

Brunner-Munzel検定について

- ・ 黒木玄
- ・ 2022-08-05

文献

- E. Brunner and U. Munzel. The nonparametric Behrens-Fisher problem: Asymptotic theory and a small-sample approximation. *Biometrical Journal*, 42:17–25, 2000. [[pdf](https://www.researchgate.net/profile/Edgar-Brunner/publication/264799502_Nonparametric_Hypotheses_and_Rank_Statistics_for_Unbalanced_Factorial_Designs/links/57f_Hypotheses-and-Rank-Statistics-for-Unbalanced-Factorial-Designs.pdf) (https://www.researchgate.net/profile/Edgar-Brunner/publication/264799502_Nonparametric_Hypotheses_and_Rank_Statistics_for_Unbalanced_Factorial_Designs/links/57f_Hypotheses-and-Rank-Statistics-for-Unbalanced-Factorial-Designs.pdf])]
- Karin Neubert and Edgar Brunner, A studentized permutation test for the non-parametric Behrens-Fisher problem, *Computational Statistics and Data Analysis*, Vol. 51, pp. 5192-5204 (2007). <https://doi.org/10.1016/j.csda.2006.05.024> (<https://doi.org/10.1016/j.csda.2006.05.024>)
- Claus P. Nowak, Markus Pauly, Edgar Brunner. The nonparametric Behrens-Fisher problem in small samples. <https://arxiv.org/abs/2208.01231> (<https://arxiv.org/abs/2208.01231>)

目次

- ▼ 1 準備
 - [1.1 パッケージの読み込みなど](#)
 - [1.2 組み合わせの生成子](#)
 - [1.3 Welchのt検定](#)
 - [1.4 単峰型の函数が正の値になる場所を見つける函数](#)
 - [1.5 2つの分布が「互角」になるシフトの仕方を求める函数](#)
- ▼ 2 Brunner-Munzel検定
 - [2.1 Brunner-Munzel検定の実装](#)
 - [2.2 よく使われているっぽいテストデータで正しく実装されているかを確認](#)
 - [2.3 組み合わせの生成子](#)
 - [2.4 Brunner-Munzel検定のpermutation版の実装](#)
 - [2.5 permutation版が正しく実装されているかの確認](#)
- 3 計算例
- ▼ 4 Brunner-Munzel検定とWelchのt検定の比較
 - [4.1 第一種の過誤の確率](#)
 - [4.2 Brunner-Munzel検定は中央値に関する検定ではないことの証拠](#)
 - [4.3 BM検定による互角シフトの信頼区間とWelchのt検定による平均の差の信頼区間の比較](#)
- 5 小サンプルでのpermutation版の検定とBM検定とWelchのt検定の比較

1 準備

1.1 パッケージの読み込みなど

```
In [1]: 1 using Base.Threads
2 using BenchmarkTools
3 using Distributions
4 using PrettyPrinting
5 using QuadGK
6 using Random
7 using RCall
8 using Roots
9 using StatsBase
10 using StatsFuns
11 using StatsPlots
12 default(fmt=:png, size=(400, 250),
13         titlefontsize=10, guidefontsize=8, tickfontsize=6)
14
15 x ≈ y = x < y || x ≈ y
16 x ≈ y = x > y || x ≈ y
17 safemul(x, y) = x == 0 ? x : isinf(x) ? typeof(x)(Inf) : x*y
18 safediv(x, y) = x == 0 ? x : isinf(y) ? zero(y) : x/y
```

Out[1]: safediv (generic function with 1 method)

1.2 組み合わせの生成子

In [2]:

```
1 """
2     nextcombination!(n, t, c = typeof(t)[min(t-1, i) for i in 1:t])
3
4 '[1,2,...,n]' からの重複無しの 't' 個の組み合わせ 'c' をすべて生成したい。
5
6 'nextcombination!(n, t, c)' は配列で表現された組み合わせ 'c' をその次の組み合わせに書き換えて,
7
8 初期条件を 'c = typeof(t)[min(t-1, i) for i in 1:t]' にすると, 'binomial(n, t)' 回の 'nextcomb
9 """
10 function nextcombination!(n, t, c = typeof(t)[min(t-1, i) for i in 1:t])
11     t == 0 && return c
12     @inbounds for i in t:-1:1
13         c[i] += 1
14         c[i] > (n - (t - i)) && continue
15         for j in i+1:t
16             c[j] = c[j-1] + 1
17         end
18         break
19     end
20     c
21 end
22 """
23 """
24 mycombinations!(n::Integer, t, c)
25
26 事前に割り当てられた組み合わせを格納する配列 'c' を使って, '[1,2,...,n]' からの重複無しの 't' 個
27 """
28 function mycombinations!(n::Integer, t, c)
29     for i in 1:t c[i] = min(t - 1, i) end
30     (nextcombination!(n, t, c) for _ in 1:binomial(n, t))
31 end
32 """
33 """
34 mycombinations!(a, t, c)
35
36 事前に割り当てられた組み合わせを格納する配列 'c' を使って, 配列 'a' からのインデックスに重複が
37 """
38 function mycombinations!(a, t, c)
39     t < 0 && (t = length(a) + 1)
40     (view(a, indices) for indices in mycombinations!(length(a), t, c))
41 end
42 """
43 """
44 mycombinations(x, t)
45
46 'x' が整数ならば '[1,2,...,x]' からの, 'x' が配列ならば 'x' からのインデックスに重複がない 't' 个
47 """
48 mycombinations(x, t) = mycombinations!(x, t, Vector{typeof(t)}(undef, t))
```

Out[2]: mycombinations

1.3 Welchのt検定

```
In [3]: 1 function tvalue_welch(m, x̄, sx², n, ȳ, sy²; Δμ=0)
2     (x̄ - ȳ - Δμ) / √(sx²/m + sy²/n)
3 end
4
5 function tvalue_welch(x, y; Δμ=0)
6     m, x̄, sx² = length(x), mean(x), var(x)
7     n, ȳ, sy² = length(y), mean(y), var(y)
8     tvalue_welch(m, x̄, sx², n, ȳ, sy²; Δμ)
9 end
10
11 function degree_of_freedom_welch(m, sx², n, sy²)
12     (sx²/m + sy²/n)² / ((sx²/m)²/(m-1) + (sy²/n)²/(n-1))
13 end
14
15 function degree_of_freedom_welch(x, y)
16     m, sx² = length(x), var(x)
17     n, sy² = length(y), var(y)
18     degree_of_freedom_welch(m, sx², n, sy²)
19 end
20
21 function pvalue_welch(m, x̄, sx², n, ȳ, sy²; Δμ=0)
22     t = tvalue_welch(m, x̄, sx², n, ȳ, sy²; Δμ)
23     v = degree_of_freedom_welch(m, sx², n, sy²)
24     2ccdf(TDist(v), abs(t))
25 end
26
27 function pvalue_welch(x, y; Δμ=0)
28     m, x̄, sx² = length(x), mean(x), var(x)
29     n, ȳ, sy² = length(y), mean(y), var(y)
30     pvalue_welch(m, x̄, sx², n, ȳ, sy²; Δμ)
31 end
32
33 function confint_welch(m, x̄, sx², n, ȳ, sy²; α=0.05)
34     v = degree_of_freedom_welch(m, sx², n, sy²)
35     c = quantile(TDist(v), 1-α/2)
36     SEhat = √(sx²/m + sy²/n)
37     [x̄-ȳ-c*SEhat, x̄-ȳ+c*SEhat]
38 end
39
40 function confint_welch(x, y; α=0.05)
41     m, x̄, sx² = length(x), mean(x), var(x)
42     n, ȳ, sy² = length(y), mean(y), var(y)
43     confint_welch(m, x̄, sx², n, ȳ, sy²; α)
44 end
```

Out[3]: confint_welch (generic function with 2 methods)

1.4 単峰型の函数が正の値になる場所を見つける函数

```
In [4]: 1 function findpositive(f, a, b; maxsplit = 30)
2     @assert f(a) < 0
3     @assert f(b) > 0
4     c = (a + b)/2
5     f(c) > 0 && return c
6     w = b - a
7     for k in 2:maxsplit
8         for d in range(w/2^(k+1), w/2-w/2^(k+1), step=w/2^k)
9             x = c + d
10            f(x) > 0 && return x
11            x = c - d
12            f(x) > 0 && return x
13        end
14    end
15    error("k > maxsplit = $maxsplit")
16 end
17
18 f(x) = abs(x) < 1e-4 ? 1.0 : -1.0
19
20 @time findpositive(f, -100abs(randn()), 20abs(randn()))
```

0.000607 seconds

Out[4]: 7.110625170270168e-5

1.5 2つの分布が「互角」になるシフトの仕方を求める函数

In [5]:

```
1     """
2         prob_x_le_y(distx::UnivariateDistribution, disty::UnivariateDistribution;
3             a = 0.0)
4
5     この 函数は、連続分布 `distx`、`disty` と実数 `a` について、
6     `distx` と `disty` に従って生成される乱数をそれぞれ X, Y と書くとき、
7      $X \leq Y + a$  が成立する確率を返す。
8     """
9     function prob_x_le_y(distx::UnivariateDistribution, disty::UnivariateDistribution,
10         a = 0.0)
11         H(y) = cdf(distx, y) * pdf(disty, y-a)
12         quadgk(H, extrema(disty + a)...)[1]
13     end
14
15     """
16     tieshift(distx::UnivariateDistribution, disty::UnivariateDistribution;
17             p = 0.5)
18
19     この 函数は、連続分布 `distx`、`disty` と実数 `p` について、
20     `distx` と `disty` に従って生成される乱数をそれぞれ X, Y と書くとき、
21      $X \leq Y + a$  が成立する確率が `p` に等しくなるような実数 a を返す。
22     """
23     function tieshift(distx::UnivariateDistribution, disty::UnivariateDistribution;
24             p=0.5)
25         find_zero(a → prob_x_le_y(distx, disty, a) - p, 0.0)
26     end
27
28     @show tieshift(Normal(0, 1), Normal(2, 2))
29     @show tieshift(Normal(0, 1), Laplace(2, 2))
30     @show tieshift(Normal(0, 1), Uniform(0, 1));

```

```
tieshift(Normal(0, 1), Normal(2, 2)) = -1.9999999999999232
tieshift(Normal(0, 1), Laplace(2, 2)) = -1.9999999999994498
tieshift(Normal(0, 1), Uniform(0, 1)) = -0.4999999999999983
```

2 Brunner-Munzel検定

2.1 Brunner-Munzel検定の実装


```

78  \\ &
79  \mathrm{sehat} =
80  \widehat{\mathrm{se}} = \sqrt{\frac{\hat{\sigma}_x^2}{m} + \frac{\hat{\sigma}_y^2}{n}},
81  \\ &
82  \mathrm{tvalue} = t = \frac{\hat{p} - p}{\widehat{\mathrm{se}}},
83  \\ &
84  \mathrm{df} =
85  \nu =
86  \frac{\left(\hat{\sigma}_x^2/m + \hat{\sigma}_y^2/n\right)^2}{\frac{\left(\hat{\sigma}_x^2/m\right)^2}{m-1} +
87  \frac{\left(\hat{\sigma}_y^2/n\right)^2}{n-1}},
88  },
89  \\ &
90  \mathrm{pvalue} =
91  2\mathit{ccdf}(\mathit{TDist}(\nu), |t|).
92  \end{aligned}
93  """
94
95  function statistics_brunner_munzel(X, Y,
96      Hx = similar(X, Float64),
97      Hy = similar(Y, Float64);
98      p = 1/2
99      )
100     m, n = length(X), length(Y)
101     for (i, x) in pairs(X)
102         Hx[i] = sum(h_brunner_munzel(y, x) for y in Y)
103     end
104     for (j, y) in pairs(Y)
105         Hy[j] = sum(h_brunner_munzel(x, y) for x in X)
106     end
107     phat = (mean(Hy) - mean(Hx) + n)/(m + n)
108     sx2, sy2 = var(Hx)/n^2, var(Hy)/m^2
109     sehat = \sqrt(sx2/m + sy2/n)
110     tvalue = (phat - p)/sehat
111     df = safediv((sx2/m + sy2/n)^2, (sx2/m)^2/(m-1) + (sy2/n)^2/(n-1))
112     pvalue = (df != 0 && isfinite(df)) ? 2ccdf(TDist(df), abs(tvalue)) : zero(df)
113     (; phat, sx2, sy2, sehat, tvalue, df, pvalue)
114  end
115
116  @doc raw"""
117      pvalue_brunner_munzel(X, Y,
118          Hx = similar(X, Float64),
119          Hy = similar(Y, Float64);
120          p = 1/2
121      )
122
123  この函数はBrunner-Munzel検定のP値 `pvalue` を返す.
124  """
125
126  function pvalue_brunner_munzel(X, Y,
127      Hx = similar(X, Float64),
128      Hy = similar(Y, Float64);
129      p = 1/2
130      )
131      statistics_brunner_munzel(X, Y, Hx, Hy; p).pvalue
132  end
133
134  """
135
136  tieshift(X::AbstractVector, Y::AbstractVector; p = 1/2)
137
138  この函数は `phat_brunner_munzel(X, Y .+ a)` の値が `p` に等しくなる `a` を返す.
139  """
140  function tieshift(X::AbstractVector, Y::AbstractVector; p = 1/2)
141      shiftmin = minimum(X) - maximum(Y) - 0.1
142      shiftmax = maximum(X) - minimum(Y) + 0.1
143      find_zero(a → phat_brunner_munzel(X, Y .+ a) - p, (shiftmin, shiftmax))
144  end
145
146
147  @doc raw"""
148      brunner_munzel(X, Y,
149          Hx = similar(X, Float64),
150          Hy = similar(Y, Float64),
151          Ytmp = similar(Y, Float64);
152          p = 1/2,
153          α = 0.05,
154          maxsplit = 30
155      )

```

```

156
157 この函数はBrunner-Munzel検定を実行する。 詳細は以下の通り。
158
159 この函数は `phat`, `sehat`, `tvalue`, `df`, `p`, `pvalue`, `α` および \
160 以下達の named tuple を返す。
161
162 ```\math
163 \begin{aligned}
164 & \\
165 \mathrm{confint\_p} = (\text{$p$ の信頼度 $1-\alpha$ の信頼区間}), \\
166 \\ & \\
167 \mathrm{confint\_shift} = (\text{2つの集団が互角になるようなシフトの信頼度 $1-\alpha$ の信頼区間}), \\
168 \\ & \\
169 \mathrm{pvalue\_shift} = (\mathrm{confint\_shift} の計算で使われたP値函数), \\
170 \\ & \\
171 \mathrm{shiftthat} = (\text{2つの集団が互角になるようなシフトの点推定値}). \\
172 \end{aligned}
173
174
175 さらに, $\mathrm{shiftmin}$, $\mathrm{shiftmax}$ はデータから推定されるシフトの下限と上限。
176
177 """
178 function brunner_munzel(X, Y,
179     Hx = similar(X, Float64),
180     Hy = similar(Y, Float64),
181     Ytmp = similar(Y, Float64);
182     p = 1/2,
183     α = 0.05,
184     maxsplit = 30
185 )
186     (; phat, sehat, tvalue, df, pvalue) = statistics_brunner_munzel(X, Y, Hx, Hy; p)
187     c = df == 0 ? Inf : quantile(TDist(df), 1 - α/2)
188     confint_p = [max(0, phat - c*sehat), min(1, phat + c*sehat)]
189
190     function pvalue_shift(a)
191         @. Ytmp = Y + a
192         pvalue_brunner_munzel(X, Ytmp, Hx, Hy; p)
193     end
194     shiftmin = minimum(X) - maximum(Y) - 0.1
195     shiftmax = maximum(X) - minimum(Y) + 0.1
196     shifthat = tieshift(X, Y; p)
197     confint_shift = [
198         find_zero(a → pvalue_shift(a) - α, (shiftmin, shifthat))
199         find_zero(a → pvalue_shift(a) - α, (shifthat, shiftmax))
200     ]
201
202     (; phat, sehat, tvalue, df, p, pvalue, α, confint_p,
203         confint_shift, pvalue_shift, shifthat, shiftmin, shiftmax)
204 end
205
206
207 function show_plot_brunner_munzel(X, Y,
208     Hx = similar(X, Float64),
209     Hy = similar(Y, Float64),
210     Ytmp = similar(Y, Float64);
211     p = 1/2,
212     α = 0.05,
213     showXY = false,
214     kwargs...
215 )
216     showXY && (@show X Y)
217     (; phat, sehat, tvalue, df, p, pvalue, α, confint_p,
218         confint_shift, pvalue_shift, shifthat, shiftmin, shiftmax) =
219         brunner_munzel(X, Y, Hx, Hy, Ytmp; p, α)
220     pprint(; phat, sehat, tvalue, df, p, pvalue, α, confint_p,
221         confint_shift, shifthat))
222     println()
223     @show median(X) median(Y)
224     plot(pvalue_shift, shiftmin, shiftmax; label="")
225     vline!([tieshift(X, Y)]; label="", ls=:dash)
226     title!("P-value function of shift")
227     plot!(ytick=0:0.05:1)
228     plot!(; kwargs...)
229 end

```

Out[6]: `show_plot_brunner_munzel (generic function with 4 methods)`

```
In [7]: 1 @doc h_brunner_munzel
```

```
Out[7]: h_brunner_munzel(x, y)
```

この函数は, $x < y$ のとき 1.0 を, $x = y$ のとき 0.5 を返す.

```
In [8]: 1 @doc phat_brunner_munzel
```

```
Out[8]: phat_brunner_munzel(X, Y)
```

まず以下のようにおく:

$$H(x, y) = \begin{cases} 1 & (x < y) \\ 1/2 & (x = y), \end{cases}$$
$$m = \text{length}(X), \quad n = \text{length}(Y), \quad x_i = X[i], \quad y_j = Y[j]$$

この函数は次の \hat{p} を返す:

$$\hat{p} = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n H(x_i, y_j).$$

```
In [9]: 1 @doc statistics_brunner_munzel
```

```
Out[9]: statistics_brunner_munzel(X, Y,
    Hx = similar(X, Float64),
    Hy = similar(Y, Float64);
    p = 1/2
)
```

この函数はデータ X , Y について, Brunner-Munzel検定関係の統計量達を計算する. 詳細は以下の通り.

