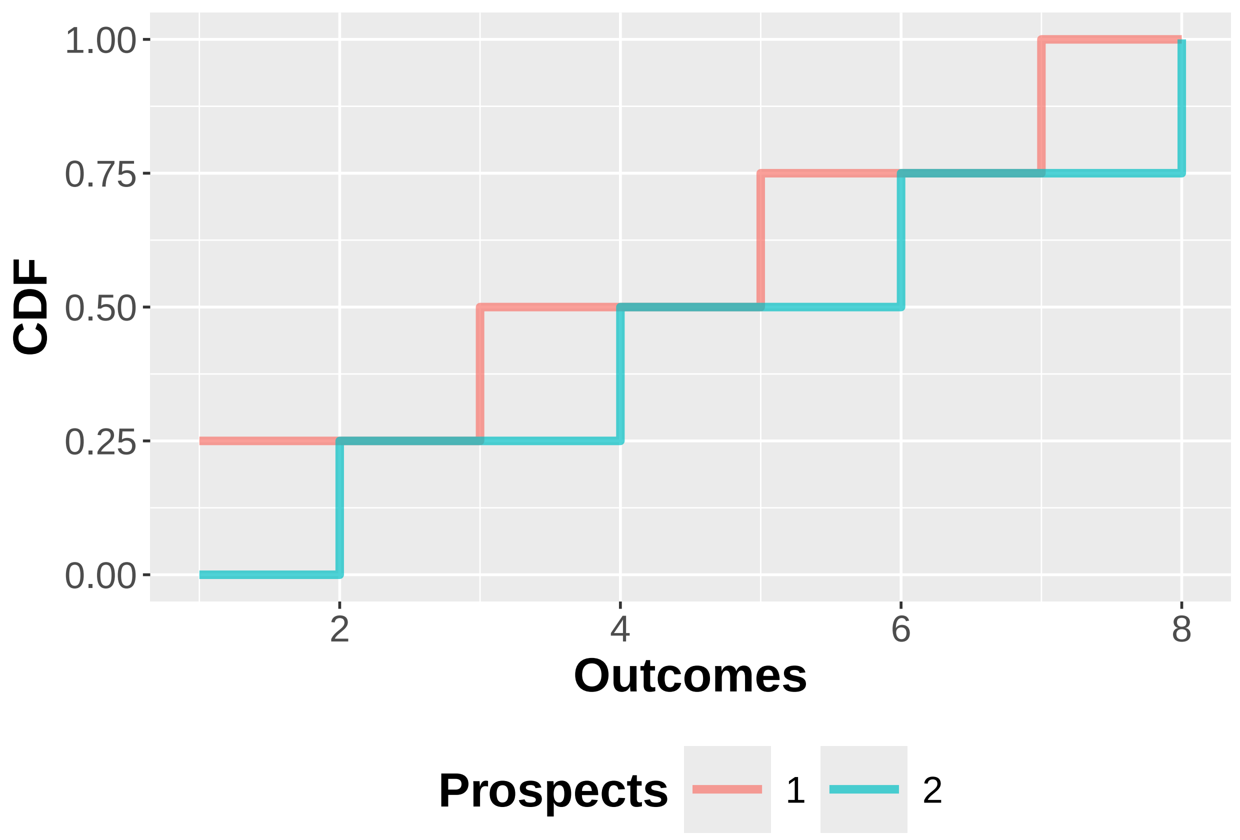
Here, we are going to provide two simple example to investigate the FSD rule and corresponding functions of the package.

To get the index of the dominant distribution based on FSD rule, we call `fsd.test` function and pass the object. Also, `fsd.plot` visualizes the CDF values as the step wise functions.

```
fsd.test(sd)
```

```
## [1] 2
```

```
fsd.plot(sd)
```



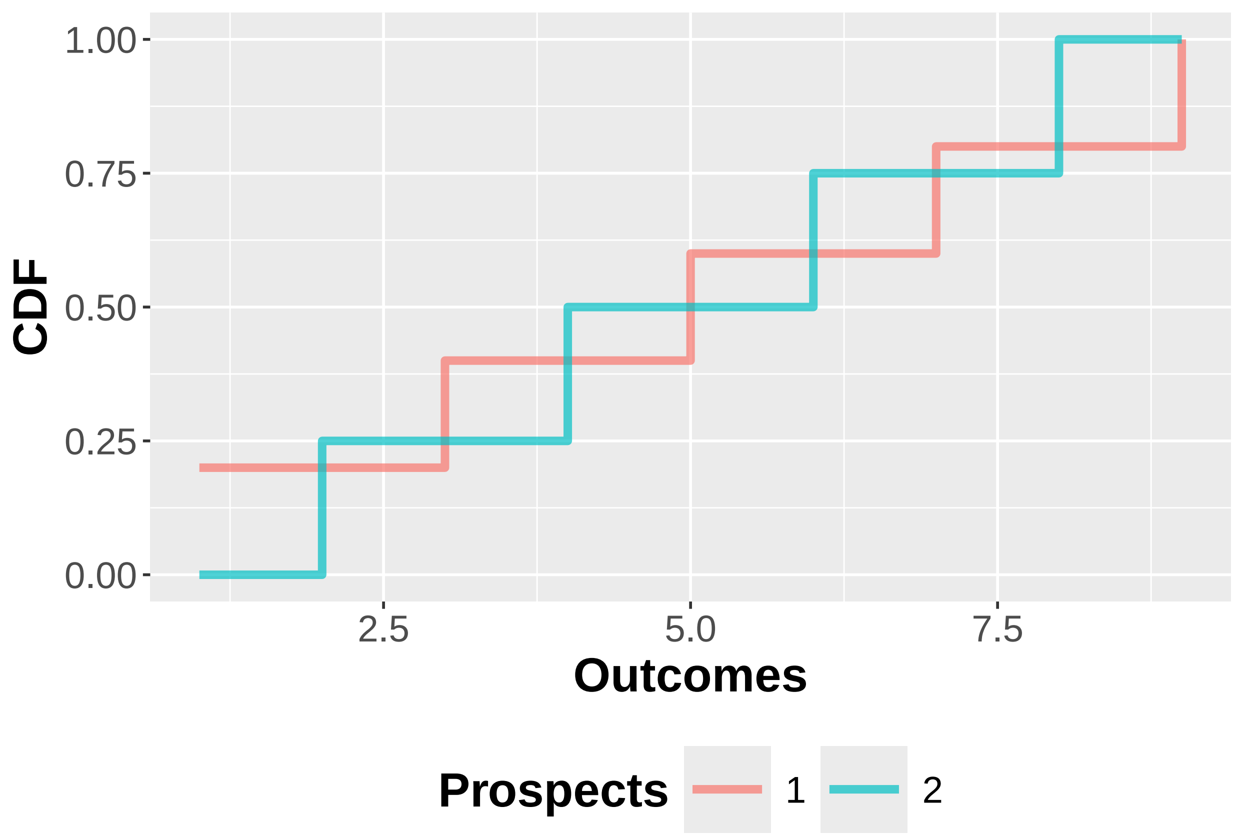
According to the results, the second distribution dominates the other one by FSD, because the CDF values of the second distribution is always less than or equal than that of the first one. It can be seen from the plot above.

However, it does not happen all the time. Below are an example that there exists no FSD domination.

```
val1 = c(1,3,5,7,9)
val2 = c(2,4,6,8)
pr1 = rep(1/5,5)
pr2 = rep(1/4,4)
sd = createStochasticDominance(val1,val2,pr1,pr2)
fsd.test(sd)
```

```
## [1] 0
```

```
fsd.plot(sd)
```



As seen above, the output value of `fsd.test` function is 0, meaning no domination by any of the distributions based on FSD. According to the plot, the CDFs intersect, which proves no domination.

SSD

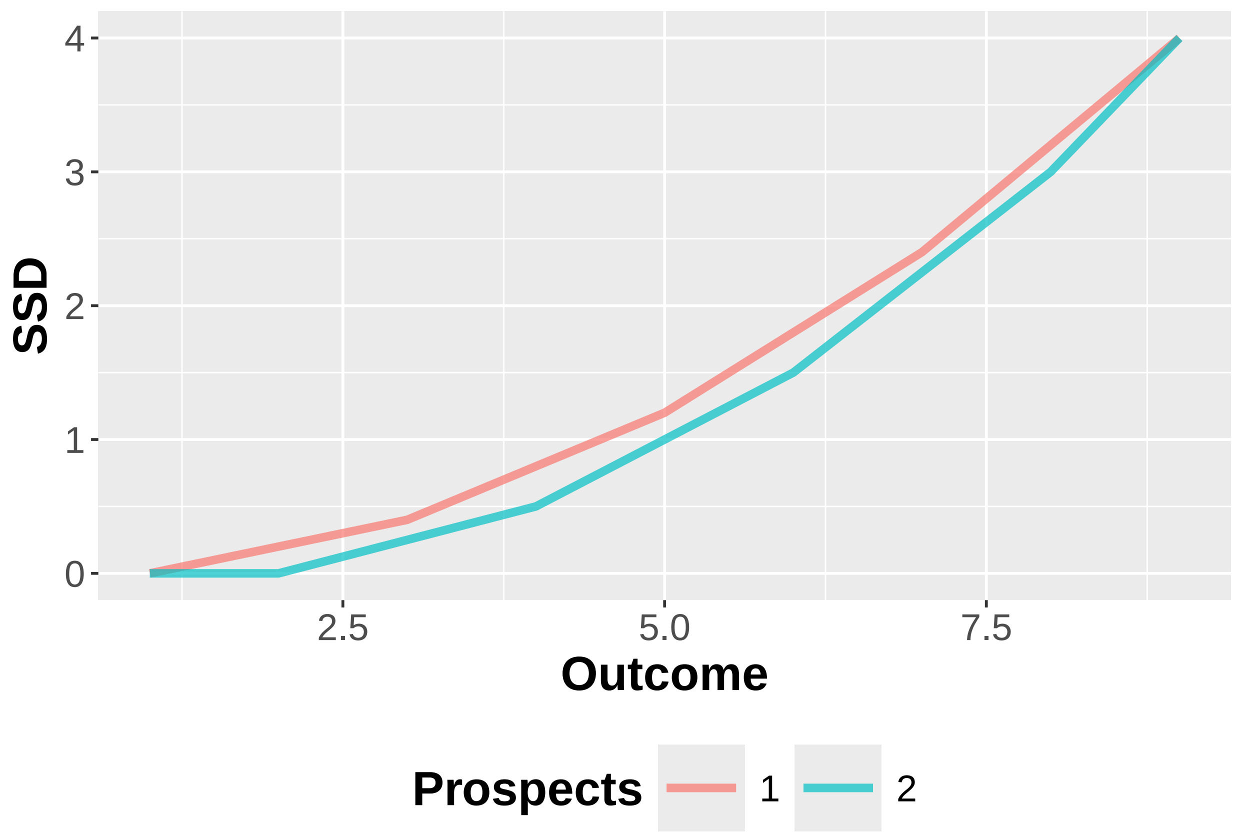
Two examples will be checked here to dive deeper into SSD concept and the `ssd.test` and `ssd.plot` functions.

First, we test the last example from the previous section to check if either distributions dominates the other by SSD.

```
ssd.test(sd)
```

```
## [1] 2
```

```r
ssd.plot(sd)
```