函数 $H(x, y)$ と $\hat{p}, H_i^x, H_j^y, \bar{H}^x, \bar{H}^y$ を次のように定める:

$$\begin{aligned}m &= \text{length}(X), \quad n = \text{length}(Y), \quad x_i = X[i], \quad y_j = Y[j], \\ \hat{p} &= \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n H(x_i, y_j), \\ H(x, y) &= \begin{cases} 1 & (x < y) \\ 1/2 & (x = y), \end{cases} \\ H_i^x &= \sum_{j=1}^n H(y_j, x_i), \quad H_j^y = \sum_{i=1}^m H(x_i, y_j), \\ \bar{H}^x &= \frac{1}{m} \sum_{i=1}^m H_i^x = n - np\hat{p}, \\ \bar{H}^y &= \frac{1}{n} \sum_{j=1}^n H_j^y = m\hat{p}. \end{aligned}$$

この函数は以下達の named tuple で返す:

$$\begin{aligned}\text{phat} &= \hat{p} = \frac{\bar{H}^x - \bar{H}^y + n}{m + n}, \\ \text{sx2} &= \hat{\sigma}_x^2 = \frac{1}{n^2} \frac{1}{m-1} \sum_{i=1}^m (H_i^x - \bar{H}^x)^2, \\ \text{sy2} &= \hat{\sigma}_y^2 = \frac{1}{m^2} \frac{1}{n-1} \sum_{j=1}^n (H_j^y - \bar{H}^y)^2, \\ \text{sehat} &= \widehat{\text{se}} = \sqrt{\frac{\hat{\sigma}_x^2}{m} + \frac{\hat{\sigma}_y^2}{n}}, \\ \text{tvalue} &= t = \frac{\hat{p} - p}{\widehat{\text{se}}}, \\ \text{df} &= \nu = \frac{\left(\hat{\sigma}_x^2/m + \hat{\sigma}_y^2/n\right)^2}{\frac{\left(\hat{\sigma}_x^2/m\right)^2}{m-1} + \frac{\left(\hat{\sigma}_y^2/n\right)^2}{n-1}}, \\ \text{pvalue} &= 2\text{ccdf}(\text{TDist}(\nu), |t|). \end{aligned}$$

```
In [10]: 1 @doc brunner_munzel
```

```
Out[10]: brunner_munzel(X, Y,
    Hx = similar(X, Float64),
    Hy = similar(Y, Float64),
    Ytmp = similar(Y, Float64);
    p = 1/2,
    α = 0.05,
    maxsplit = 30
)
```

この函数はBrunner-Munzel検定を実行する。詳細は以下の通り。

この函数は `phat`, `sehat`, `tvalue`, `df`, `p`, `pvalue`, `α` および以下達の named tuple を返す。

`confint_p` = (p の信頼度 $1 - \alpha$ の信頼区間),
`confint_shift` = (2つの集団が互角になるようなシフトの信頼度 $1 - \alpha$ の信頼区間),
`pvalue_shift` = (\$`confint_shift$`の計算で使われた P 値函数),
`shifthat` = (2つの集団が互角になるようなシフトの点推定値)。

さらに, `shiftmin`, `shiftmax` はデータから推定されるシフトの下限と上限。

```
In [11]:
```

```
1 X = randn(10)
2 Y = randn(10)
3 @show shiftmin = minimum(X) - maximum(Y) - 1
4 @show shiftmax = maximum(X) - minimum(Y) + 1
5 pvalue_brunner_munzel(X, Y)
```

```
shiftmin = (minimum(X) - maximum(Y)) - 1 = -3.3979011222264486
shiftmax = (maximum(X) - minimum(Y)) + 1 = 4.7013322085735325
```

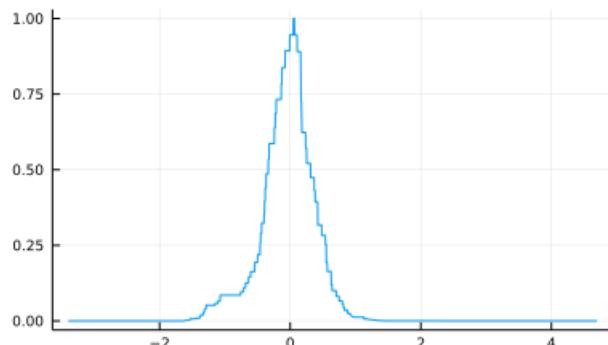
```
Out[11]: 0.9457952516578215
```

```
In [12]:
```

```
1 plot(a → pvalue_brunner_munzel(X, Y .+ a), shiftmin, shiftmax;
      label="", title="P-value function of shift")
```

```
Out[12]:
```

P-value function of shift



2.2 よく使われているつぽいテストデータで正しく実装されているかを確認

<https://okumuralab.org/~okumura/stat/brunner-munzel.html> (<https://okumuralab.org/~okumura/stat/brunner-munzel.html>)

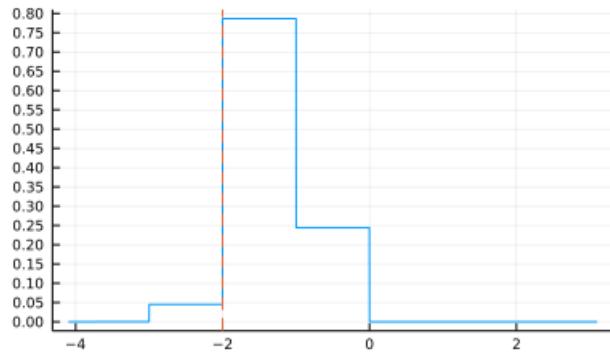
```
x = c(1,2,1,1,1,1,1,1,1,2,4,1,1)
y = c(3,3,4,3,1,2,3,1,1,5,4)
brunnermunzel.test(x, y)

data: x and y
Brunner-Munzel Test Statistic = 3.1375, df = 17.683, p-value = 0.005786
95 percent confidence interval:
 0.5952169 0.9827052
sample estimates:
P(X<Y)+.5*P(X=Y)
 0.788961
```

```
In [13]: 1 X = [1,2,1,1,1,1,1,1,1,1,2,4,1,1]
2 Y = [3,3,4,3,1,2,3,1,1,5,4]
3 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.788961038961039,
sehat = 0.09210009046816862,
tvalue = 3.1374674823029505,
df = 17.682841979481545,
p = 0.5,
pvalue = 0.005786208666151463,
alpha = 0.05,
confint_p = [0.5952168642537363, 0.9827052136683416],
confint_shift = [-2.0000000000000004, -5.551115123125783e-17],
shiftthat = -1.9999999999999998)
median(X) = 1.0
median(Y) = 3.0
```

Out[13]: P-value function of shift



```
In [14]: 1 X = [1,2,1,1,1,1,1,1,1,1,2,4,1,1]
2 Y = [3,3,4,3,1,2,3,1,1,5,4]
3 @rput X Y
4 R"""
5 library(lawstat)
6 brunner.munzel.test(X, Y)
7 """
```

Out[14]: RObject{VecSxp}

```
Brunner-Munzel Test

data: X and Y
Brunner-Munzel Test Statistic = 3.1375, df = 17.683, p-value = 0.005786
95 percent confidence interval:
 0.5952169 0.9827052
sample estimates:
P(X<Y)+.5*P(X=Y)
 0.788961
```

このように Brunner-Munzel 検定は R では [lawstat](https://cran.r-project.org/package=lawstat) (<https://cran.r-project.org/package=lawstat>) パッケージの [brunner.munzel.test](https://rdrr.io/cran/lawstat/man/brunner.munzel.test.html) (<https://rdrr.io/cran/lawstat/man/brunner.munzel.test.html>) で使える。

2.3 組み合わせの生成子

```
In [15]: 1 """
2     complementcomb!(complcomb::AbstractVector, comb::AbstractVector)
3
4 'comb' が {1,2,...,N} から重複無しに m 個を選ぶ組み合わせを表す配列であり, 'comb' の中で数は小さ
5 このとき, この函数は配列 'complcomb' に配列 'comb' の補集合を格納し, 'complcomb' を返す.
6
7 この函数はメモリ割り当てゼロで実行される.
8 """
9
10 function complementcomb!(complcomb::AbstractVector, comb::AbstractVector)
11     N = length(comb) + length(complcomb)
12     k = 0
13     a = 0
14     @inbounds for b in comb
15         for i in a+1:b-1
16             k += 1
17             complcomb[k] = i
18         end
19         a = b
20     end
21     @inbounds for i in a+1:N
22         k += 1
23         complcomb[k] = i
24     end
25     complcomb
26 end
27 """
28     complementcomb(N, comb::AbstractVector)
29
30 'comb' が {1,2,...,N} から重複無しに m 個を選ぶ組み合わせを表す配列であり, 'comb' の中で数は小さ
31 この函数は 'comb' の補集合の配列を返す.
32
33 この函数は返り値の配列の分だけのメモリ割り当てを行う.
34 """
35
36 complementcomb(N, comb::AbstractVector) =
37     complementcomb!(similar(comb, N - length(comb)), comb)
```

Out[15]: complementcomb

In [16]: 1 @doc complementcomb!

Out[16]: complementcomb!(complcomb::AbstractVector, comb::AbstractVector)

comb が {1,2,...,N} から重複無しに m 個を選ぶ組み合わせを表す配列であり, comb の中で数は小さな順に並んでいるとし, complcomb は長さ N - m の配列であると仮定する.

このとき, この函数は配列 complcomb に配列 comb の補集合を格納し, complcomb を返す.

この函数はメモリ割り当てゼロで実行される.

In [17]: 1 @doc complementcomb

Out[17]: complementcomb(N, comb::AbstractVector)

comb が {1,2,...,N} から重複無しに m 個を選ぶ組み合わせを表す配列であり, comb の中で数は小さな順に並んでいると仮定する.

この函数は comb の補集合の配列を返す.

この函数は返り値の配列の分だけのメモリ割り当てを行う.

In [18]: 1 N = 10
2 comb = [2, 4, 5, 8]
3 ccomb = similar(comb, N - length(comb))
4 @btime complementcomb!(\$ccomb, \$comb);

13.727 ns (0 allocations: 0 bytes)

```
In [19]: 1 N, m = 5, 3
          2 ccomb = Vector{Int}(undef, N-m)
          3 [(copy(comb), copy(complementcomb!(ccomb, comb))) for comb in mycombinations(1:N, m)]
```

```
Out[19]: 10-element Vector{Tuple{Vector{Int64}, Vector{Int64}}}:
([1, 2, 3], [4, 5])
([1, 2, 4], [3, 5])
([1, 2, 5], [3, 4])
([1, 3, 4], [2, 5])
([1, 3, 5], [2, 4])
([1, 4, 5], [2, 3])
([2, 3, 4], [1, 5])
([2, 3, 5], [1, 4])
([2, 4, 5], [1, 3])
([3, 4, 5], [1, 2])
```

```
In [20]: 1 N, m = 5, 3
          2 ccomb = Vector{Int}(undef, N-m)
          3 [(copy(comb), complementcomb(N, comb)) for comb in mycombinations(1:N, m)]
```

```
Out[20]: 10-element Vector{Tuple{Vector{Int64}, Vector{Int64}}}:
([1, 2, 3], [4, 5])
([1, 2, 4], [3, 5])
([1, 2, 5], [3, 4])
([1, 3, 4], [2, 5])
([1, 3, 5], [2, 4])
([1, 4, 5], [2, 3])
([2, 3, 4], [1, 5])
([2, 3, 5], [1, 4])
([2, 4, 5], [1, 3])
([3, 4, 5], [1, 2])
```

2.4 Brunner-Munzel検定のpermutation版の実装

In [21]:

```
1 """
2     permutation_tvalues_brunner_munzel(X, Y,
3         XandY = Vector{Float64}(undef, length(X)+length(Y)),
4         Tval = Vector{Float64}(undef, binomial(length(X)+length(Y), length(X))),
5         Hx = similar(X, Float64),
6         Hy = similar(Y, Float64)
7     )
8
9 Brunner-Munzel検定のt値を '[X; Y]' から \
10 インデックスの重複無しに 'length(X)' 個取る組み合わせと \
11 その補集合への分割のすべてについて計算して, 'Tval' に格納して返す.
12 """
13 function permutation_tvalues_brunner_munzel(X, Y,
14     XandY = Vector{Float64}(undef, length(X)+length(Y)),
15     Tval = Vector{Float64}(undef, binomial(length(X)+length(Y), length(X))),
16     Hx = similar(X, Float64),
17     Hy = similar(Y, Float64),
18     ccomb = Vector{Int}(undef, length(Y))
19 )
20 m, n = length(X), length(Y)
21 N = m + n
22 @views XandY[1:m] .= X
23 @views XandY[m+1:N] .= Y
24 for (k, comb) in enumerate(mycombinations(1:N, m))
25     complementcomb!(ccomb, comb)
26     Tval[k] = statistics_brunner_munzel(
27         view(XandY, comb), view(XandY, ccomb), Hx, Hy).tvalue
28 end
29 Tval
30 end
31 """
32 pvalue_brunner_munzel_perm(X, Y,
33     Tval = permutation_tvalues_brunner_munzel(X, Y),
34     tval = statistics_brunner_munzel(X, Y).tvalue;
35     le = ≈
36 )
37
38 Brunner-Munzel検定のpermutation版のP値を返す.
39 """
40 function pvalue_brunner_munzel_perm(X, Y,
41     Tval = permutation_tvalues_brunner_munzel(X, Y),
42     tval = statistics_brunner_munzel(X, Y).tvalue;
43     le = ≈
44 )
45     pvalue_perm = mean(T → le(abs(tval), abs(T)), Tval)
46 end
47 end
```

Out[21]: pvalue_brunner_munzel_perm

In [22]: 1 @doc permutation_tvalues_brunner_munzel

```
Out[22]: permutation_tvalues_brunner_munzel(X, Y,
    XandY = Vector{Float64}(undef, length(X)+length(Y)),
    Tval = Vector{Float64}(undef, binomial(length(X)+length(Y), length(X))),
    Hx = similar(X, Float64),
    Hy = similar(Y, Float64)
)
```

Brunner-Munzel検定のt値を [X; Y] からインデックスの重複無しに length(X) 個取る組み合わせとその補集合への分割のすべてについて計算して, Tval に格納して返す.

In [23]: 1 @doc pvalue_brunner_munzel_perm

```
Out[23]: pvalue_brunner_munzel_perm(X, Y,
    Tval = permutation_tvalues_brunner_munzel(X, Y),
    tval = statistics_brunner_munzel(X, Y).tvalue;
    le = ≈
)
```

Brunner-Munzel検定のpermutation版のP値を返す.

```

bm = brunner.munzel.test(x, y)$statistic
n1 = length(x)
n2 = length(y)
N = n1 + n2
xandy = c(x, y)
foo = function(X) {
  brunner.munzel.test(xandy[X], xandy[-X])$statistic
}
z = combn(1:N, n1, foo)
mean(abs(z) >= abs(bm))

```

結果は 0.008037645 となりました。

In [24]:

```

1 X = [1,2,1,1,1,1,1,1,1,1,2,4,1,1]
2 Y = [3,3,4,3,1,2,3,1,1,5,4]
3 @show X Y
4 @show m, n = length(X), length(Y)
5
6 Tval = @time permutation_tvalues_brunner_munzel(X, Y)
7 @show pvalue_brunner_munzel_perm(X, Y, Tval)
8 stephist(Tval; norm=true, bin=101, label="", title="permutation t-values")

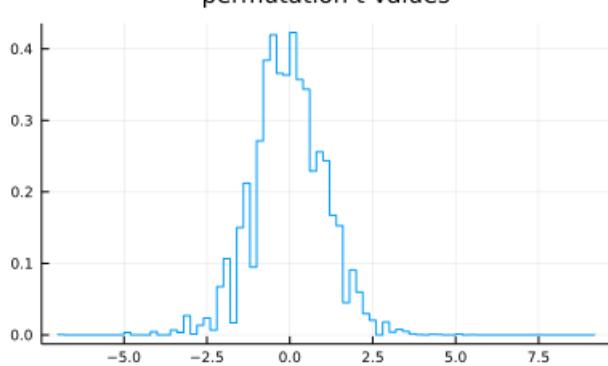
```

```

X = [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 4, 1, 1]
Y = [3, 3, 4, 3, 1, 2, 3, 1, 1, 5, 4]
(m, n) = (length(X), length(Y)) = (14, 11)
4.820885 seconds (904.65 k allocations: 80.359 MiB, 0.50% gc time, 5.46% compilation time)
pvalue_brunner_munzel_perm(X, Y, Tval) = 0.008037645264055279

```

Out[24]:



2.5 permutation版が正しく実装されているかの確認

- <https://github.com/toshi-ara/brunnermunzel/issues/14> (<https://github.com/toshi-ara/brunnermunzel/issues/14>)
- <https://github.com/toshi-ara/brunnermunzel/files/4395032/mwe.R.zip> (<https://github.com/toshi-ara/brunnermunzel/files/4395032/mwe.R.zip>)

追記 2022-08-06: <https://twitter.com/TA25140989/status/1555825941451923457> (<https://twitter.com/TA25140989/status/1555825941451923457>) を参照せよ。

<https://github.com/toshi-ara/brunnermunzel/tree/development> (<https://github.com/toshi-ara/brunnermunzel/tree/development>) の修正版の brunnermunzel パッケージをインストールし直した。以下のセルの実行結果が変わるはずなので、その記録を残しておく。

以下の記録を見なくても、

- <https://github.com/genkuroki/public/blob/e4faafc52721b63876b3b705f9450eade3c902f5/0034/Brunner-Munzel.ipynb> (<https://github.com/genkuroki/public/blob/e4faafc52721b63876b3b705f9450eade3c902f5/0034/Brunner-Munzel.ipynb>)

で閲覧できるが、わざわざ見に行くのも面倒なのでこのファイルにも記録を残しておく。

以前の実行結果:

```
@show pval_J - pval_J_le;
```

```
idx = @show findall(pval_J .!= pval_J_le)  
length(idx)
```

```
findall(pval_J .!= pval_J_le) = [3, 4, 5, 6, 9, 10, 12, 13, 15, 19, 21, 22, 23, 25, 27, 3  
2, 34, 36, 40, 42, 44, 47, 48, 49, 50, 51, 53, 55, 68, 69, 71, 78, 79, 80, 82, 86, 89, 90,  
92, 94]
```

40

```
@show pval_R - pval_J_le;
```

```
pval_R - pval_J_le = [0.0, 0.0, 0.007936507936507936, 0.023809523809523808, 0.023809523809523808, 0.0793650793650793, 0.0, 0.0, 0.007936507936507936, 0.0357142857142857, 0.0, 0.015873015873015928, 0.015873015873015872, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.023809523809523836, 0.007936507936507908, 0.011904761904761918, 0.0, 0.015873015873015928, 0.0, 0.003968253968253968, 0.0, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.00793650793650802, 0.0, 0.015873015873015928, 0.0, 0.0, 0.0, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.0, 0.007936507936507936, 0.007936507936507936, 0.011904761904761918, 0.007936507936507936, 0.03968253968253965, 0.0, 0.007936507936507936, 0.0, 0.05555555555555547, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015873015873015872, 0.015873015873015872, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.007936507936507908, 0.007936507936507964, 0.007936507936507964, 0.0, 0.03968253968253954, 0.0, 0.0, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0357142857142857, 0.015873015873015872, 0.0, 0.023809523809523725, 0.0, 0.023809523809523808, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
idx = @show findall(pval_R .!= pval_J_le)
length(idx)
```

```
findall(pval_R .!= pval_J_le) = [3, 4, 5, 6, 9, 10, 12, 13, 15, 19, 21, 22, 23, 25, 27, 3  
2, 34, 36, 40, 42, 44, 47, 48, 49, 50, 51, 53, 55, 68, 69, 78, 79, 80, 82, 86, 89, 90, 92,  
94]
```

39

```
@show pval_J - pval_R;
```

```
idx = @show findall(.!{pval_J ≈ pval_R})  
length(idx)
```

```
findall(.!{pval_J .≈ pval_R}) = [22, 34, 71, 78]
```

4

```
all(pval_J .≥ pval_R .≥ pval_J_le)
```

true

以前のコメントは以下の通り:

なるほど！

<https://github.com/toshi-ara/brunnermunzel/issues/14>

に書いてあるように，22, 34, 71 and 78 の4つで，値が一致していない。

○○以下または○○以上の判定を \$x \approx y\$ のときも true にする必要があるのだが，その部分で違いが生じているものと思われる。

現時点では <https://CRAN.R-project.org/package=brunnermunzel> にアクセスすると，

```
>Package ‘brunnermunzel’ was removed from the CRAN repository.  
>  
>Formerly available versions can be obtained from the [archive](https://cran.r-project.org/src/contrib/Archive/brunnermunzel/).  
>  
>Archived on 2022-03-04 as check problems were not corrected in time. , LENGTH_1 checks.  
>  
>A summary of the most recent check results can be obtained from the [check results archive](https://cran-archive.r-project.org/web/checks/2022/2022-03-04\_check\_results\_brunnermunzel.html).  
>  
>Please use the canonical form https://CRAN.R-project.org/package=brunnermunzel to link to  
this page.
```

と表示される。

In [25]:

```
1 R"""  
2 library(brunnermunzel)  
3 set.seed(1290)  
4 reps = 100  
5 xx = c()  
6 yy = c()  
7 pval_R = numeric(reps)  
8 for (i in seq_len(reps)){  
9   x = rnorm(5)  
10  y = rnorm(5)  
11  
12  xx = c(xx, x)  
13  yy = c(yy, y)  
14  
15  res_bm_perm ← brunnermunzel.permutation.test(x,y)  
16  pval_R[i] ← res_bm_perm$p.value  
17 }  
18 """  
19  
20 @rget xx yy pval_R  
21 XX = reshape(xx, 5, 100)  
22 YY = reshape(yy, 5, 100)  
23  
24 pval_J = zeros(100)  
25 pval_J_le = zeros(100)  
26 for i in 1:100  
27   pval_J[i] = pvalue_brunner_munzel_perm(XX[ :,i], YY[ :,i]; le = ≈)  
28   pval_J_le[i] = pvalue_brunner_munzel_perm(XX[ :,i], YY[ :,i]; le = ≤)  
29 end
```

```
In [26]: 1 @show pval_J - pval_J_le;
```

```
pval_J - pval_J_le = [0.0, 0.0, 0.007936507936507936, 0.023809523809523808, 0.023809523809523808, 0.0793650793650793, 0.0, 0.0, 0.007936507936507936, 0.0357142857142857, 0.0, 0.015873015873015928, 0.015873015873015872, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.023809523809523836, 0.015873015873015928, 0.011904761904761918, 0.0, 0.015873015873015928, 0.0, 0.003968253968253968, 0.0, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.015873015873015928, 0.0, 0.015873015873015928, 0.0, 0.0, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.0, 0.007936507936507936, 0.007936507936507936, 0.011904761904761918, 0.007936507936507936, 0.03968253968253965, 0.0, 0.007936507936507936, 0.0, 0.05555555555555547, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015873015873015872, 0.015873015872, 0.0, 0.015873015873015928, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015873015873015928, 0.007936507964, 0.007936507936507964, 0.0, 0.003968253968253954, 0.0, 0.0, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0, 0.0357142857142857, 0.015873015873015872, 0.0, 0.023809523809523725, 0.0, 0.023809523808, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
In [27]: 1 idx = @show findall(pval_J .!= pval_J_le)
2 length(idx)
```

```
findall(pval_J .!= pval_J_le) = [3, 4, 5, 6, 9, 10, 12, 13, 15, 19, 21, 22, 23, 25, 27, 32, 34, 36, 40, 42, 44, 47, 48, 49, 50, 51, 53, 55, 68, 69, 71, 78, 79, 80, 82, 86, 89, 90, 92, 94]
```

Out[27]: 40

```
In [28]: 1 @show pval_R - pval_J_le;
```

```
pval_R - pval_J_le = [0.0, 0.0, 0.007936507936507936, 0.023809523809523808, 0.023809523809523808, 0.0793650793650793, 0.0, 0.0, 0.007936507936507936, 0.0357142857142857, 0.0, 0.015873015873015928, 0.015873015873015872, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.023809523809523836, 0.015873015873015928, 0.011904761904761918, 0.0, 0.015873015873015928, 0.0, 0.003968253968253968, 0.0, 0.0, 0.0, 0.0, 0.007936507936507936, 0.0, 0.015873015873015928, 0.0, 0.015873015873015928, 0.0, 0.0, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.007936507936507908, 0.0, 0.0, 0.007936507936507936, 0.007936507936507936, 0.011904761904761918, 0.007936507936507936, 0.03968253968253965, 0.0, 0.007936507936507936, 0.0, 0.05555555555555547, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015873015873015872, 0.015873015873015872, 0.0, 0.015873015873015928, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015873015873015928, 0.007936507964, 0.007936507936507964, 0.0, 0.003968253968253954, 0.0, 0.0, 0.0, 0.015873015873015872, 0.0, 0.0, 0.0, 0.0357142857142857, 0.015873015873015872, 0.0, 0.023809523809523725, 0.0, 0.023809523808, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
In [29]: 1 idx = @show findall(pval_R .!= pval_J_le)
2 length(idx)
```

```
findall(pval_R .!= pval_J_le) = [3, 4, 5, 6, 9, 10, 12, 13, 15, 19, 21, 22, 23, 25, 27, 32, 34, 36, 40, 42, 44, 47, 48, 49, 50, 51, 53, 55, 68, 69, 71, 78, 79, 80, 82, 86, 89, 90, 92, 94]
```

Out[29]: 40

```
In [30]: 1 @show pval_J - pval_R;
```

```
In [31]: 1 idx = @show findall(.!{pval_J .≈ pval_R})
2 length(idx)
```

```
findall(.!(pval_J .≈ pval_R)) = Int64[]
```

Out[31]: 0

```
In [32]: 1 all(pval_J .≥ pval_R .≥ pval_J_le)
```

Out[32]: true

2022-08-06: やった! 値が完全に一致した! permutation版Brunner-Munzel検定について、

- <https://github.com/toshi-ara/brunnermunzel/tree/development> (<https://github.com/toshi-ara/brunnermunzel/tree/development>)

の実装と私による実装の計算結果は以上の場合において完全に一致している。

3 計算例

In [33]:

```
1 m, n = 10, 10
2 X, Y = rand(Normal(0, 1), m), rand(Normal(0, 2), n)
3 @show pval_brmu = pvalue_brunner_munzel(X, Y)
4 @show pval_perm = pvalue_brunner_munzel_perm(X, Y);

pval_brmu = pvalue_brunner_munzel(X, Y) = 0.18972278377606908
pval_perm = pvalue_brunner_munzel_perm(X, Y) = 0.1742730953257269
```

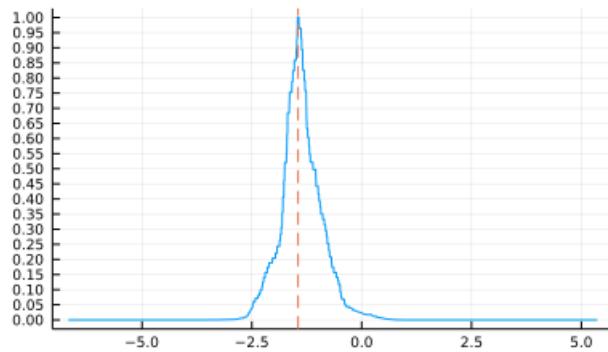
In [34]:

```
1 Random.seed!(4)
2
3 m, n = 10, 20
4 X, Y = rand(Normal(0, 1), m), rand(Normal(0, 2), n)
5 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.75,
sehat = 0.10056855914111178,
tvalue = 2.485866379463735,
df = 18.385702920172452,
p = 0.5,
pvalue = 0.022742714988590425,
α = 0.05,
confint_p = [0.5390305665719614, 0.9609694334280386],
confint_shift = [-2.4770370893730127, -0.3717172074086195],
shiftthat = -1.4469029166494445)
median(X) = -0.5058729691113518
median(Y) = 1.1395412005826189
```

Out[34]:

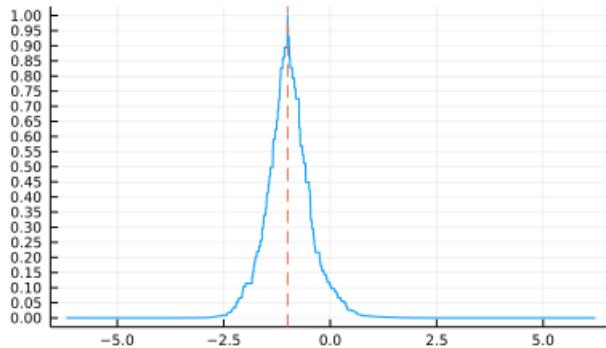
P-value function of shift



```
In [35]: 1 m, n = 10, 20
2 X, Y = rand(Normal(0, 1), m), rand(Normal(0, 2), n)
3 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.675,
sehat = 0.1049296617650541,
tvalue = 1.667783895004246,
df = 27.705176944214784,
p = 0.5,
pvalue = 0.10662428006873317,
α = 0.05,
confint_p = [0.45995823964489857, 0.8900417603551015],
confint_shift = [-2.2060635904773678, 0.3296238740154929],
shifthat = -0.999368724114565)
median(X) = -0.17409123783026362
median(Y) = 0.8382134149102431
```

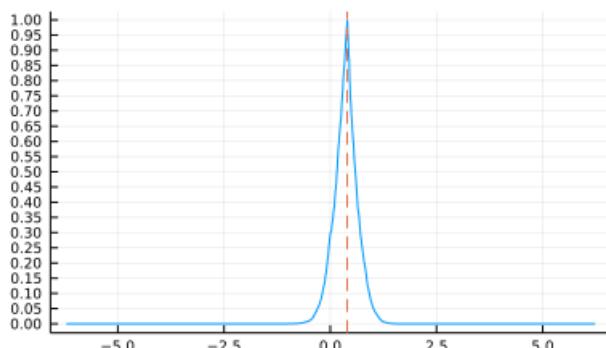
Out[35]: P-value function of shift



```
In [36]: 1 m, n = 100, 50
2 X, Y = rand(Normal(0, 1), m), rand(Normal(0, 2), n)
3 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.4397999999999997,
sehat = 0.05714122667226539,
tvalue = -1.0535300606211744,
df = 60.575330843208235,
p = 0.5,
pvalue = 0.2962824213806768,
α = 0.05,
confint_p = [0.3255228528192483, 0.5540771471807516],
confint_shift = [-0.3011419367780337, 1.0179208275627627],
shifthat = 0.3967840840930079)
median(X) = -0.0536449473157313
median(Y) = -0.6163649344394873
```

Out[36]: P-value function of shift



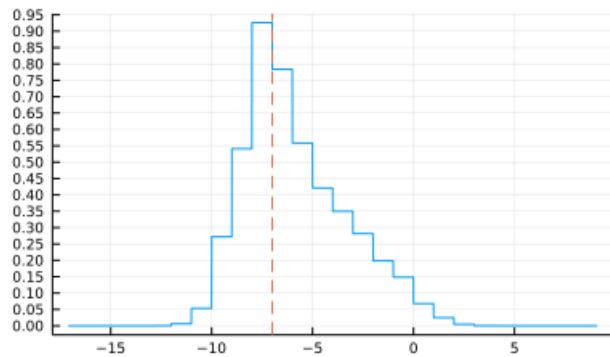
In [37]:

```
1 Random.seed!(4)
2
3 m, n = 10, 20
4 X, Y = rand(1:m, m), rand(1:n, n)
5 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.6725,
sehat = 0.1005725567999867,
tvalue = 1.715179622439735,
df = 20.71835365424782,
p = 0.5,
pvalue = 0.10123190720539166,
α = 0.05,
confint_p = [0.46317465971191163, 0.8818253402880883],
confint_shift = [-10.999999999999998, 0.9999999999999997],
shifthat = -7.000000000000001)
median(X) = 6.5
median(Y) = 14.5
```

Out[37]:

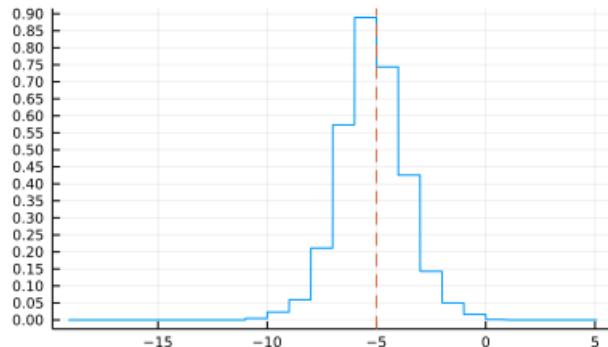
P-value function of shift



```
In [38]: 1 m, n = 10, 20
2 X, Y = rand(1:m, m), rand(1:n, n)
3 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.765,
sehat = 0.08626255311649104,
tvalue = 3.0720166564295583,
df = 27.60349766038036,
p = 0.5,
pvalue = 0.004741546491682432,
α = 0.05,
confint_p = [0.5881847509519754, 0.9418152490480246],
confint_shift = [-8.999999999999998, -1.999999999999993],
shifthat = -5.0)
median(X) = 5.5
median(Y) = 12.0
```

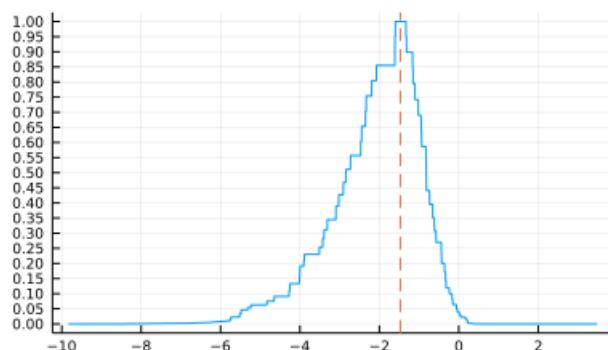
Out[38]: P-value function of shift



```
In [39]: 1 distx, disty = LogNormal(), LogNormal(1)
2 m, n = 10, 10
3 X, Y = rand(distx, m), rand(disty, n)
4 show_plot_brunner_munzel(X, Y)
```

```
(phat = 0.75,
sehat = 0.11205157542647741,
tvalue = 2.231115439907736,
df = 18.0,
p = 0.5,
pvalue = 0.03863103202434314,
α = 0.05,
confint_p = [0.5145883755427825, 0.9854116244572175],
confint_shift = [-5.318179432610124, -0.05841508479073754],
shifthat = -1.4733410643825495)
median(X) = 1.3089930416807416
median(Y) = 2.9790059563220423
```

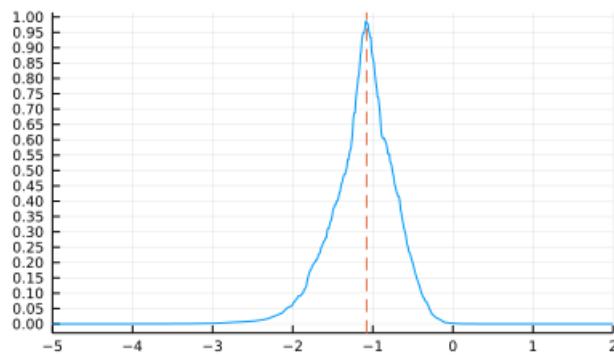
Out[39]: P-value function of shift



```
In [40]: 1 distx, disty = LogNormal(), LogNormal(1)
2 m, n = 40, 40
3 X, Y = rand(distx, m), rand(disty, n)
4 show_plot_brunner_munzel(X, Y; xlim=(-5, 2))
```

```
(phat = 0.6912499999999999,
sehat = 0.05960002903974512,
tvalue = 3.208891053936604,
df = 75.84080882915966,
p = 0.5,
pvalue = 0.001953574321225763,
α = 0.05,
confint_p = [0.5725422250913458, 0.8099577749086541],
confint_shift = [-2.07997163873439, -0.29464656357396524],
shifthat = -1.0768936945378669)
median(X) = 1.044779302249417
median(Y) = 2.2527593566191237
```

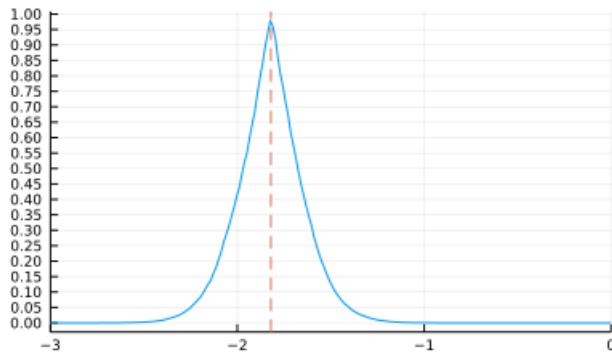
Out[40]: P-value function of shift



```
In [41]: 1 distx, disty = LogNormal(), LogNormal(1)
2 m, n = 160, 160
3 X, Y = rand(distx, m), rand(disty, n)
4 show_plot_brunner_munzel(X, Y; xlim=(-3, 0))
```

```
(phat = 0.79234375,
sehat = 0.025077139980214335,
tvalue = 11.657778767062629,
df = 315.6587283148897,
p = 0.5,
pvalue = 2.3287816342563994e-26,
α = 0.05,
confint_p = [0.7430042840809774, 0.8416832159190226],
confint_shift = [-2.250313266911184, -1.404197630202607],
shifthat = -1.8203983580938916)
median(X) = 0.8417596982690712
median(Y) = 2.974128627007357
```

Out[41]: P-value function of shift



4 Brunner-Munzel検定とWelchのt検定の比較

4.1 第一種の過誤の確率

In [42]:

```

1 function sim_brunner_mumzel();
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10,
3     L = 10^6)
4     pval_bm = Vector{Float64}(undef, L)
5     tmpX = [Vector{Float64}(undef, m) for _ in 1:nthreads()]
6     tmpY = [Vector{Float64}(undef, n) for _ in 1:nthreads()]
7     tmpHx = [Vector{Float64}(undef, m) for _ in 1:nthreads()]
8     tmpHy = [Vector{Float64}(undef, n) for _ in 1:nthreads()]
9     @threads for i in 1:L
10         X = rand!(distx, tmpX[threadid()])
11         Y = rand!(disty, tmpY[threadid()])
12         pval_bm[i] = pvalue_brunner_munzel(X, Y, tmpHx[threadid()], tmpHy[threadid()])
13     end
14     ecdf(pval_bm)
15 end
16
17 function sim_welch();
18     distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10,
19     L = 10^6)
20     pval_w = Vector{Float64}(undef, L)
21     tmpX = [Vector{Float64}(undef, m) for _ in 1:nthreads()]
22     tmpY = [Vector{Float64}(undef, n) for _ in 1:nthreads()]
23     @threads for i in 1:L
24         X = rand!(distx, tmpX[threadid()])
25         Y = rand!(disty, tmpY[threadid()])
26         pval_w[i] = pvalue_welch(X, Y)
27     end
28     ecdf(pval_w)
29 end
30
31 function printcompact(io, xs...)
32     print(IOContext(io, :compact => true), xs...)
33 end
34
35 function distname(dist)
36     replace(sprint(printcompact, dist), r"\{\^\}\*\\"=>"")
37 end
38
39 function plot_ecdf(ecdf_pval, distx, disty, m, n, a;
40     testname = "", kwargs...)
41     plot(p → ecdf_pval(p), 0, 0.1; label="ecdf of P-values")
42     plot!([0, 0.1], [0, 0.1]; label="", ls=:dot, c=:black)
43     plot!(legend=:topleft)
44     plot!(xtick=0:0.01:0.1, ytick=0:0.01:1)
45     plot!(xguide="nominal significance level  $\alpha$ ",
46           yguide="probability of P-value <  $\alpha$ ")
47     s = (a < 0 ? "-" : "+") * string(round(a; digits=4))
48     title!("$(testname)X: $(distname(distx)), m=$m\n\
49           Y: $(distname(disty))$s, n=$n")
50     plot!(size=(400, 450))
51     plot!(; kwargs...)
52 end
53
54 function plot_pvals();
55     distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10,
56     L = 10^6, a = nothing, Δμ = nothing, kwargs...
57     @show (mean(distx), std(distx))
58     @show (mean(disty), std(disty))
59
60     if isnothing(a)
61         @show a = tieshift(distx, disty)
62         @show prob_x_le_y(distx, disty + a)
63     else
64         @show a
65         @show median(distx) - median(disty)
66     end
67     if isnothing(Δμ)
68         @show Δμ = mean(distx) - mean(disty)
69         @show mean(distx), mean(disty + Δμ)
70     else
71         @show Δμ
72         @show mean(distx), mean(disty + Δμ)
73     end
74
75     ecdf_bm = @time sim_brunner_mumzel();
76         distx = distx,
77         disty = disty + a,