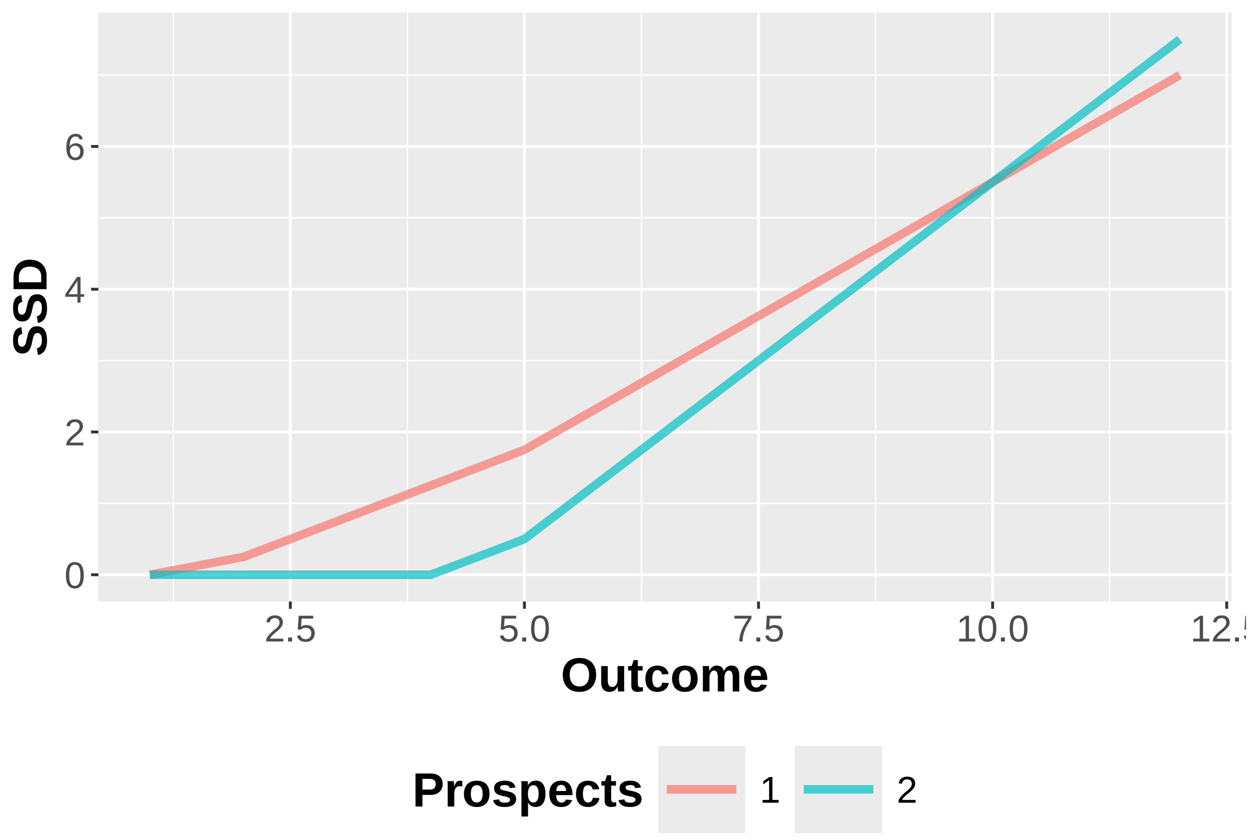


According to the results, the second distribution dominates the first one by SSD. We can see it from the plot, where the SSD plot corresponding to the second distribution is always lower than or equal to that of the first distribution. Now let's look at another example.

```r
val1 = c(1,2,5,12)
val2 = c(4,5)
pr1 = rep(1/4,4)
pr2 = rep(1/2,2)
sd = createStochasticDominance(val1,val2,pr1,pr2)
ssd.test(sd)
```

```
## [1] 0
```

```r
ssd.plot(sd)
```

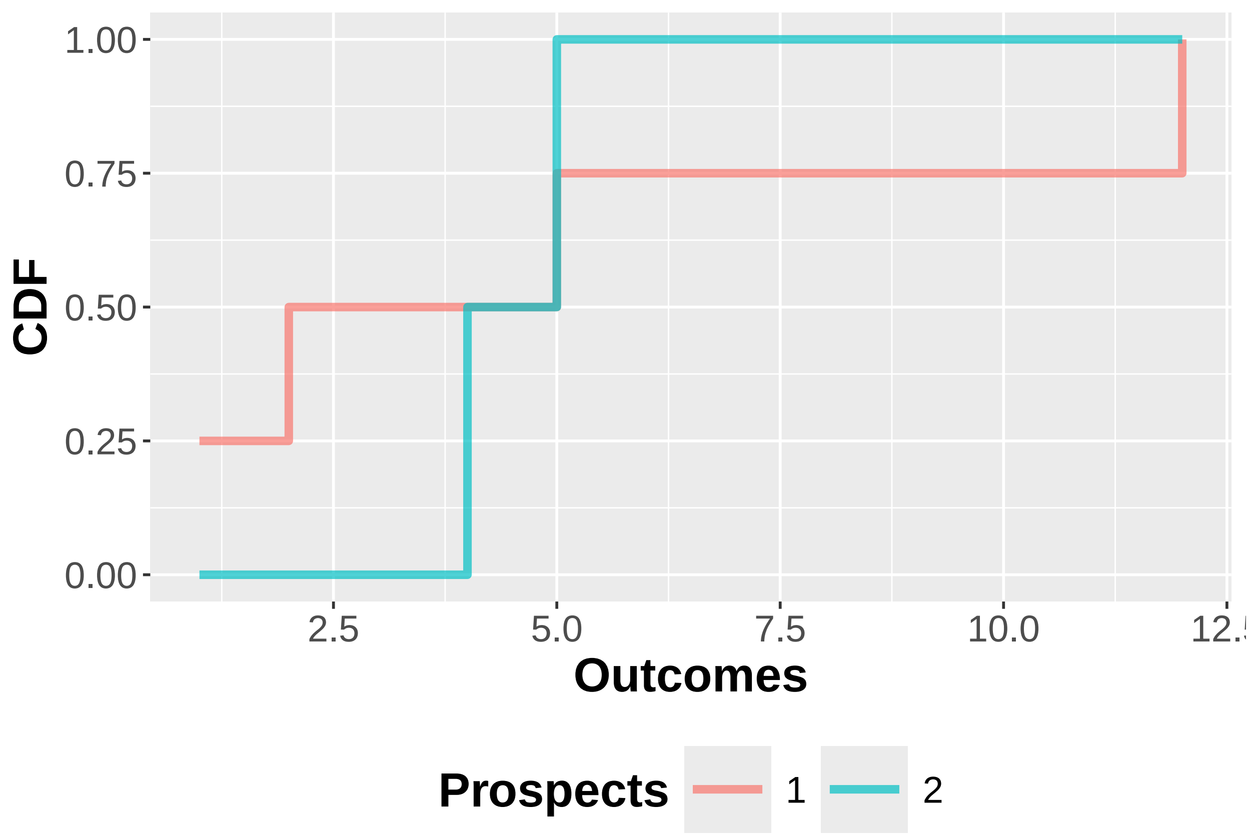


Because the SSD values intersect, there is no SSD domination, and `ssd.test` returns 0.