```

```

78     m, n, L, kwargs...)
79     ecdf_w = @time sim_welch();
80     distx = distx,
81     disty = disty + Δμ,
82     m, n, L, kwargs...)
83     ymax = max(ecdf_bm(0.1), ecdf_w(0.1))
84     P1 = plot_ecdf(ecdf_bm, distx, disty, m, n, a;
85         testname="Brunner-Munzel test\n",
86         ylim=(-0.002, 1.02*ymax), kwargs...)
87     P2 = plot_ecdf(ecdf_w, distx, disty, m, n, Δμ;
88         testname="Welch t-test\n",
89         ylim=(-0.002, 1.02*ymax), kwargs...)
90     plot(P1, P2; size=(800, 450), topmargin=3.5Plots.mm)
91 end

```

Out[42]: `plot_pvals` (generic function with 1 method)

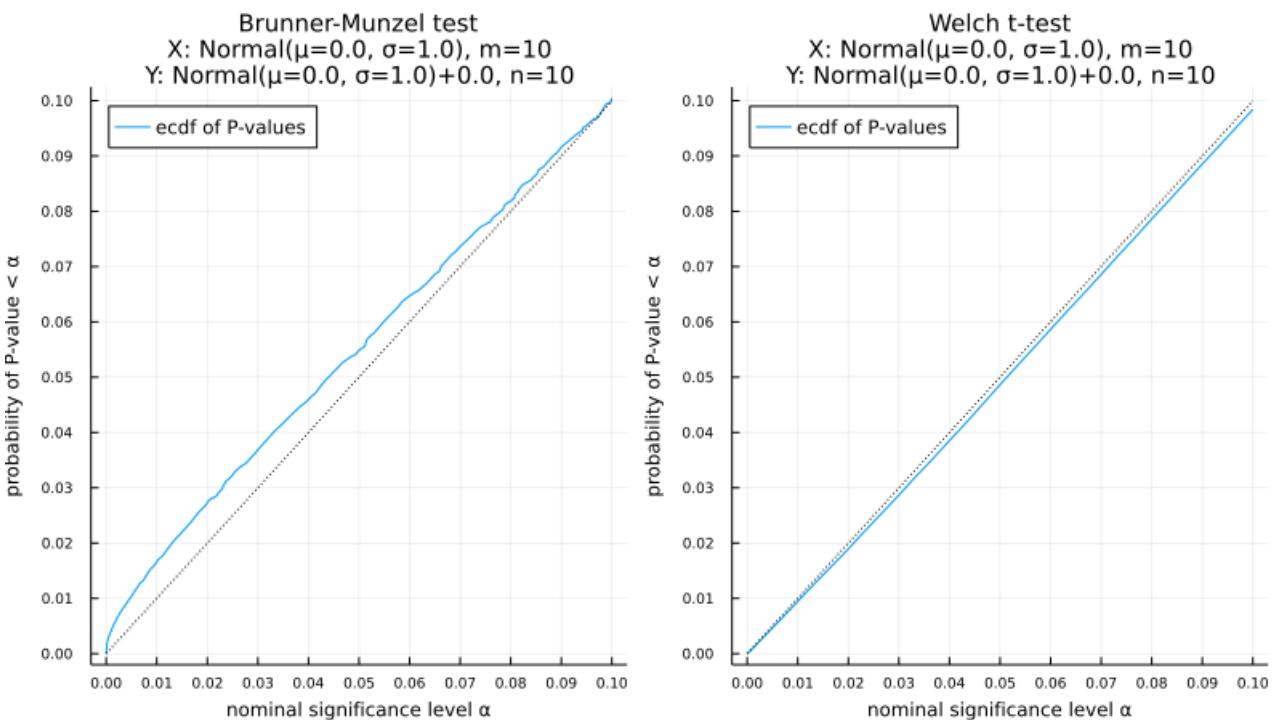
In [43]: 1 `plot_pvals(; distx = Normal(0, 1), disty = Normal(0, 1), m = 10, n = 10)`

```

(mean(distx), std(distx)) = (0.0, 1.0)
(mean(disty), std(disty)) = (0.0, 1.0)
a = tieshift(distx, disty) = 0.0
prob_x_le_y(distx, disty + a) = 0.5
Δμ = mean(distx) - mean(disty) = 0.0
(mean(distx), mean(disty + Δμ)) = (0.0, 0.0)
0.327782 seconds (55.66 k allocations: 25.861 MiB, 4.14% gc time, 16.23% compilation time)
0.253240 seconds (61.38 k allocations: 26.204 MiB, 22.86% compilation time)

```

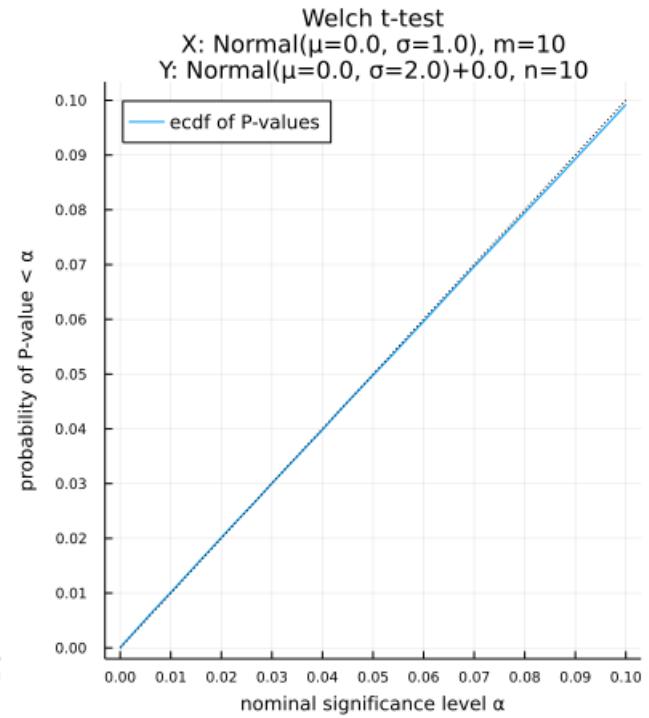
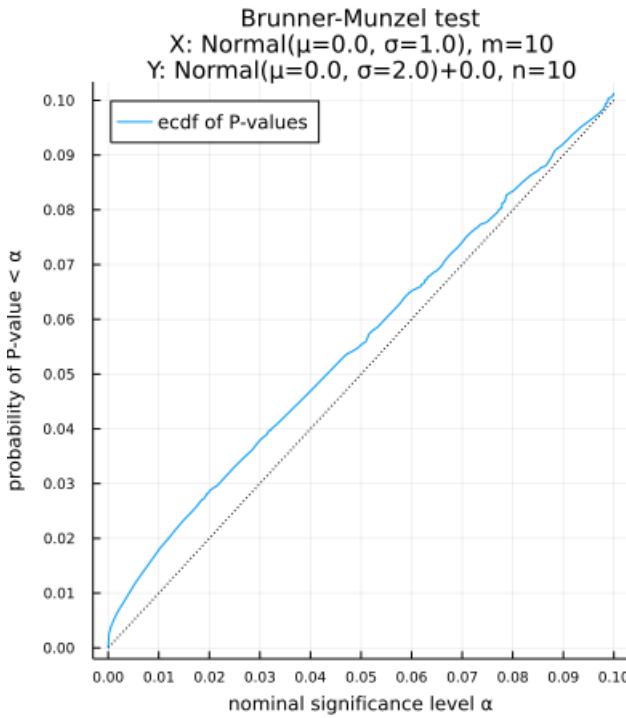
Out[43]:



```
In [44]: 1 plot_pvals(; distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10)
```

```
(mean(distx), std(distx)) = (0.0, 1.0)
(mean(disty), std(disty)) = (0.0, 2.0)
a = tieshift(distx, disty) = 7.685641860444171e-14
prob_x_le_y(distx, disty + a) = 0.5
Δμ = mean(distx) - mean(disty) = 0.0
(mean(distx), mean(disty + Δμ)) = (0.0, 0.0)
0.265909 seconds (233 allocations: 22.913 MiB)
0.188669 seconds (99 allocations: 22.899 MiB, 7.19% gc time)
```

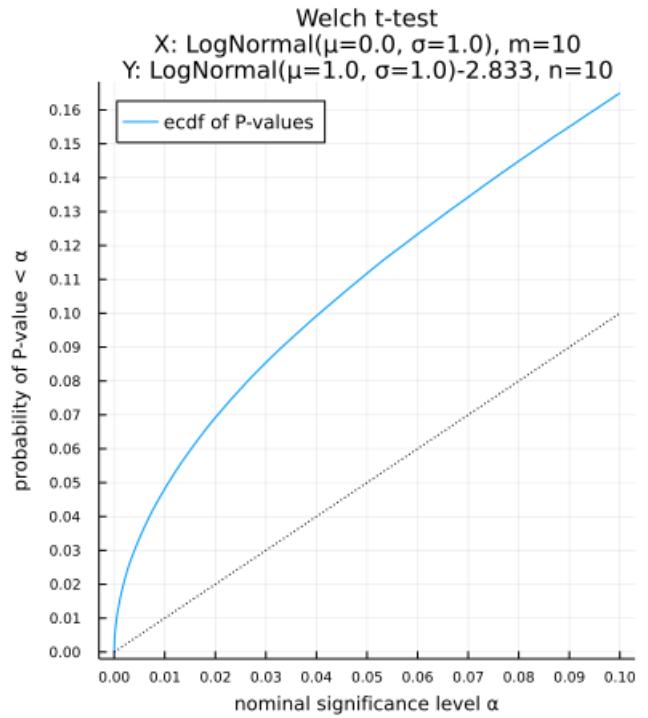
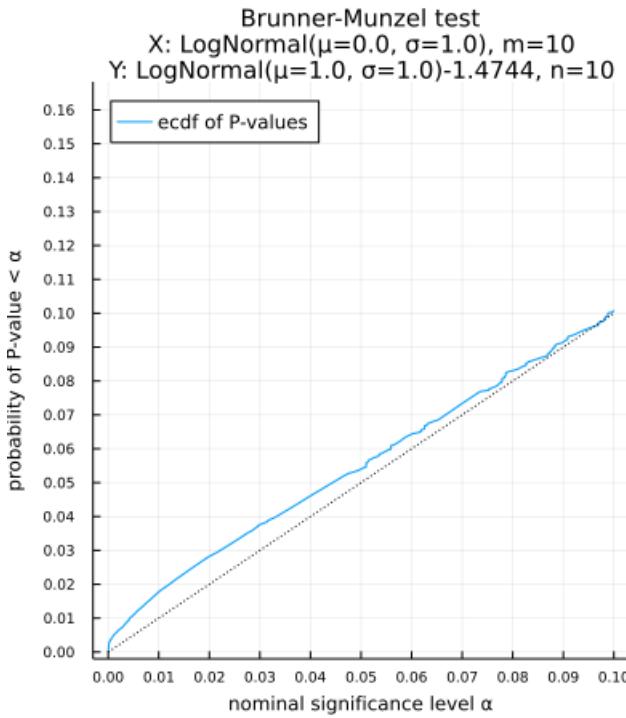
Out[44]:



```
In [45]: 1 plot_pvals(; distx = LogNormal(), disty = LogNormal(1), m = 10, n = 10)
```

```
(mean(distx), std(distx)) = (1.6487212707001282, 2.1611974158950877)
(mean(disty), std(disty)) = (4.4816890703380645, 5.874743663340262)
a = tieshift(distx, disty) = -1.4744426128871542
prob_x_le_y(distx, disty + a) = 0.5
Δμ = mean(distx) - mean(disty) = -2.8329677996379363
(mean(distx), mean(disty + Δμ)) = (1.6487212707001282, 1.6487212707001282)
0.336978 seconds (39.11 k allocations: 24.957 MiB, 5.04% gc time, 10.55% compilation time)
0.232974 seconds (22.04 k allocations: 24.034 MiB, 12.13% compilation time)
```

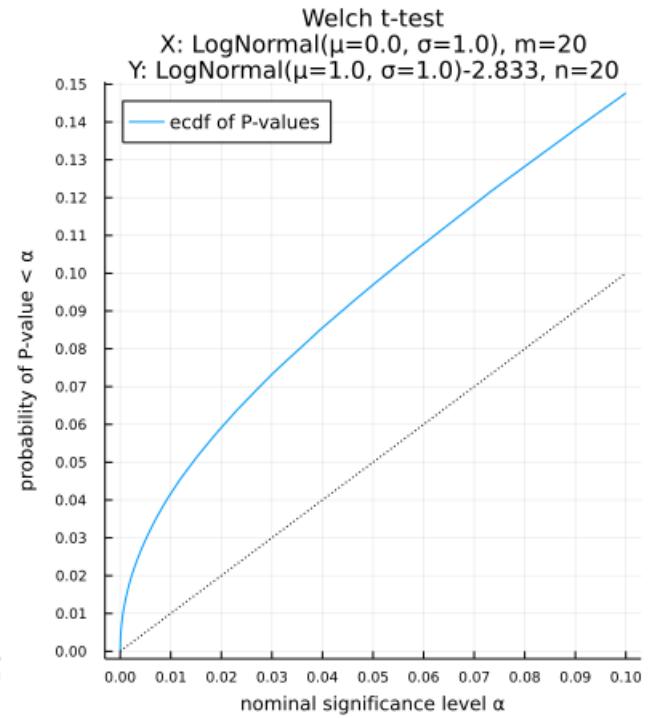
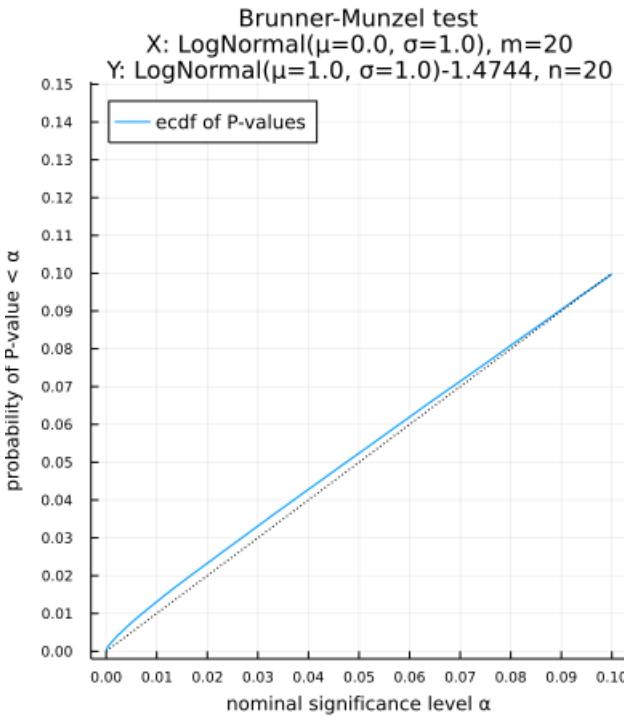
Out[45]:



```
In [46]: 1 plot_pvals(; distx = LogNormal(), disty = LogNormal(1), m = 20, n = 20)
```

```
(mean(distx), std(distx)) = (1.6487212707001282, 2.1611974158950877)
(mean(disty), std(disty)) = (4.4816890703380645, 5.874743663340262)
a = tieshift(distx, disty) = -1.4744426128871542
prob_x_le_y(distx, disty + a) = 0.5
Δμ = mean(distx) - mean(disty) = -2.8329677996379363
(mean(distx), mean(disty + Δμ)) = (1.6487212707001282, 1.6487212707001282)
0.463325 seconds (225 allocations: 22.916 MiB)
0.254335 seconds (208 allocations: 22.910 MiB, 3.64% gc time)
```

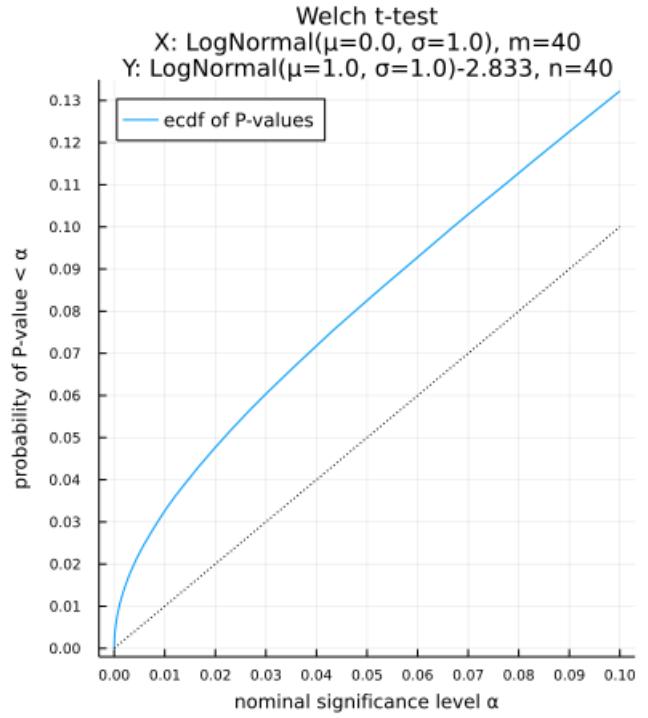
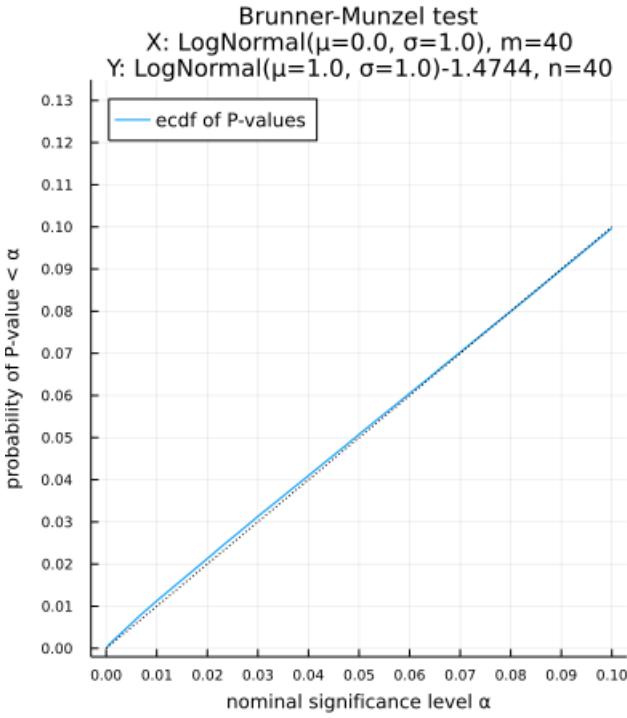
Out[46]:



```
In [47]: 1 plot_pvals(; distx = LogNormal(), disty = LogNormal(1), m = 40, n = 40)
```

```
(mean(distx), std(distx)) = (1.6487212707001282, 2.1611974158950877)
(mean(disty), std(disty)) = (4.4816890703380645, 5.874743663340262)
a = tieshift(distx, disty) = -1.4744426128871542
prob_x_le_y(distx, disty + a) = 0.5
Δμ = mean(distx) - mean(disty) = -2.8329677996379363
(mean(distx), mean(disty + Δμ)) = (1.6487212707001282, 1.6487212707001282)
0.821970 seconds (228 allocations: 22.925 MiB)
0.349232 seconds (206 allocations: 22.914 MiB)
```