AFSD

When there is no dominance by FSD, it means all utilities do not agree with a distribution. This often happens in real problems when even a minor violation can make this occurs. In this situation, we can apply AFSD as a relaxed version of FSD where most (not all) utilities agree with a distribution to be dominant. It is achieved by eliminating extreme utilities, and AFSD guarantees to minimize this elimination. It is quantified by the violated area between two CDF plots. If the violated area is shown by $area1 - area2$, The distribution with the minimum violated area value dominates. It this value is positive, the first distribution dominates, otherwise the second one dominates. Let's clarify it with the last example of the previous section, where we know there is no dominance by FSD.

```
fsd.plot(sd)
```



```r
afsd.test(sd)
```

```
## $winner
## [1] 1
##
## $epsilon
## [1] 0.417
##
## $area
## [1]  0.25  1.00  0.00 -1.75
##
## $total.area
## [1] 3
##
## $positive.area
## [1] 1.25
##
## $negative.area
## [1] 1.75
```

According to the output of `afsd.test` function, the `area` is a numeric vector that shows the area difference corresponding to each segment. So, the area of the first distribution between 1 and 2 is 0.25 larger than that of the second one, and 1.75 smaller between 5 and 12. The `total.area` is the total amount of area surrounded by the CDFs, calculated by adding up the absolute values within the `area` vector. The `total.area` can be divided into two other values: `positive.area` that is the summation of the positive values inside the `area` vector, and `negative.area` that is the summation of the negative values amount. If the CDF of the first distribution is always smaller, all values of the `area` vector would be negative and it dominates the other, so the positive values represent the violation for the first distribution. Based on this, the negative values are the violation of the second distribution. The violation ratio is defined by the violation area (which is `positive.area` and `negative.area`) over `total.area`. The `epsilon` indicates the minimum violation ratio and in this example, it corresponds to the positive area, meaning the first distribution has a smaller violation ratio, and dominates by AFSD. It can also be seen by the `winner`, which shows the dominant index. A condition that should be satisfied is $0 < \epsilon < 0.5$. If $\epsilon = 0$, we have FSD and if $\epsilon = 0.5$, the violated areas (`positive.area` and `negative.area`) are equal. This happens if and only if the expected value of two distributions are equal, so the dominant distribution by AFSD (like FSD) always has a higher expected value. In this case, the `winner` would be 0.

ASSD

Like the case with AFSD and FSD, when all risk-averse utilities do not agree with a distribution to dominate that is shown by SSD, a decision-maker can use ASSD, which shows what most risk-averse utilities and decision-makers agree. It is achieved by eliminating or ignoring the most extreme utilities. The concept behind ASSD is exactly like AFSD, which is the minimum violation area. However, there are different ways to calculate violation area, and each represents a version of the ASSD. In this package, the version proposed by (Leshno and Levy 2002) is called **LL**, and the one by (Tzeng, Huang, and Shih 2013) is called **THS**. In the **LL** version, the area between CDFs are considered, but the positive and negative areas are determined by both CDF and SSD values. For example, the positive area is where the CDF and SSD values of the first distribution is larger that those of the second one. In the **THS** version, the area between SSDs are considered, and the positive area is where SSD values of the first distribution is larger than that of the second one. Another condition that should be satisfied here is that the dominant distribution should have higher expected value. So, if the `epsilon` of the distribution with higher expected value is larger than 0.5, there is no domination by ASSD.

```
assd.test(sd, 'll')
```

```
## $winner
## [1] 1
##
## $epsilon
## [1] 0.417
##
## $area
## [1]  0.25  1.00  0.00 -1.25 -0.50
##
## $total.area
## [1] 3
##
## $positive.area
## [1] 1.25
##
## $negative.area
## [1] 0.5
```

Here, the `area` has been changed because the SSDs intersect and there is a new segmentation (we consider the steps in CDFs and all intersections of SSDs). The `total.area` is the same, but the `positive.area` and `negative.area` are different. From 1 to 4, both CDFs and SSDs of the first distribution are larger, so the `positive.area` is 1.25. From 4 to 10, the SSDs of the first distribution are larger, while the CDFs are equal or smaller, so these values are not considered in any of the `positive.area` or `negative.area`. And from 10 to 12, Both values are negative, and the area in that segment is considered in `negative.area`. Based on the output values, both ratios are below zero, so the one with higher expected value will dominate. In this example, the first distribution has higher mean and dominates and the `epsilon` is equal to the ratio of `positive.area` over `total.area`, however the ratio of the `negative.area` is lower.

```
assd.test(sd, 'ths')
```

```
## $winner
## [1] 0
##
## $epsilon
## [1] 0.077
##
## $area
## [1]  0.125  1.500  1.250  3.125 -0.500
##
## $total.area
## [1] 6.5
##
## $positive.area
## [1] 6
##
## $negative.area
## [1] 0.5
```

Here, the `area` is calculated according to the area between SSDs. The `total.area`, `positive.area`, and `negative.area` is calculated in the same way in AFSD. The `epsilon` shows the minimum ratio of `positive.area` or `negative.area` over `total.area`. However, like the other approach, if the expected value of the corresponding distribution is smaller, there will be no dominance. Despite the **LL** version that the ϵ of both distribution may be smaller than 0.5, the ϵ of this **THS** version is smaller than 0.5 for only one of the distributions.

Real Examples

`data_ex`

There exists a data set in this package, called `data_ex`. It is a real data set extracted from (Fields 2020). This data set can be used in multi-environmental analysis in plant breeding, and includes the yield of 13 unique cultivars (genotypes) planted in 29 environments. It is a complete set, meaning all cultivars are planted in all environments.

```
str(data_ex)
```

```
## tibble [377 x 3] (S3: tbl_df/tbl/data.frame)
##  $ gen  : chr [1:377] "2369/LH123HT" "2369/LH123HT" "2369/LH123HT" "2369/LH123HT" ...
##  $ env  : chr [1:377] "ARH12018" "ARH22018" "DEH12018" "GAH12018" ...
##  $ yield: num [1:377] 101 105 123 171 225 ...
```

Using this data set, we will check a scenario where there is no dominance by FSD, but SSD and AFSD can determine the dominated distribution.

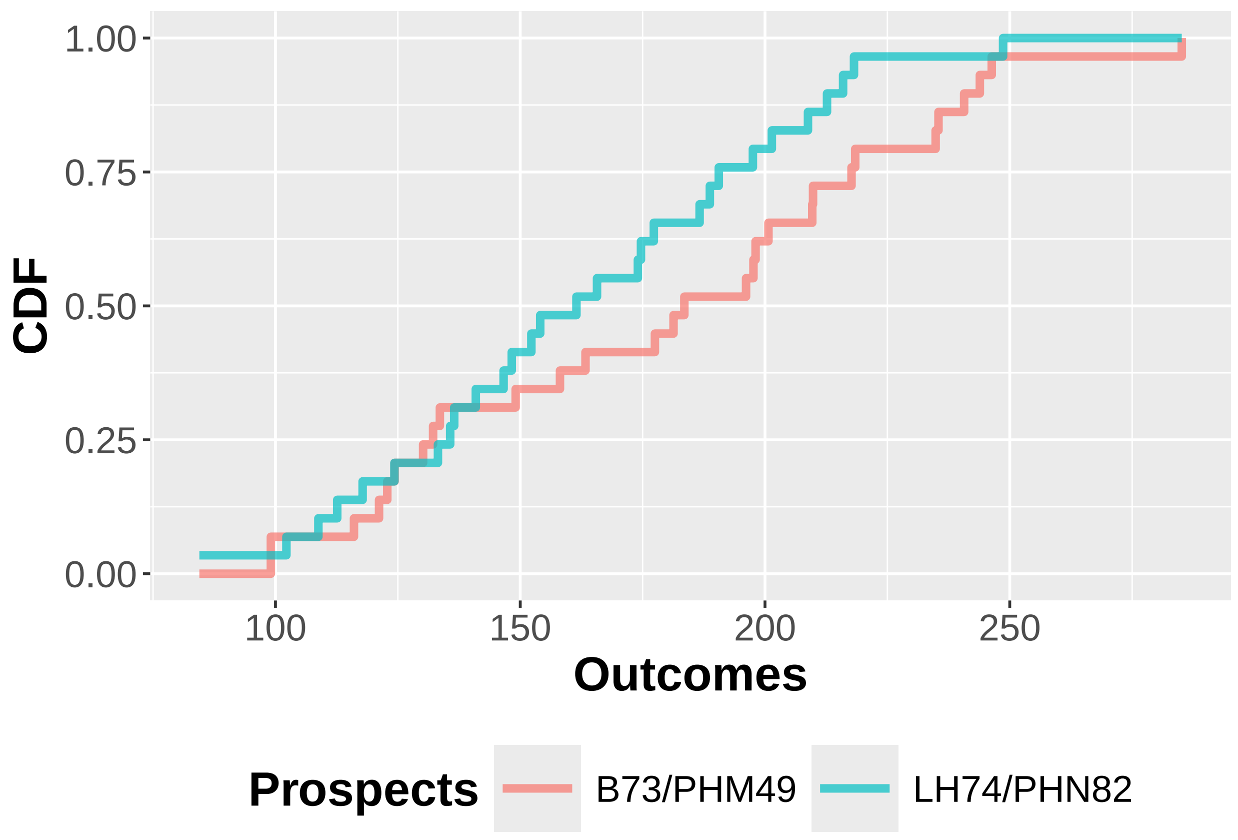
```r
gen1 = 'B73/PHM49'
yield1 = data_ex$yield[data_ex$gen == gen1]
pr1 = rep(1/length(yield1), length(yield1))

gen2 = 'LH74/PHN82'
yield2 = data_ex$yield[data_ex$gen == gen2]
pr2 = rep(1/length(yield2), length(yield2))

sd = createStochasticDominance(yield1, yield2, pr1, pr2)
```

Using the following functions, it can be seen that neither distributions dominates the other by FSD, because they intersect. However, it is obvious that most of the time, the first distribution is below. So, because of a few extreme utilities, the FSD rule has been violated.

```r
fsd.test(sd)
```

```
## [1] 0
```

```r
fsd.plot(sd, names = c(gen1, gen2))
```