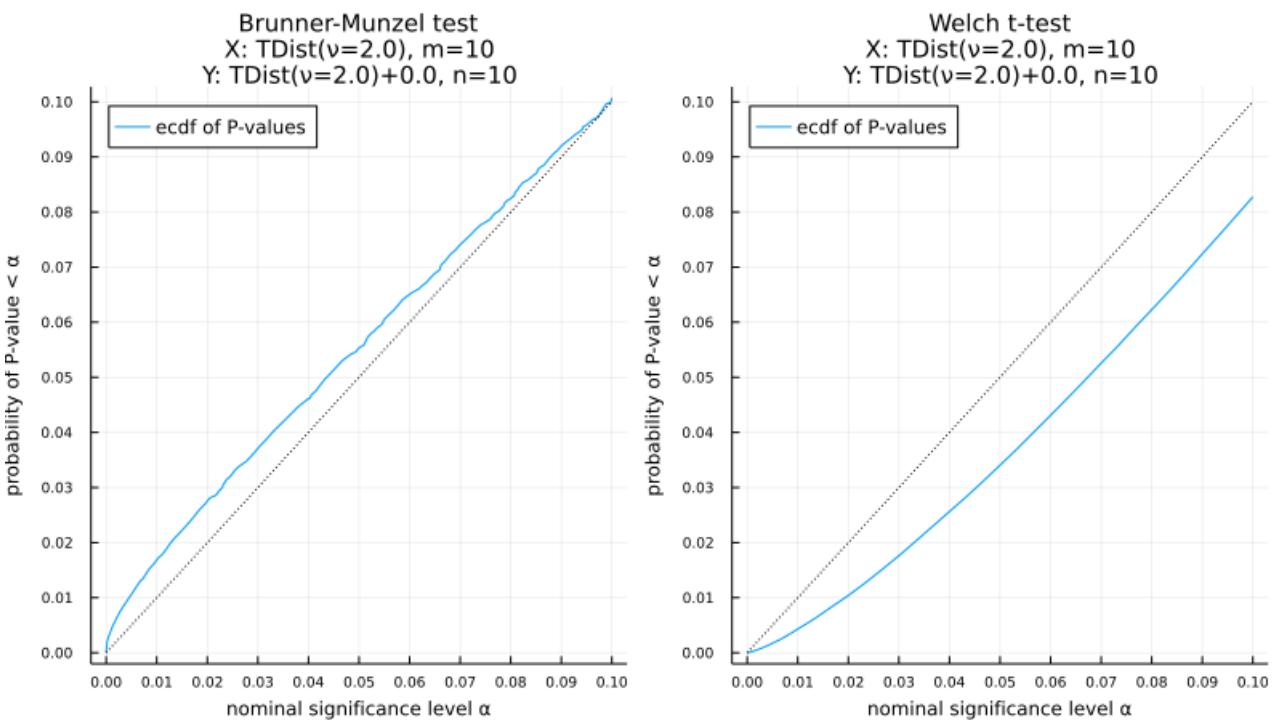
Out[47]:



```
In [48]: 1 plot_pvals(; distx = TDist(2), disty = TDist(2), m = 10, n = 10, Δμ = 0.0)
```

```
(mean(distx), std(distx)) = (0.0, Inf)
(mean(disty), std(disty)) = (0.0, Inf)
a = tieshift(distx, disty) = 0.0
prob_x_le_y(distx, disty + a) = 0.5000000000000001
Δμ = 0.0
(mean(distx), mean(disty + Δμ)) = (0.0, 0.0)
0.508473 seconds (426.25 k allocations: 45.673 MiB, 36.21% compilation time)
0.307722 seconds (22.04 k allocations: 24.034 MiB, 5.68% gc time, 10.40% compilation time)
```

Out[48]:

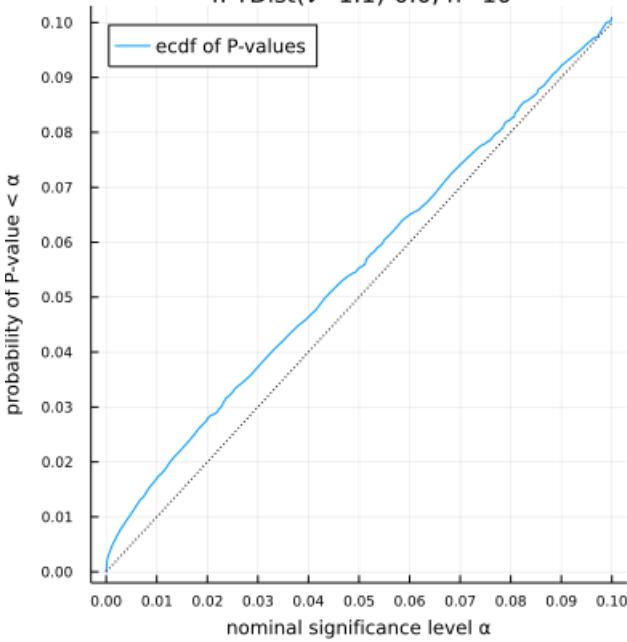


```
In [49]: 1 plot_pvals(; distx = TDist(2), disty = TDist(1.1), m = 10, n = 10, Δμ = 0.0)
```

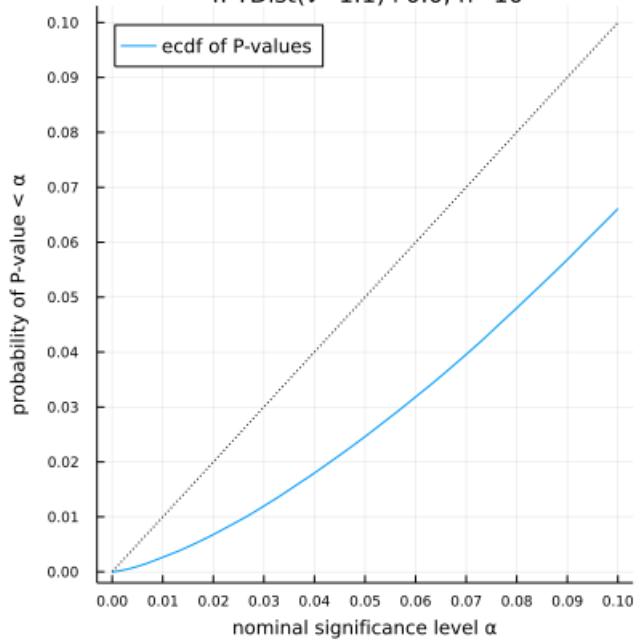
```
(mean(distx), std(distx)) = (0.0, Inf)
(mean(disty), std(disty)) = (0.0, Inf)
a = tieshift(distx, disty) = -3.4064499775914207e-9
prob_x_le_y(distx, disty + a) = 0.5
Δμ = 0.0
(mean(distx), mean(disty + Δμ)) = (0.0, 0.0)
0.406995 seconds (227 allocations: 22.913 MiB)
0.387248 seconds (207 allocations: 22.908 MiB)
```

Out[49]:

Brunner-Munzel test
X: TDist(v=2.0), m=10
Y: TDist(v=1.1)-0.0, n=10



Welch t-test
X: TDist(v=2.0), m=10
Y: TDist(v=1.1)+0.0, n=10

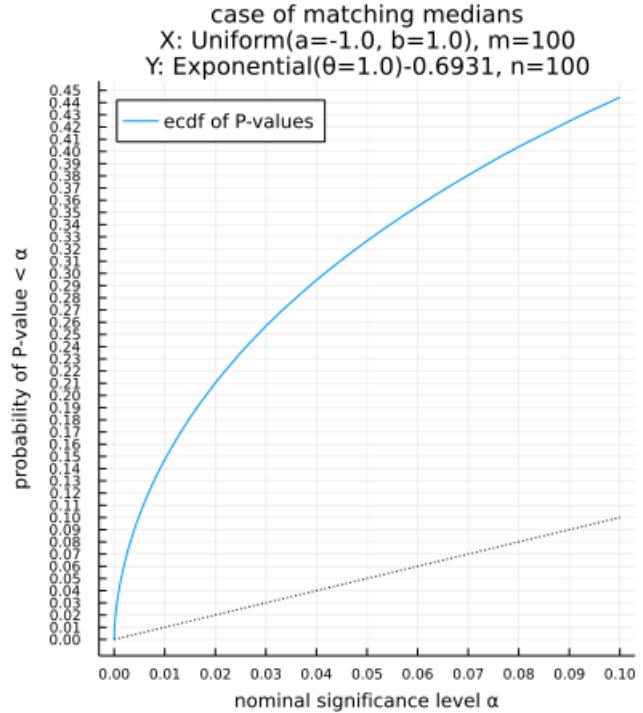
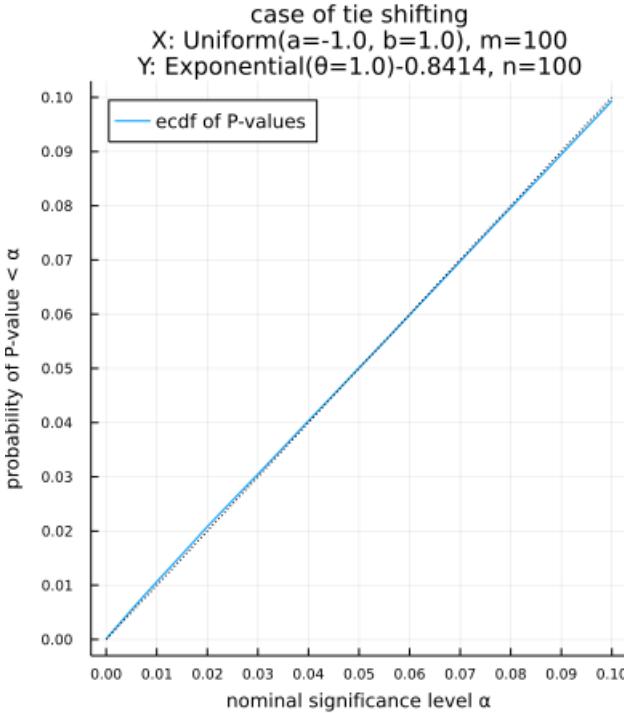


4.2 Brunner-Munzel検定は中央値に関する検定ではないことの証拠

```
In [50]: 1 distx, disty = Uniform(-1, 1), Exponential()
2 m, n, = 100, 100
3
4 @show distx, std(distx)
5 @show disty, std(disty)
6
7 @show a = tieshift(distx, disty)
8 ecdf_pval1 = @time sim_brunner_mumzel();
9     distx = distx, disty = disty + a, m, n)
10 P1 = plot_ecdf(ecdf_pval1, distx, disty, m, n, a;
11     testname="case of tie shifting\n")
12
13 @show a = median(distx) - median(disty)
14 ecdf_pval2 = @time sim_brunner_mumzel();
15     distx = distx, disty = disty + a, m, n)
16 P2 = plot_ecdf(ecdf_pval2, distx, disty, m, n, a;
17     testname="case of matching medians\n")
18
19 plot(P1, P2; size=(800, 450), topmargin=4Plots.mm)
```

```
(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
(disty, std(disty)) = (Exponential{Float64}(\theta=1.0), 1.0)
a = tieshift(distx, disty) = -0.8414056600399943
3.434486 seconds (472.42 k allocations: 48.296 MiB, 4.99% compilation time)
a = median(distx) - median(disty) = -0.6931471805599453
3.213846 seconds (239 allocations: 22.946 MiB, 0.54% gc time)
```

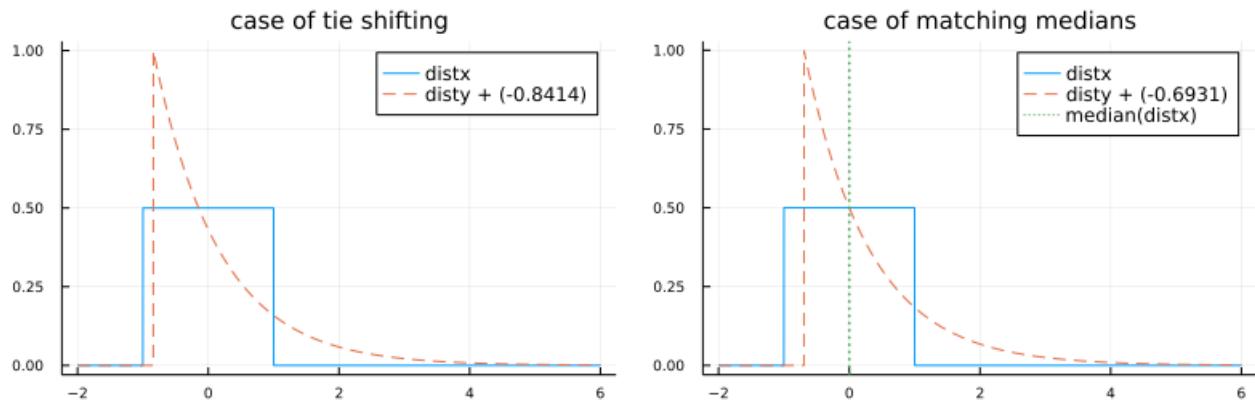
Out[50]:



```
In [51]: 1 distx, disty = Uniform(-1, 1), Exponential()
2 @show distx, std(distx)
3 @show disty, std(disty)
4
5 a = @show tieshift(distx, disty)
6 P1 = plot(distx, -2, 6; label="distx")
7 plot!(disty + a, -2, 6; label="disty + $(round(a; digits=4))", ls=:dash)
8 title!("case of tie shifting")
9
10 a = @show median(distx) - median(disty)
11 P2 = plot(distx, -2, 6; label="distx")
12 plot!(disty + a, -2, 6; label="disty + $(round(a; digits=4))", ls=:dash)
13 vline!([median(distx)]; label="median(distx)", ls=:dot, lw=1.5)
14 title!("case of matching medians")
15
16 plot(P1, P2; size=(800, 250))
```

```
(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
(disty, std(disty)) = (Exponential{Float64}(\theta=1.0), 1.0)
tieshift(distx, disty) = -0.8414056600399943
median(distx) - median(disty) = -0.6931471805599453
```

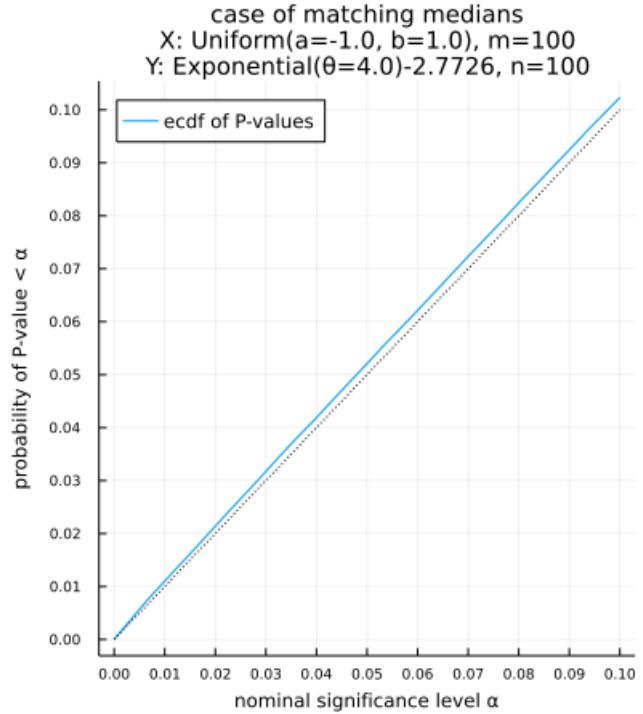
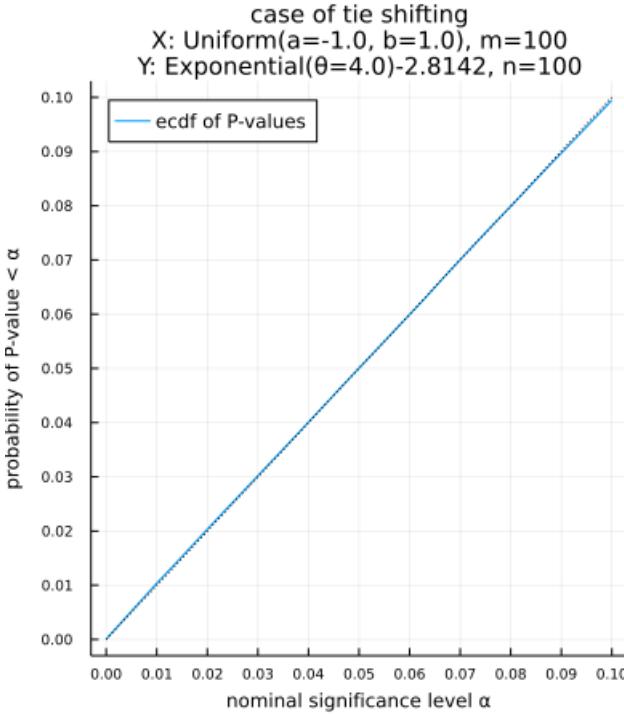
Out[51]:



```
In [52]: 1 distx, disty = Uniform(-1, 1), Exponential(4)
2 m, n, = 100, 100
3
4 @show distx, std(distx)
5 @show disty, std(disty)
6
7 @show a = tieshift(distx, disty)
8 ecdf_pval1 = @time sim_brunner_mumzel();
9     distx = distx, disty = disty + a, m, n)
10 P1 = plot_ecdf(ecdf_pval1, distx, disty, m, n, a;
11     testname="case of tie shifting\n")
12
13 @show a = median(distx) - median(disty)
14 ecdf_pval2 = @time sim_brunner_mumzel();
15     distx = distx, disty = disty + a, m, n)
16 P2 = plot_ecdf(ecdf_pval2, distx, disty, m, n, a;
17     testname="case of matching medians\n")
18
19 plot(P1, P2; size=(800, 450), topmargin=4Plots.mm)
```

```
(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
(disty, std(disty)) = (Exponential{Float64}(\theta=4.0), 4.0)
a = tieshift(distx, disty) = -2.814168911097315
3.207957 seconds (234 allocations: 22.947 MiB)
a = median(distx) - median(disty) = -2.772588722239781
3.218125 seconds (228 allocations: 22.946 MiB)
```

Out[52]:



In [53]:

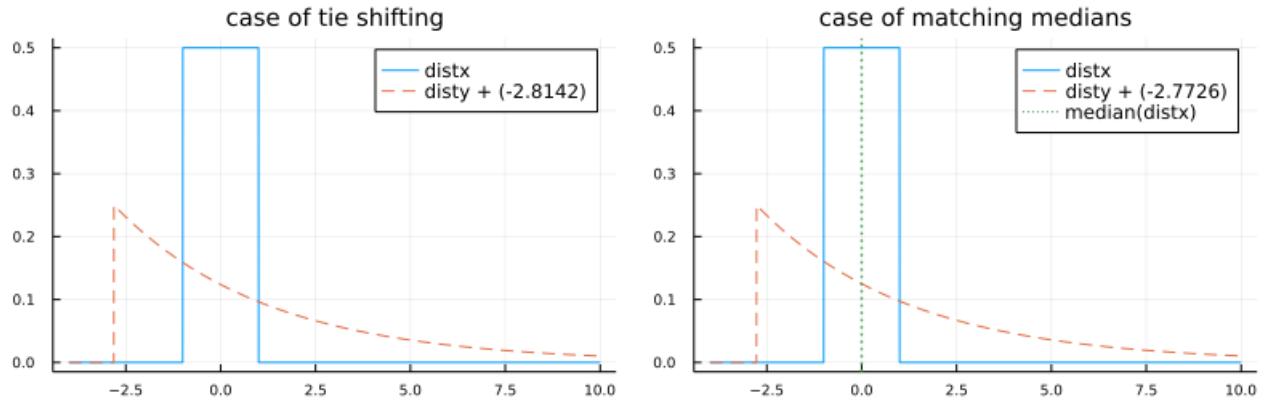
```

1 distx, disty = Uniform(-1, 1), Exponential(4)
2 @show distx, std(distx)
3 @show disty, std(disty)
4
5 a = @show tieshift(distx, disty)
6 P1 = plot(distx, -4, 10; label="distx")
7 plot!(disty + a, -4, 10; label="disty + $(round(a; digits=4))", ls=:dash)
8 title!("case of tie shifting")
9
10 a = @show median(distx) - median(disty)
11 P2 = plot(distx, -4, 10; label="distx")
12 plot!(disty + a, -4, 10; label="disty + $(round(a; digits=4))", ls=:dash)
13 vline!([median(distx)]; label="median(distx)", ls=:dot, lw=1.5)
14 title!("case of matching medians")
15
16 plot(P1, P2; size=(800, 250))