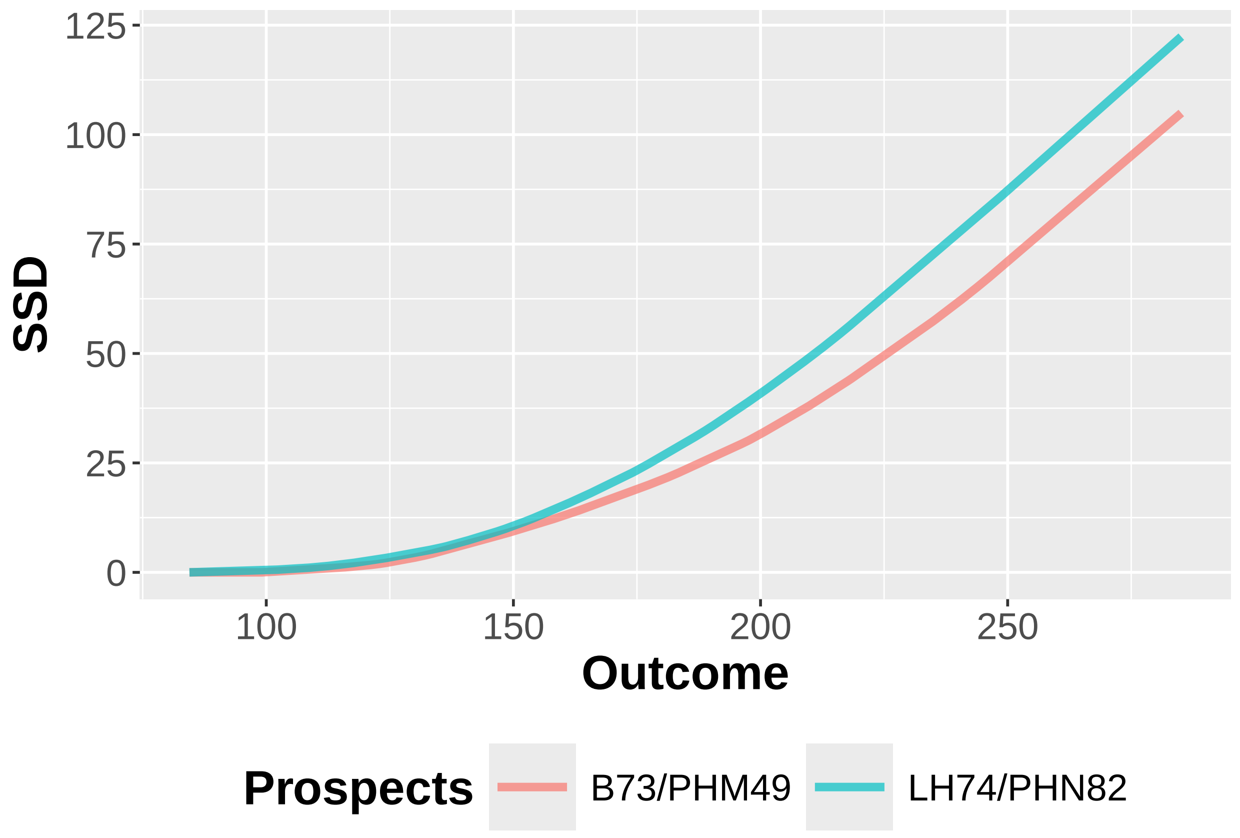


When the FSD rule does not determine, we can use SSD or AFSD. The SSD rule considers all risk-averse utilities, while AFSD considers most utilities with different risk attitudes. Below, the results from SSD rule show that the first distribution (corresponds to 'B73/PHM49' cultivar) dominates.

```r
ssd.test(sd)
```

```
## [1] 1
```

```r
ssd.plot(sd, names = c(gen1, gen2))
```



AFSD rule also shows the same result, that the first distribution dominates with a negligible violation or ϵ.

```r
afsd.test(sd)
```

```
## $winner
## [1] 1
##
## $epsilon
## [1] 0.024
##
## $area
##  [1] -0.503620690  0.109482759  0.000000000 -0.132931034 -0.235862069
##  [6] -0.060862069 -0.231379310 -0.058103448  0.000000000 -0.001494253
## [11]  0.000000000  0.070172414  0.068620690  0.013448276  0.145172414
## [16]  0.028505747  0.000000000 -0.195919540 -0.114396552 -0.082715517
## [21] -0.221379310 -0.187413793 -0.556551724 -0.347327586 -0.255862069
## [26] -0.243362069 -1.151034483 -0.108620690 -0.544137931 -0.053103448
## [31] -0.787241379 -0.381034483 -0.429655172 -0.361206897 -0.376551724
## [36] -1.345689655 -0.288620690 -0.027758621 -0.090517241 -0.453448276
## [41] -0.095517241 -1.272413793 -0.181379310 -0.027298851 -0.393448276
## [46] -0.566522989 -0.355586207 -0.087586207 -0.051724138 -2.831896552
## [51] -0.080689655 -0.541862069 -0.221172414 -0.083793103  0.000000000
## [56] -1.258735632
##
## $total.area
## [1] 18.31283
##
## $positive.area
## [1] 0.4354023
##
## $negative.area
## [1] 17.87743
```

This example shows the importance of ASD as a powerful complementary to the classical SD methods, because in real comparison and ranking problems, the SD methods often fail to determine the dominated distributions.

`shafii.rapeseed`

Another example is provided from `agridat` package named `shafii.rapeseed`, which is a multi-environmental trial of rapeseed in the U.S. and includes 6 cultivars (genotypes) planted in 14 locations and 3 years having 3 replications. The environment (column `env`) can be defined as the combination of year and location (columns `year` and `loc`), respectively. Then, the variability inside each group of cultivars and environments are discarded by averaging out the replications (column `rep`). In this case, these steps make the data set complete, meaning all cultivars (6) are planted in all environments (27). These steps are showing below.

```
df = shafii.rapeseed %>%
  unite(year, loc, col = 'env') %>%
  group_by(gen, env) %>%
  summarise(yield = mean(yield), .groups = 'drop')
```

The list of all cultivars is shown below.

```
cultivars = unique(df$gen)
cultivars
```

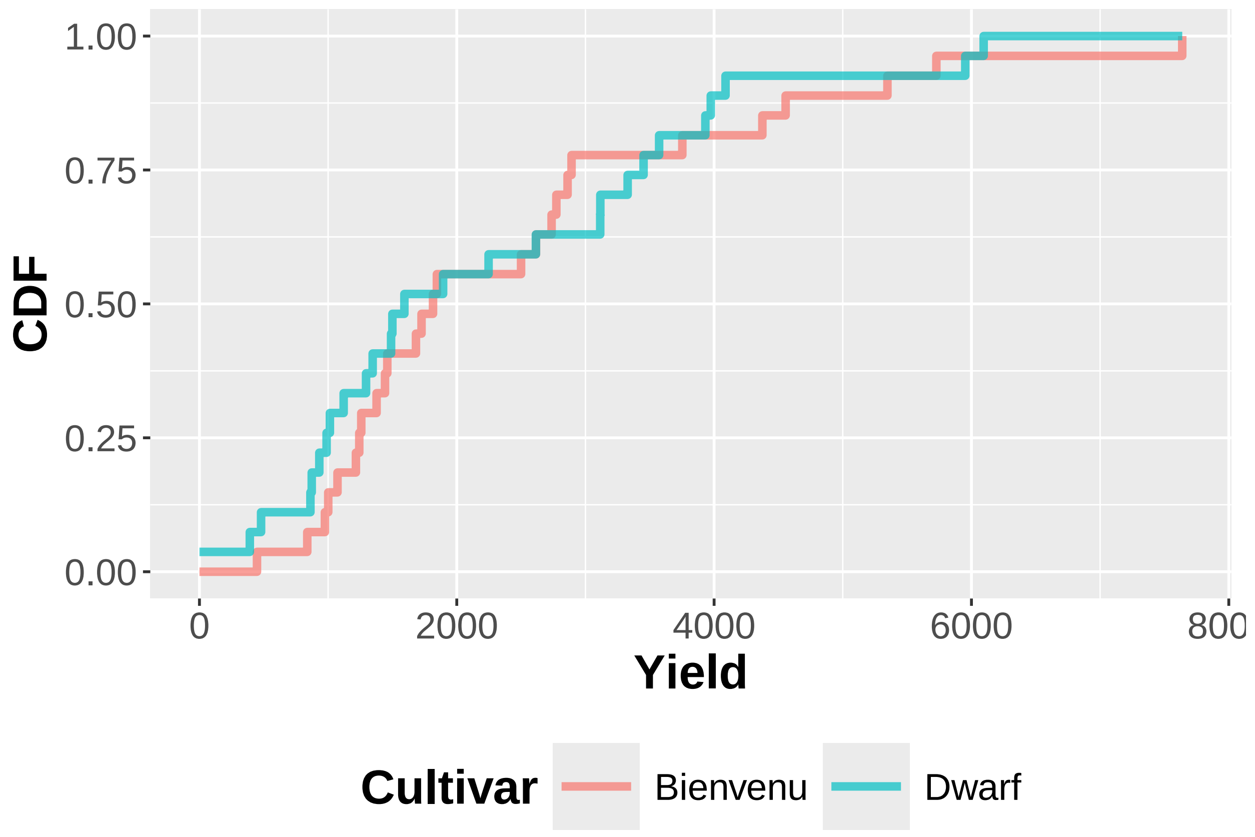
```
## [1] Bienvenu Bridger  Cascade  Dwarf    Glacier  Jet
## Levels: Bienvenu Bridger Cascade Dwarf Glacier Jet
```

Now, suppose we want to perform comparison between Bienvenu and Dwarf, which are the first and fourth in the list of cultivars.

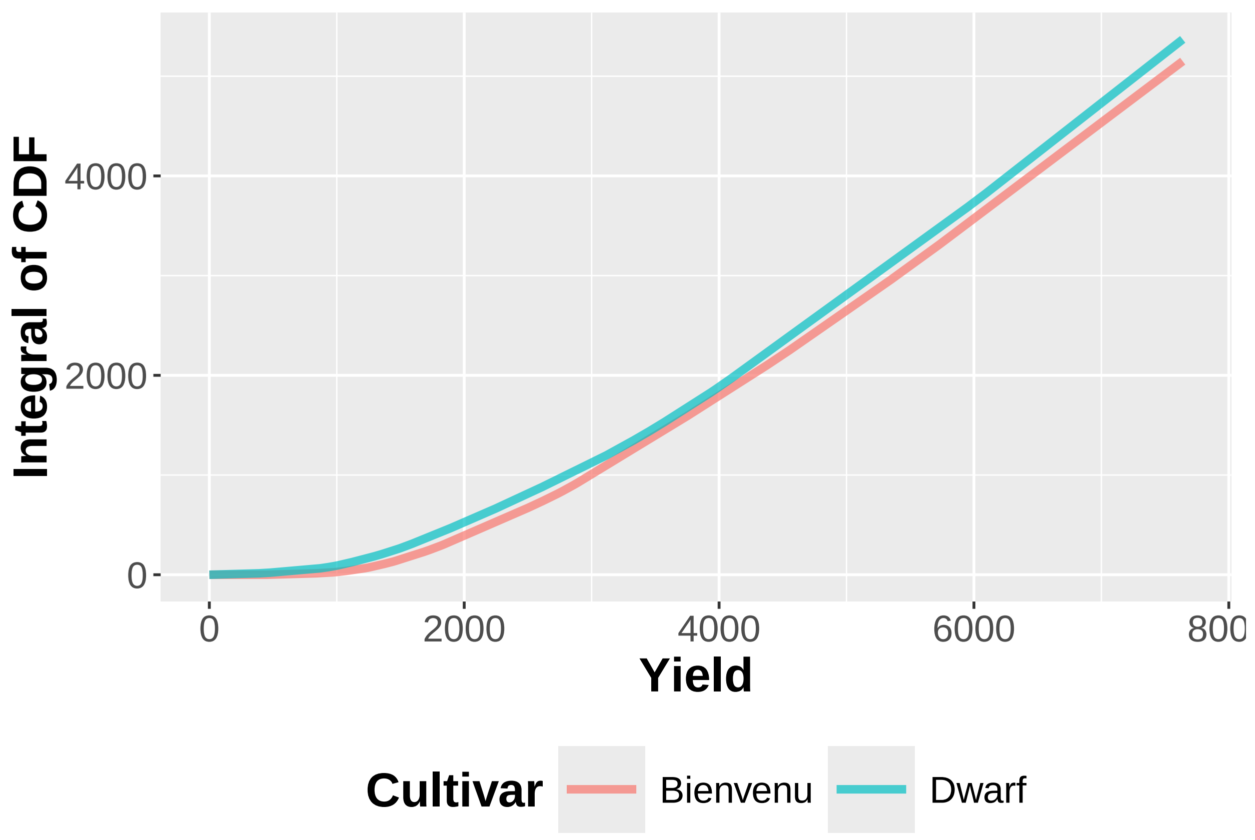
```
pr = rep(1/27, 27)
y1 = df$yield[df$gen == cultivars[1]]
y2 = df$yield[df$gen == cultivars[4]]
pair = createStochasticDominance(y1, y2, pr, pr)
```

Applying `fsd.test` and `ssd.test` functions, it can be seen that Bienvenu dominates by SSD, and not FSD. Below, we shows the plots and how they can be modified.

```r
fsd.plot(pair, names = c('Bienvenu', 'Dwarf')) +
  labs(x = 'Yield', y = 'CDF', color = 'Cultivar')
```



```r
ssd.plot(pair, names = c('Bienvenu', 'Dwarf')) +
  labs(x = 'Yield', y = 'Integral of CDF', color = 'Cultivar')
```



Pipeline

Until now, we have been using our package to perform comparison between only two distributions. Here we are going to present a way for ranking all distributions in a real-world data set. We use `data_ex` in this section as well.