```

(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
 (disty, std(disty)) = (Exponential{Float64}($\theta=4.0$), 4.0)
 tieshift(distx, disty) = -2.814168911097315
 median(distx) - median(disty) = -2.772588722239781

Out[53]:

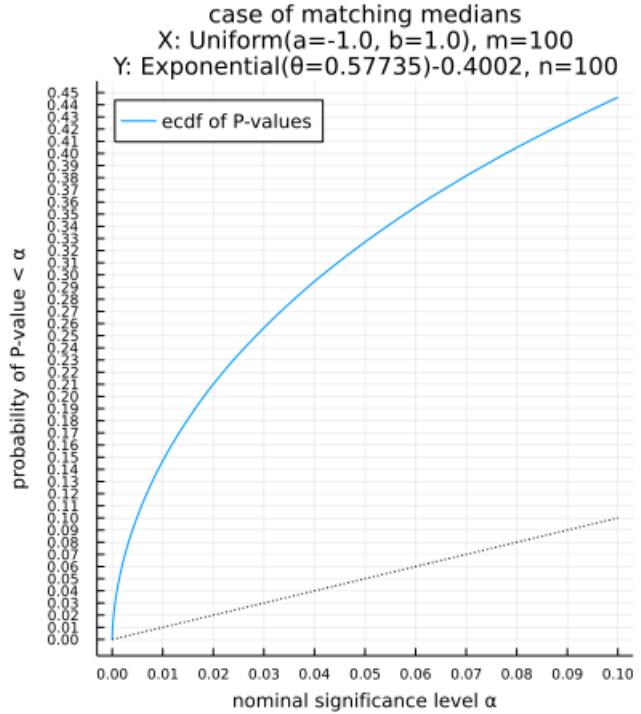
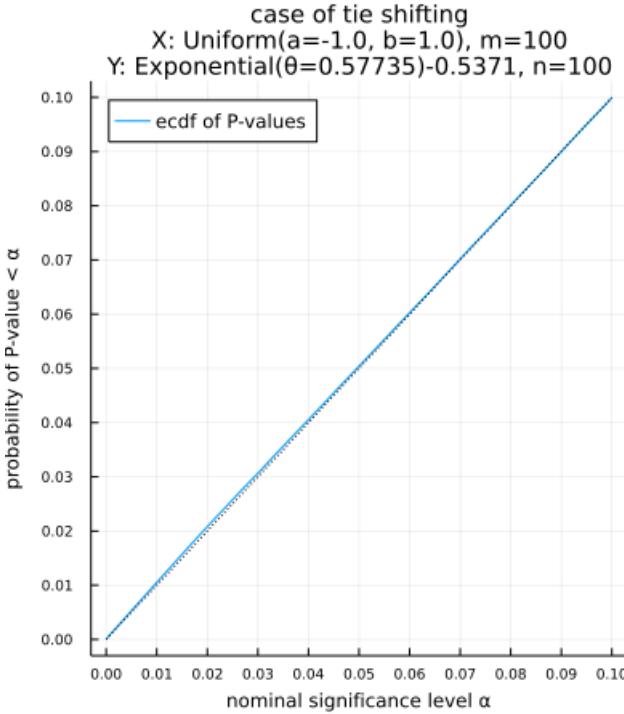


In [54]:

```
1 distx, disty = Uniform(-1, 1), Exponential(0.5773502691896257)
2 m, n, = 100, 100
3
4 @show distx, std(distx)
5 @show disty, std(disty)
6
7 @show a = tieshift(distx, disty)
8 ecdf_pval1 = @time sim_brunner_mumzel();
9     distx = distx, disty = disty + a, m, n)
10 P1 = plot_ecdf(ecdf_pval1, distx, disty, m, n, a;
11     testname="case of tie shifting\n")
12
13 @show a = median(distx) - median(disty)
14 ecdf_pval2 = @time sim_brunner_mumzel();
15     distx = distx, disty = disty + a, m, n)
16 P2 = plot_ecdf(ecdf_pval2, distx, disty, m, n, a;
17     testname="case of matching medians\n")
18
19 plot(P1, P2; size=(800, 450), topmargin=4Plots.mm)
```

```
(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
(disty, std(disty)) = (Exponential{Float64}(\theta=0.5773502691896257), 0.5773502691896257)
a = tieshift(distx, disty) = -0.5370568188698568
3.272786 seconds (231 allocations: 22.947 MiB)
a = median(distx) - median(disty) = -0.40018871128431455
3.275728 seconds (236 allocations: 22.946 MiB)
```

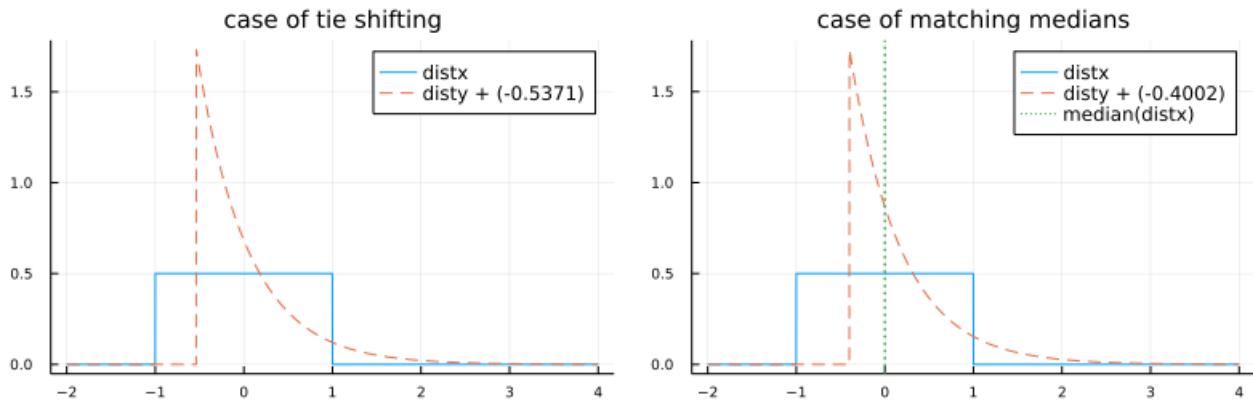
Out[54]:



```
In [55]: 1 distx, disty = Uniform(-1, 1), Exponential(0.5773502691896257)
2 @show distx, std(distx)
3 @show disty, std(disty)
4
5 a = @show tieshift(distx, disty)
6 P1 = plot(distx, -2, 4; label="distx")
7 plot!(disty + a, -2, 4; label="disty + $(round(a; digits=4))", ls=:dash)
8 title!("case of tie shifting")
9
10 a = @show median(distx) - median(disty)
11 P2 = plot(distx, -2, 4; label="distx")
12 plot!(disty + a, -2, 4; label="disty + $(round(a; digits=4))", ls=:dash)
13 vline!([median(distx)]; label="median(distx)", ls=:dot, lw=1.5)
14 title!("case of matching medians")
15
16 plot(P1, P2; size=(800, 250))
```

```
(distx, std(distx)) = (Uniform{Float64}(a=-1.0, b=1.0), 0.5773502691896257)
(disty, std(disty)) = (Exponential{Float64}(@=0.5773502691896257), 0.5773502691896257)
tieshift(distx, disty) = -0.5370568188698568
median(distx) - median(disty) = -0.40018871128431455
```

Out[55]:



4.3 BM検定による互角シフトの信頼区間とWelchのt検定による平均の差の信頼区間の比較

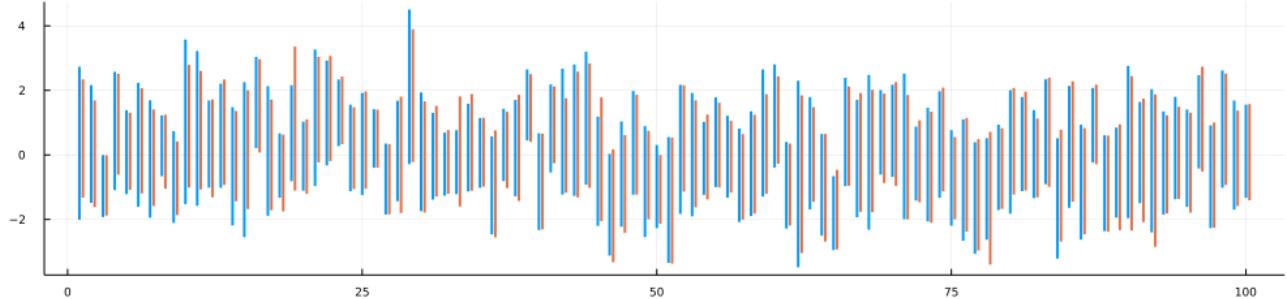
```
In [56]: 1 function plot_confints();
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10,
3     L = 100, kwargs...)
4     a = tieshift(distx, disty)
5     Δμ = mean(distx) - mean(disty)
6     BM = fill(zeros(2), 0)
7     W = fill(zeros(2), 0)
8     for _ in 1:L
9         X = rand(distx, m)
10        Y = rand(disty, n)
11        push!(BM, brunner_munzel(X, Y .+ a).confint_shift)
12        push!(W, confint_welch(X, Y .+ Δμ))
13    end
14    P = plot()
15    for i in 1:L
16        plot!(fill(i, 2), [first(BM[i]), last(BM[i])]; label="", c=1, lw=2)
17        plot!(fill(i+0.3, 2), [first(W[i]), last(W[i])]; label="", c=2, lw=2)
18    end
19    title!("X: $($distname(distx)), m=$m, Y: $($distname(disty)), n=$n")
20    plot!(size=(1000, 250))
21 end
```

Out[56]: `plot_confints` (generic function with 1 method)

```
In [57]: 1 plot_confints(distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10)
```

Out[57]:

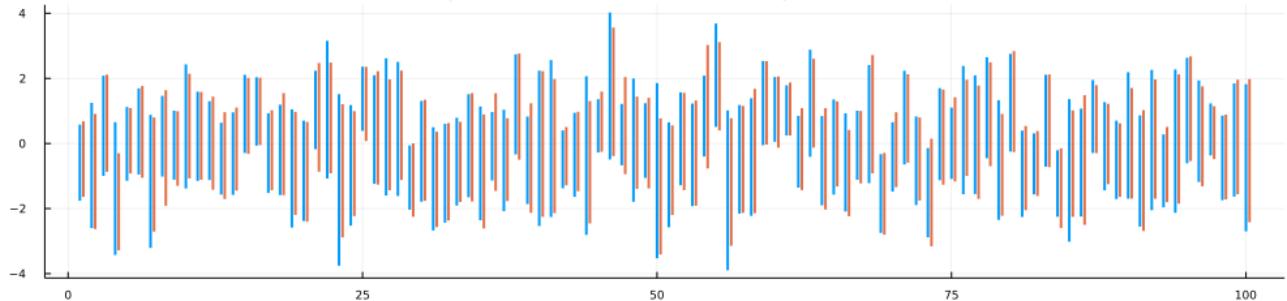
X: Normal($\mu=0.0, \sigma=1.0$), m=10, Y: Normal($\mu=0.0, \sigma=2.0$), n=10



```
In [58]: 1 plot_confints(distx = Normal(2, 1), disty = Normal(0, 2), m = 10, n = 10)
```

Out[58]:

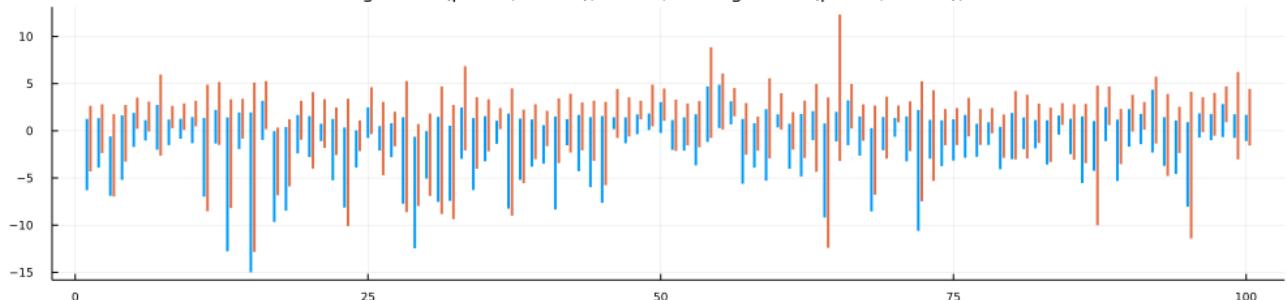
X: Normal($\mu=2.0, \sigma=1.0$), m=10, Y: Normal($\mu=0.0, \sigma=2.0$), n=10



```
In [59]: 1 plot_confints(distx = LogNormal(0), disty = LogNormal(1), m = 10, n = 10)
```

Out[59]:

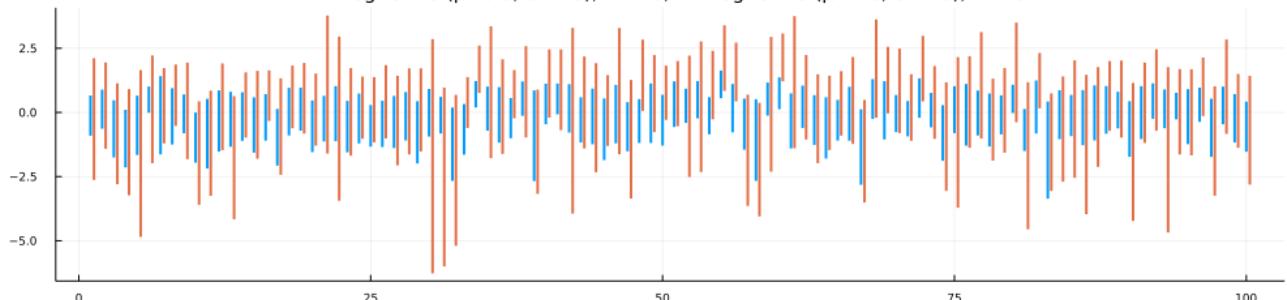
X: LogNormal($\mu=0.0, \sigma=1.0$), m=10, Y: LogNormal($\mu=1.0, \sigma=1.0$), n=10



```
In [60]: 1 plot_confints(distx = LogNormal(0), disty = LogNormal(1), m = 40, n = 40)
```

Out[60]:

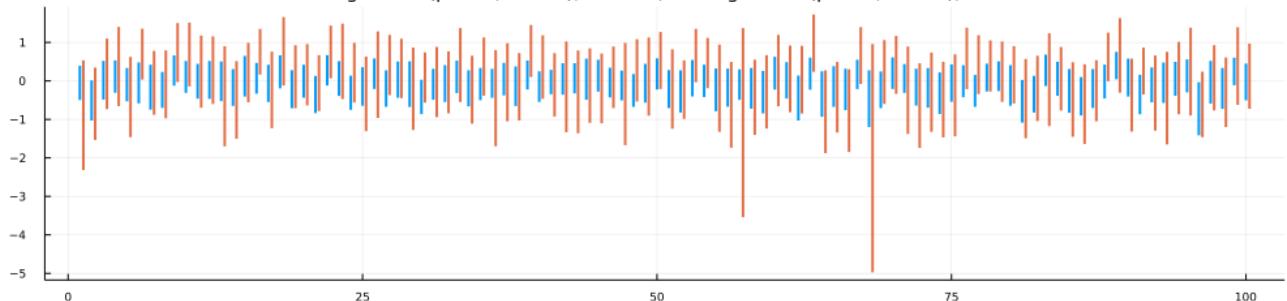
X: LogNormal($\mu=0.0, \sigma=1.0$), m=40, Y: LogNormal($\mu=1.0, \sigma=1.0$), n=40



```
In [61]: 1 plot_confints(distx = LogNormal(0), disty = LogNormal(1), m = 160, n = 160)
```

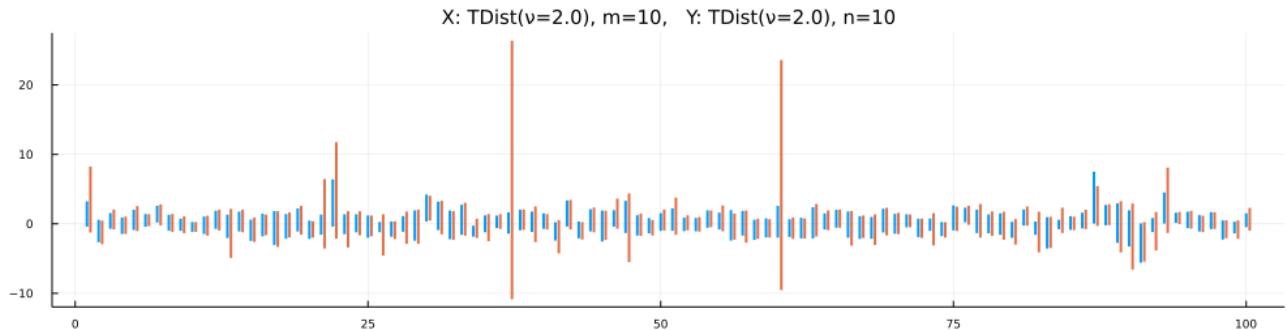
Out[61]:

X: LogNormal($\mu=0.0, \sigma=1.0$), m=160, Y: LogNormal($\mu=1.0, \sigma=1.0$), n=160



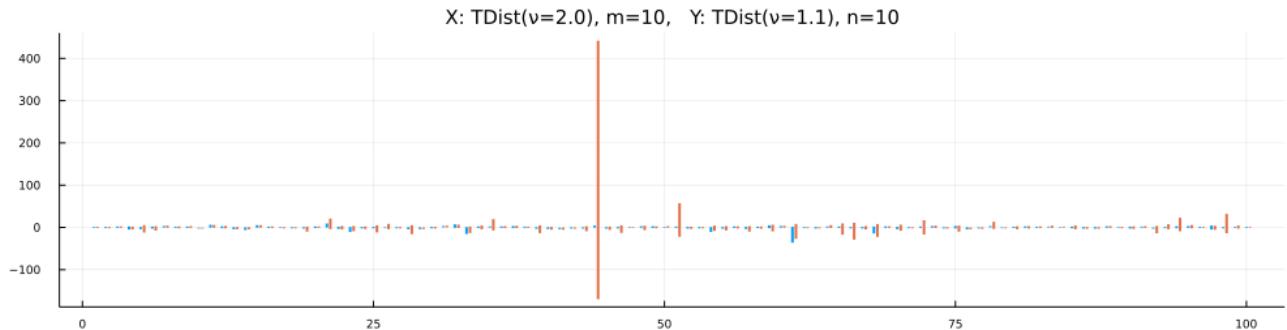
```
In [62]: 1 plot_confints(distx = TDist(2), disty = TDist(2), m = 10, n = 10)
```

Out[62]:



```
In [63]: 1 plot_confints(distx = TDist(2), disty = TDist(1.1), m = 10, n = 10)
```

Out[63]:



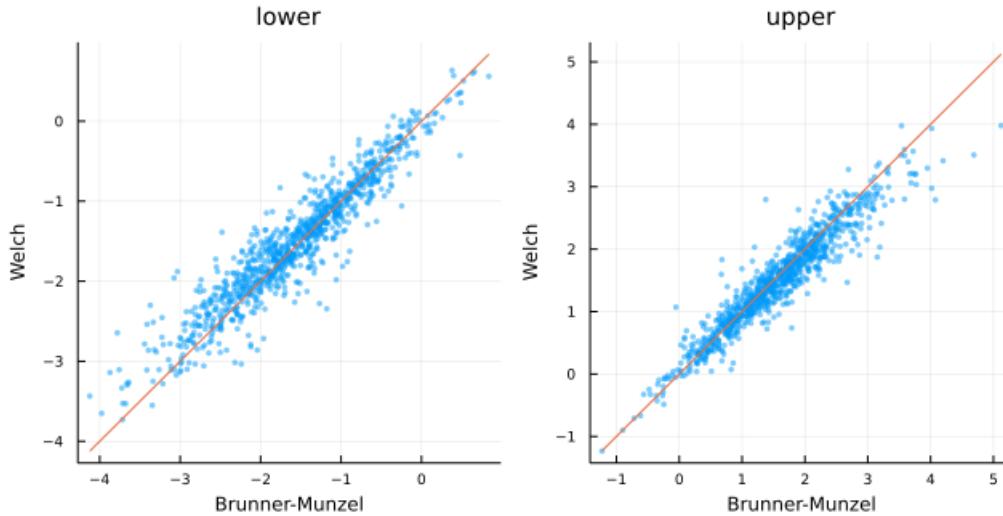
```
1 function plot_limits();
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10,
3     L = 1000, kwargs...
4
5     @show distx, m
6     @show disty, n
7
8     a = tieshift(distx, disty)
9     Δμ = mean(distx) - mean(disty)
10
11    BM = fill(zeros(2), 0)
12    W = fill(zeros(2), 0)
13    for _ in 1:L
14        X = rand(distx, m)
15        Y = rand(disty, n)
16        push!(BM, brunner_munzel(X, Y .+ a).confint_shift)
17        push!(W, confint_welch(X, Y .+ Δμ))
18    end
19
20    lower = [(first(BM[i]), first(W[i])) for i in 1:L]
21    upper = [(last(BM[i]), last(W[i])) for i in 1:L]
22
23    P1 = scatter(lower; label="", msc=:auto, ms=2, ma=0.5)
24    plot!(identity; label="")
25    plot!(xguide="Brunner-Munzel", yguide="Welch")
26    title!("lower")
27
28    P2 = scatter(upper; label="", msc=:auto, ms=2, ma=0.5)
29    plot!(identity; label="")
30    plot!(xguide="Brunner-Munzel", yguide="Welch")
31    title!("upper")
32
33    plot(P1, P2; size=(640, 320))
34 end
```

Out[64]: plot_limits (generic function with 1 method)

```
In [65]: 1 plot_limits(distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10)
```

```
(distx, m) = (Normal{Float64}(\mu=0.0, σ=1.0), 10)
(disty, n) = (Normal{Float64}(\mu=0.0, σ=2.0), 10)
```

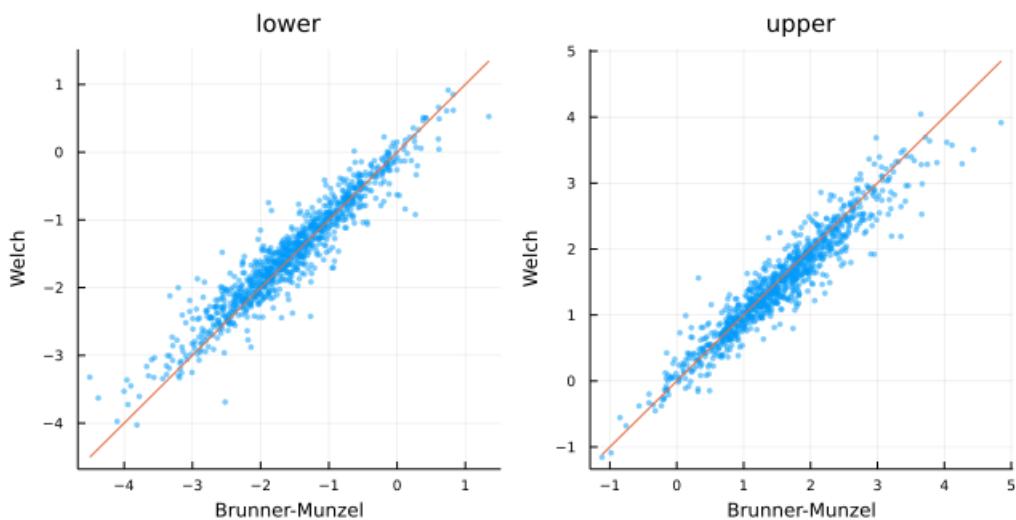
Out[65]:



```
In [66]: 1 plot_limits(distx = Normal(2, 1), disty = Normal(0, 2), m = 10, n = 10)
```

```
(distx, m) = (Normal{Float64}(\mu=2.0, σ=1.0), 10)
(disty, n) = (Normal{Float64}(\mu=0.0, σ=2.0), 10)
```

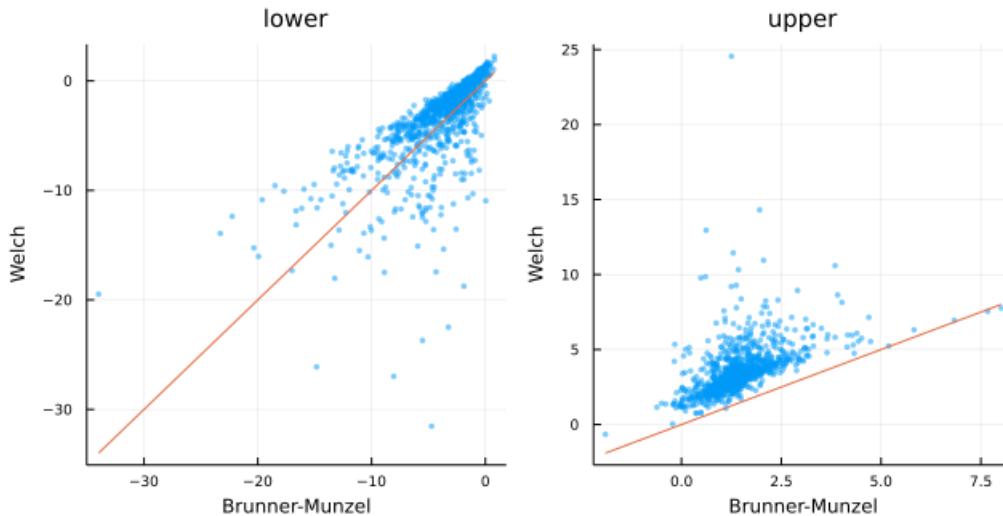
Out[66]:



```
In [67]: 1 plot_limits(distx = LogNormal(), disty = LogNormal(1), m = 10, n = 10)
```

```
(distx, m) = (LogNormal{Float64}( $\mu=0.0$ ,  $\sigma=1.0$ ), 10)
(disty, n) = (LogNormal{Float64}( $\mu=1.0$ ,  $\sigma=1.0$ ), 10)
```

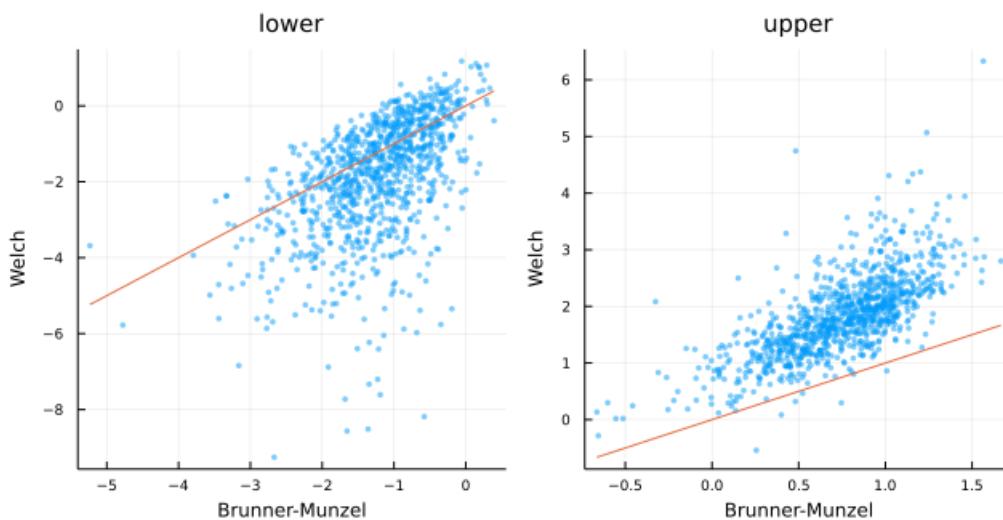
Out[67]:



```
In [68]: 1 plot_limits(distx = LogNormal(), disty = LogNormal(1), m = 40, n = 40)
```

```
(distx, m) = (LogNormal{Float64}( $\mu=0.0$ ,  $\sigma=1.0$ ), 40)
(disty, n) = (LogNormal{Float64}( $\mu=1.0$ ,  $\sigma=1.0$ ), 40)
```

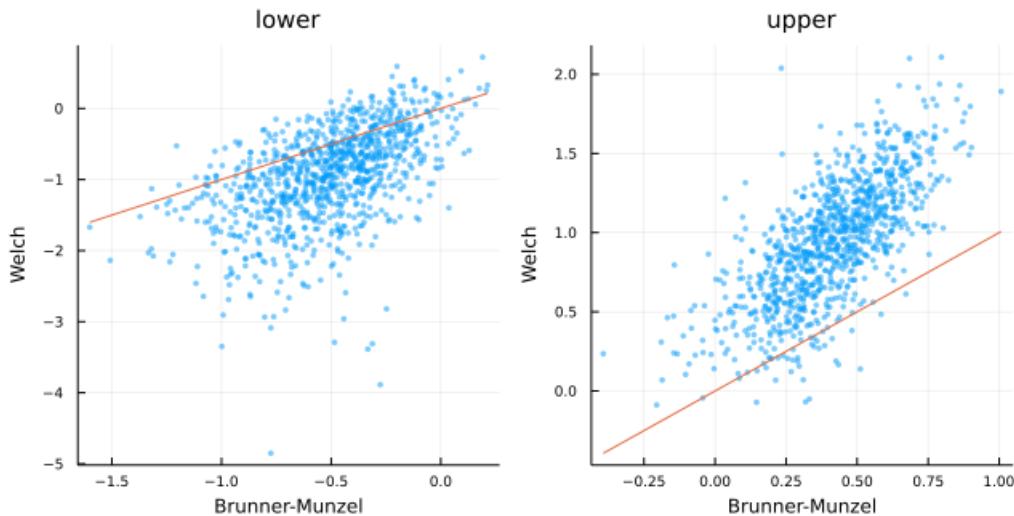
Out[68]:



```
In [69]: 1 @time plot_limits(distx = LogNormal(), disty = LogNormal(1), m = 160, n = 160)
```

```
(distx, m) = (LogNormal{Float64}( $\mu=0.0$ ,  $\sigma=1.0$ ), 160)
(disty, n) = (LogNormal{Float64}( $\mu=1.0$ ,  $\sigma=1.0$ ), 160)
6.793918 seconds (61.67 k allocations: 38.622 MiB)
```

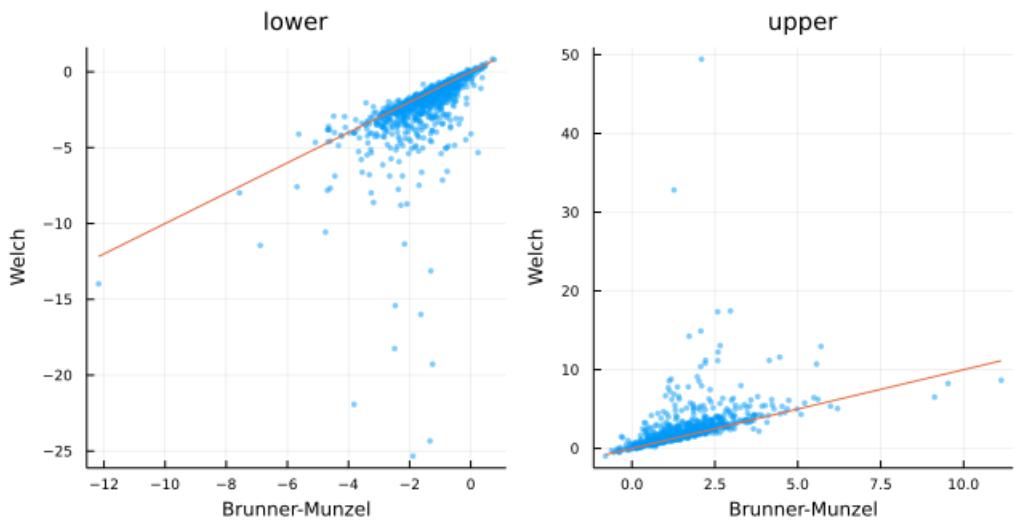
Out[69]:



```
In [70]: 1 plot_limits(distx = TDist(2), disty = TDist(2), m = 10, n = 10)
```

```
(distx, m) = (TDist{Float64}(v=2.0), 10)
(disty, n) = (TDist{Float64}(v=2.0), 10)
```

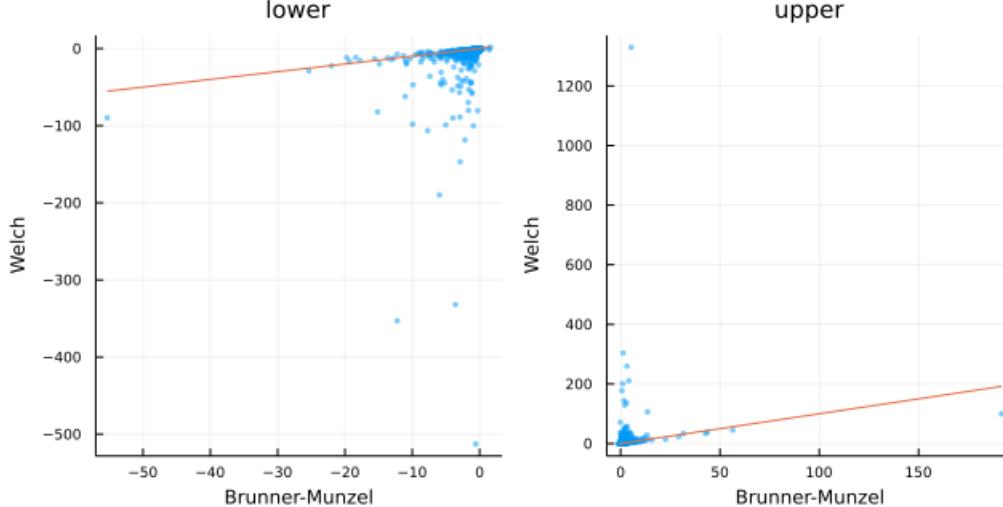
Out[70]:



```
In [71]: 1 plot_limits(distx = TDist(2), disty = TDist(1.1), m = 10, n = 10)

(distx, m) = (TDist{Float64}(ν=2.0), 10)
(disty, n) = (TDist{Float64}(ν=1.1), 10)
```

Out[71]:



5 小サンプルでのpermutation版の検定とBM検定とWelchのt検定の比較

```
In [72]: 1 function sim_brunner_mumzel_perm();
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 5, n = 5,
3     L = 10^2)
4     pval_bm_perm = Vector{Float64}(undef, L)
5     tmpX = [Vector{Float64}(undef, m) for _ in 1:nthreads()]
6     tmpY = [Vector{Float64}(undef, n) for _ in 1:nthreads()]
7     tmpXandY = [Vector{Float64}(undef, m+n) for _ in 1:nthreads()]
8     tmpTval = [Vector{Float64}(undef, binomial(m+n, m)) for _ in 1:nthreads()]
9     tmpHx = [Vector{Float64}(undef, m) for _ in 1:nthreads()]
10    tmpHy = [Vector{Float64}(undef, n) for _ in 1:nthreads()]
11    tmpccomb = [Vector{Int}(undef, n) for _ in 1:nthreads()]
12    @threads for i in 1:L
13        tid = threadid()
14        X = rand!(distx, tmpX[tid])
15        Y = rand!(disty, tmpY[tid])
16        Tval = permutation_tvalues_brunner_munzel(X, Y,
17            tmpXandY[tid], tmpTval[tid], tmpHx[tid], tmpHy[tid], tmpccomb[tid])
18        tval = statistics_brunner_munzel(X, Y, tmpHx[tid], tmpHy[tid]).tvalue
19        pval_bm_perm[i] = pvalue_brunner_munzel_perm(X, Y, Tval, tval)
20    end
21    ecdf(pval_bm_perm)
22 end
```

Out[72]: sim_brunner_mumzel_perm (generic function with 1 method)

```
In [73]: 1 @time ecdf_bm_perm = sim_brunner_mumzel_perm(  
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 7, n = 7, L = 10^4)
```

3.799263 seconds (35.80 k allocations: 2.941 Mib, 1.10% compilation time)

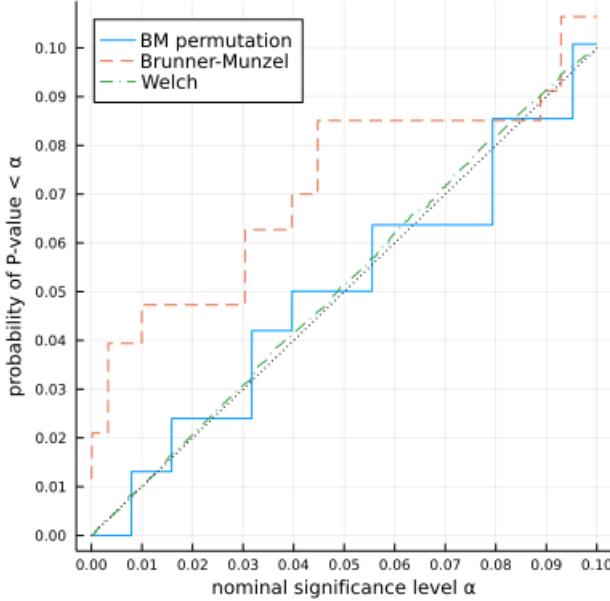
```
In [74]: 1 function plot_pvals_with_perm();
2     distx = Normal(0, 1),
3     disty = Normal(0, 2),
4     m = 7,
5     n = 7,
6     L = 10^4,
7     kwargs...
8 )
9 a = tieshift(distx, disty)
10 @time ecdf_bm_perm = sim_brunner_mumzel_perm(; distx, disty = disty + a, m, n, L)
11 @time ecdf_bm = sim_brunner_mumzel(; distx, disty = disty + a, m, n, L)
12 Δμ = mean(distx) - mean(disty)
13 @time ecdf_w = sim_welch(; distx, disty = disty + Δμ, m, n, L)
14 @show a Δμ
15
16 plot(legend=:topleft)
17 plot!(α → ecdf_bm_perm(α), 0, 0.1; label="BM permutation")
18 plot!(α → ecdf_bm(α), 0, 0.1; label="Brunner-Munzel", ls=:dash)
19 plot!(α → ecdf_w(α), 0, 0.1; label="Welch", ls=:dashdot)
20 plot!(identity; label="", c=:black, ls=:dot)
21 plot!(xtick=0:0.01:0.1, ytick=0:0.01:1)
22 plot!(xguide="nominal significance level α",
23       yguide="probability of P-value < α")
24 a_ = string(round(a; digits=4))
25 Δμ_ = string(round(Δμ; digits=4))
26 title!("X: $(distname(distx)), m=$m\n\
27         Y: $(distname(disty))+(a, Δμ), n=$n\n\
28         a=$a_, Δμ=$Δμ_")
29 plot!(size=(400, 450), titlefontsize=9)
30 plot!(; kwargs...)
31 end
```