To perform this task, we call `compare.all` function. This function helps us to create all unique unordered pairs of variables (cultivars in this data set) and the corresponding distributions, perform all SD and ASD rules, and build efficient and inefficient sets for each pair by each rule.

The required input parameters are a character vector for variable names, and two numeric vectors for probability and outcome. This information is needed to define all discrete distribution functions. Also, three input parameters can be set to determine the epsilon thresholds for ASD rules. Finally, a Boolean parameter `include.details` is set to specify whether the returned data set has to include distribution information (probability and outcome).

```
result = compare.all(variable = data_ex$gen, probability = rep(1/29, 377),
                     outcome = data_ex$yield, include.details = F)
```

This function wraps three other functions included inside this package: `compare.paired.distributions`, `screen.by.sd`, and `screen.by.asd`. Each of these functions might be called independently to perform some particular tasks. The details of all these functions are presented in package document.

Below, the data set that contains all cultivar pairs alongside with the index of dominated cultivar for every rules and epsilon values for every ASD rules.

```r
result$data
```

```
## # A tibble: 78 x 10
##    variable1 variable2     fsd   ssd  afsd afsd.eps assd.ll assd.ll.eps assd.ths
##    <chr>     <chr>       <dbl> <dbl> <dbl>    <dbl>   <dbl>       <dbl>    <dbl>
##  1 B73/PHM49 PHW52/PHN82     0     0     1    0.468       1       0.327        0
##  2 B73/PHM49 PHG39/PHN82     0     0     1    0.27        1       0.185        1
##  3 B73/PHM49 2369/LH123~     0     0     1    0.268       1       0.097        1
##  4 B73/PHM49 B73/MO17        0     0     1    0.137       1       0.035        1
##  5 B73/PHM49 B73/PHN82       0     0     1    0.161       1       0.099        1
##  6 B73/PHM49 PHW52/PHM49     0     0     1    0.102       1       0.004        1
##  7 B73/PHM49 LH74/PHN82      0     1     1    0.024       1       0            1
##  8 B73/PHM49 F42/MO17        1     1     1    0           1       0            1
##  9 B73/PHM49 F42/H95         1     1     1    0           1       0            1
## 10 B73/PHM49 B37/MO17        1     1     1    0           1       0            1
## # i 68 more rows
## # i 1 more variable: assd.ths.eps <dbl>
```

In addition, `result` contains other elements where they includes efficient and inefficient sets for this data set by applying each SD or ASD rule. Below, the efficient set of the cultivars based on FSD rule is shown.

```r
result$fsd.sets$efficient
```

```
## [1] "B73/PHM49"    "PHW52/PHN82"  "PHG39/PHN82"  "2369/LH123HT" "B73/MO17"
## [6] "B73/PHN82"    "PHW52/PHM49"  "LH74/PHN82"
```

And here, the efficient set of the cultivars based on AFSD rule when its epsilon threshold is set to 0.1 is shown.

```r
result$afsd.sets$efficient
```

```
## [1] "B73/PHM49"    "PHW52/PHN82"  "PHG39/PHN82"  "2369/LH123HT" "B73/MO17"
## [6] "B73/PHN82"
```

According the results above, we observe that the efficient set of AFSD rule has two less cultivars. It means that none of the eight cultivars in the first efficient set, dominates the other ones by FSD. It is also true for the six cultivars in the second efficient set captured by AFSD. In other words, there is at least one cultivar out of these six cultivars that dominates the other two (PHW52/PHM49 and LH74/PHN82) by AFSD with epsilon smaller than 0.1. It shows when we consider all decision makers with all utility functions, there exist someone who pick one of these two cultivars, however they can be ignored because it is rare to observe them in reality or in other word, their utility functions are extreme. In most cases, an ASD rule shrinks the size of the efficient set, which makes deciding easier and this is why decision-makers tend to apply ASD and SD rules together.

Conclusion

The `RSD` package can be used in research and business to compare and rank distributions using stochastic dominance (SD) and almost stochastic dominance (ASD). This package uses exact methods to implement the SD and ASD rules without any estimation because we focus on using discrete distributions instead of continuous ones that have to be sampled. So, the results is reliable. In addition, the algorithms are developed in a way to use vectorized implementation, and have $\mathcal{O}(n)$ in the worst case scenarios. Thus, the functions are efficient and effective even in large data sets.

References

Fields, Genomes to. 2020. "Genomes to Fields 2018 Data Set." CyVerse Data Commons. https://doi.org/https://doi.org/10.25739/anqq-sg86.

Guo, Xu, Xuehu Zhu, Wing-Keung Wong, and Lixing Zhu. 2013. "A Note on Almost Stochastic Dominance." *Economics Letters* 121 (2): 252–56. https://doi.org/https://doi.org/10.1016/j.econlet.2013.08.020.

Hadar, Josef, and William R. Russell. 1969. "Rules for Ordering Uncertain Prospects." *The American Economic Review* 59 (1): 25–34. http://www.jstor.org/stable/1811090.

Hanoch, G., and H. Levy. 1969. "The Efficiency Analysis of Choices Involving Risk." *The Review of Economic Studies* 36 (3): 335–46. http://www.jstor.org/stable/2296431.

Leshno, Moshe, and Haim Levy. 2002. "Preferred by "All" and Preferred by "Most" Decision Makers: Almost Stochastic Dominance." *Management Science* 48 (8): 1074–85. http://www.jstor.org/stable/822676.

Levy, Haim. 2015. *Stochastic Dominance: Investment Decision Making Under Uncertainty.* Springer.

Rothschild, Michael, and Joseph E Stiglitz. 1970. "Increasing Risk: I. A Definition." *Journal of Economic Theory* 2 (3): 225–43. https://doi.org/https://doi.org/10.1016/0022-0531(70)90038-4.

Tzeng, Larry Y., Rachel J. Huang, and Pai-Ta Shih. 2013. "Revisiting Almost Second-Degree Stochastic Dominance." *Management Science* 59 (5): 1250–54. https://doi.org/10.1287/mnsc.1120.1616.

Whitmore, G. A. 1970. "Third-Degree Stochastic Dominance." *The American Economic Review* 60 (3): 457–59. http://www.jstor.org/stable/1817999.