Out[74]: plot_pvals_with_perm (generic function with 1 method)

```
In [75]: 1 plot_pvals_with_perm()
2         distx = Normal(0, 1), disty = Normal(0, 2), m = 5, n = 5, L = 10^4)
```

0.229530 seconds (10.17 k allocations: 1.185 MiB)
 0.017653 seconds (131 allocations: 247.422 KiB)
 0.004728 seconds (106 allocations: 244.734 KiB)
 $a = 7.685641860444171e-14$
 $\Delta\mu = 0.0$

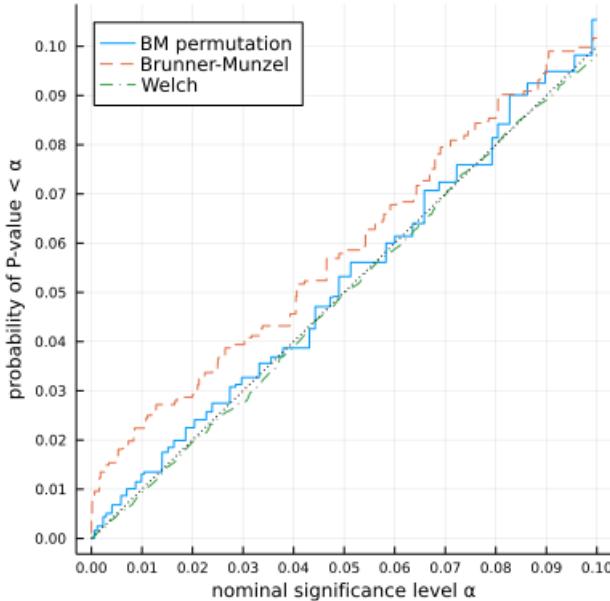
Out[75]: X: Normal($\mu=0.0, \sigma=1.0$), m=5
 Y: Normal($\mu=0.0, \sigma=2.0+(a, \Delta\mu)$, n=5
 $a=0.0, \Delta\mu=0.0$



```
In [76]: 1 plot_pvals_with_perm()
2         distx = Normal(0, 1), disty = Normal(0, 2), m = 7, n = 7, L = 10^4)
```

3.807640 seconds (10.19 k allocations: 1.629 MiB)
 0.005800 seconds (125 allocations: 247.859 KiB)
 0.027359 seconds (104 allocations: 245.047 KiB)
 $a = 7.685641860444171e-14$
 $\Delta\mu = 0.0$

Out[76]: X: Normal($\mu=0.0, \sigma=1.0$), m=7
 Y: Normal($\mu=0.0, \sigma=2.0+(a, \Delta\mu)$, n=7
 $a=0.0, \Delta\mu=0.0$



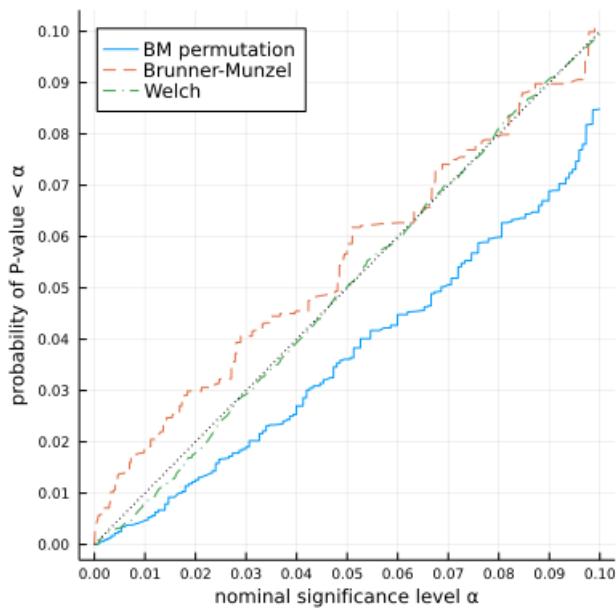
In [77]:

```
1 plot_pvals_with_perm()
2     distx = Normal(0, 1), disty = Normal(0, 2), m = 5, n = 10, L = 10^4)
```

```
3.348652 seconds (10.18 k allocations: 1.438 MiB)
0.001652 seconds (133 allocations: 248.609 KiB)
0.004396 seconds (104 allocations: 245.234 KiB)
a = 7.685641860444171e-14
Δμ = 0.0
```

Out[77]:

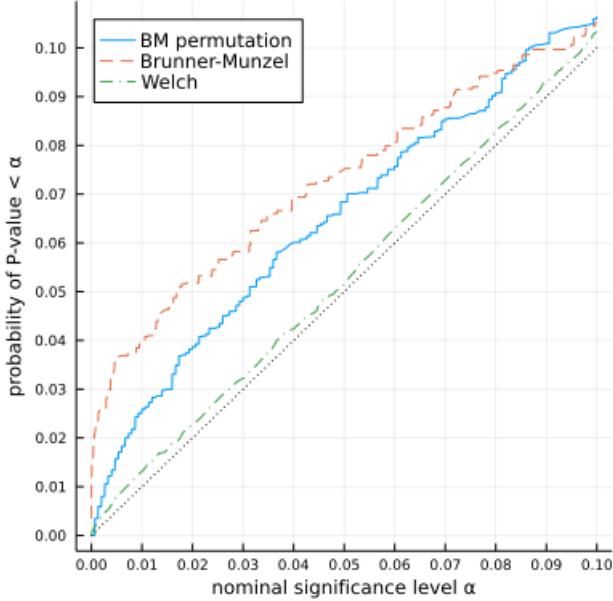
X: $\text{Normal}(\mu=0.0, \sigma=1.0)$, m=5
Y: $\text{Normal}(\mu=0.0, \sigma=2.0) + (a, \Delta\mu)$, n=10
 $a=0.0, \Delta\mu=0.0$



```
In [78]: 1 plot_pvals_with_perm()
2         distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 5, L = 10^4)
```

3.429233 seconds (10.18 k allocations: 1.895 MiB)
0.018557 seconds (125 allocations: 248.438 KiB)
0.004641 seconds (99 allocations: 245.125 KiB)
a = 7.685641860444171e-14
 $\Delta\mu = 0.0$

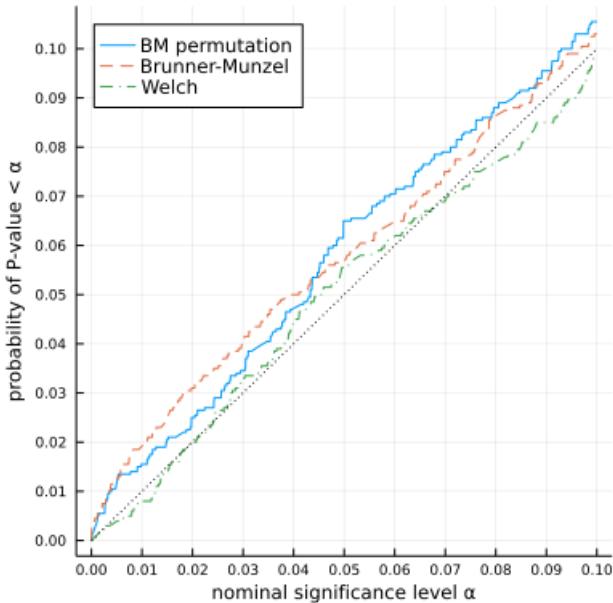
Out[78]: X: Normal($\mu=0.0, \sigma=1.0$), m=10
Y: Normal($\mu=0.0, \sigma=2.0+(a, \Delta\mu)$, n=5
 $a=0.0, \Delta\mu=0.0$



```
In [79]: 1 plot_pvals_with_perm()
2         distx = Normal(0, 1), disty = Normal(0, 2), m = 10, n = 10, L = 2000)
```

53.438533 seconds (2.23 k allocations: 17.257 MiB, 0.03% gc time)
0.000561 seconds (130 allocations: 62.625 KiB)
0.025479 seconds (101 allocations: 58.531 KiB)
a = 7.685641860444171e-14
 $\Delta\mu = 0.0$

Out[79]: X: Normal($\mu=0.0, \sigma=1.0$), m=10
Y: Normal($\mu=0.0, \sigma=2.0+(a, \Delta\mu)$, n=10
 $a=0.0, \Delta\mu=0.0$



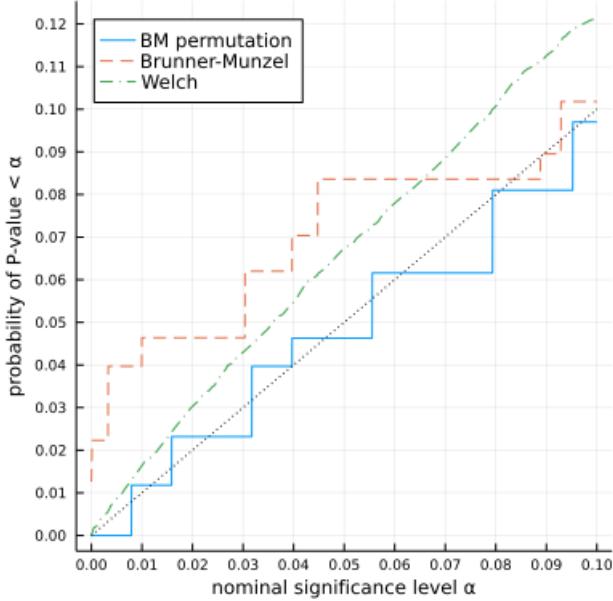
In [80]:

```
1 plot_pvals_with_perm()
2     distx = Exponential(1), disty = Exponential(2),
3     m = 5, n = 5, L = 10^4)
```

```
0.293526 seconds (41.28 k allocations: 2.806 MiB, 15.89% compilation time)
0.029709 seconds (20.94 k allocations: 1.318 MiB, 94.94% compilation time)
0.041234 seconds (18.79 k allocations: 1.204 MiB, 57.05% compilation time)
a = -0.5753641445892759
Δμ = -1.0
```

Out[80]:

X: Exponential($\theta=1.0$), m=5
Y: Exponential($\theta=2.0$) $+(a, \Delta\mu)$, n=5
 $a=-0.5754, \Delta\mu=-1.0$



In [81]:

```

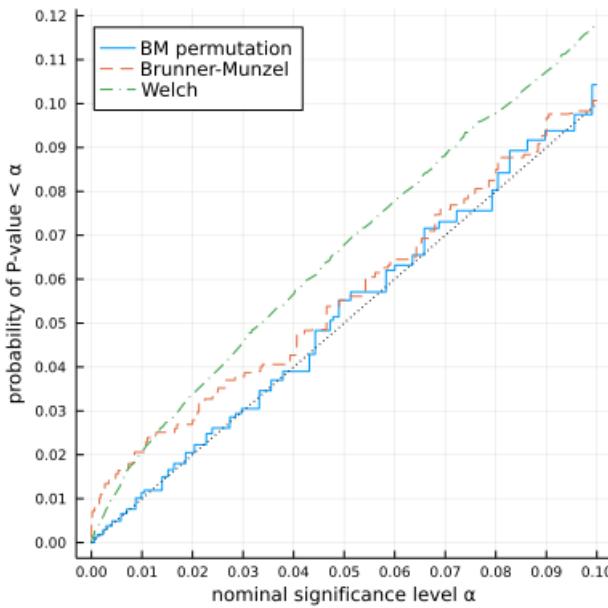
1 plot_pvals_with_perm(
2     distx = Exponential(1), disty = Exponential(2),
3     m = 7, n = 7, L = 10^4)

```

3.948691 seconds (10.18 k allocations: 1.629 MiB)
0.004954 seconds (125 allocations: 247.859 KiB)
0.017548 seconds (104 allocations: 245.047 KiB)
 $a = -0.5753641445892759$
 $\Delta\mu = -1.0$

Out[81]:

X: Exponential($\theta=1.0$), m=7
Y: Exponential($\theta=2.0$)+(a, $\Delta\mu$), n=7
 $a=-0.5754$, $\Delta\mu=-1.0$



In [82]:

```

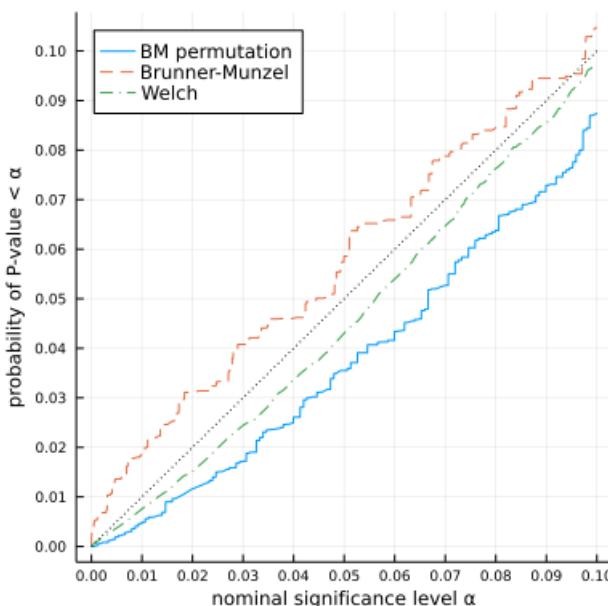
1 plot_pvals_with_perm(
2     distx = Exponential(1), disty = Exponential(2),
3     m = 5, n = 10, L = 10^4)

```

3.462247 seconds (10.18 k allocations: 1.438 MiB)
0.001549 seconds (124 allocations: 248.203 KiB)
0.001251 seconds (107 allocations: 245.328 KiB)
 $a = -0.5753641445892759$
 $\Delta\mu = -1.0$

Out[82]:

X: Exponential($\theta=1.0$), m=5
Y: Exponential($\theta=2.0$)+(a, $\Delta\mu$), n=10
 $a=-0.5754$, $\Delta\mu=-1.0$



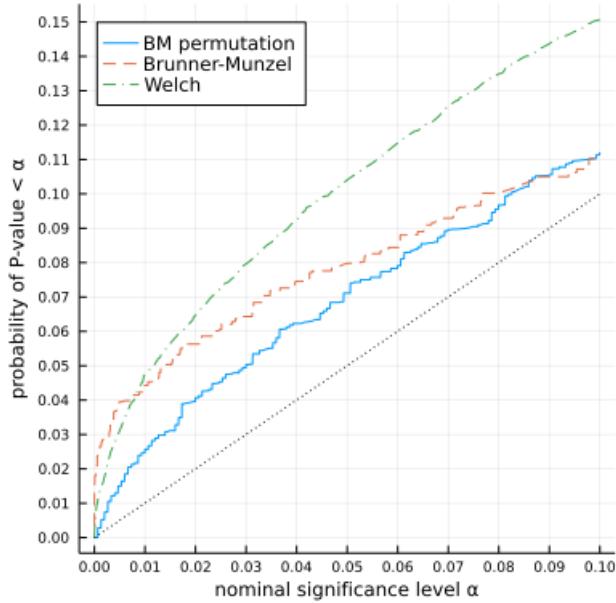
In [83]:

```
1 plot_pvals_with_perm()
2     distx = Exponential(1), disty = Exponential(2),
3     m = 10, n = 5, L = 10^4)
```

```
3.563473 seconds (10.18 k allocations: 1.895 MiB)
0.022323 seconds (132 allocations: 248.844 KiB)
0.014712 seconds (104 allocations: 245.234 KiB)
a = -0.5753641445892759
Δμ = -1.0
```

Out[83]:

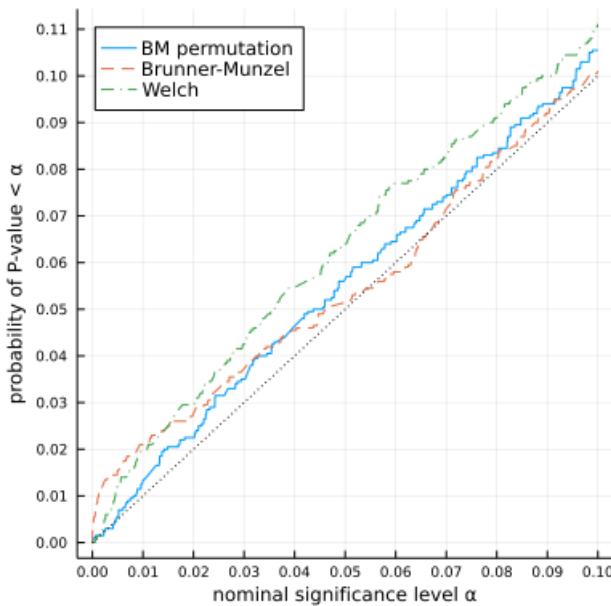
X: $\text{Exponential}(\theta=1.0)$, m=10
Y: $\text{Exponential}(\theta=2.0)+(a, \Delta\mu)$, n=5
 $a=-0.5754, \Delta\mu=-1.0$



```
In [84]: 1 plot_pvals_with_perm()
2     distx = Exponential(1), disty = Exponential(2),
3     m = 10, n = 10, L = 2000,
```

53.972017 seconds (2.24 k allocations: 17.257 MiB)
0.000422 seconds (131 allocations: 62.500 KiB)
0.000441 seconds (98 allocations: 58.484 KiB)
 $a = -0.5753641445892759$
 $\Delta\mu = -1.0$

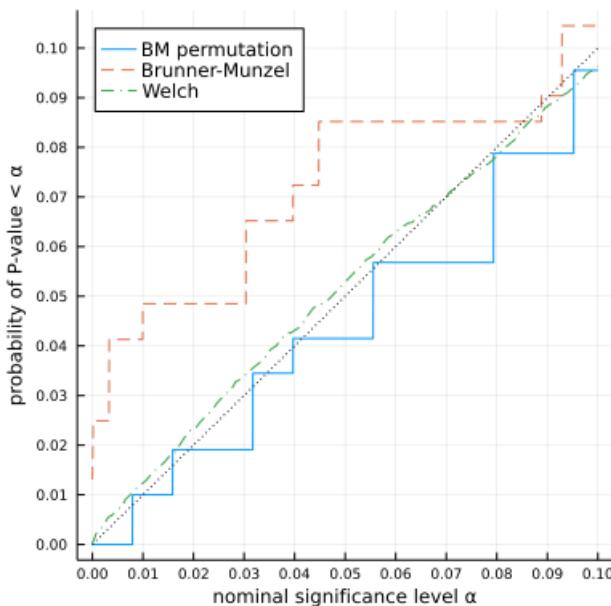
Out[84]: X: Exponential($\theta=1.0$), m=10
Y: Exponential($\theta=2.0$)+(a, $\Delta\mu$), n=10
 $a=-0.5754$, $\Delta\mu=-1.0$



```
In [85]: 1 plot_pvals_with_perm()
2     distx = Uniform(-1, 1), disty = Exponential(0.5773502691896257),
3     m = 5, n = 5, L = 10^4)
```

0.277728 seconds (32.46 k allocations: 2.343 MiB, 13.42% compilation time)
0.001420 seconds (125 allocations: 247.125 KiB)
0.024834 seconds (18.79 k allocations: 1.204 MiB, 95.25% compilation time)
 $a = -0.5370568188698568$
 $\Delta\mu = -0.5773502691896257$

Out[85]: X: Uniform(a=-1.0, b=1.0), m=5
Y: Exponential($\theta=0.57735$)+(a, $\Delta\mu$), n=5
 $a=-0.5371$, $\Delta\mu=-0.5774$



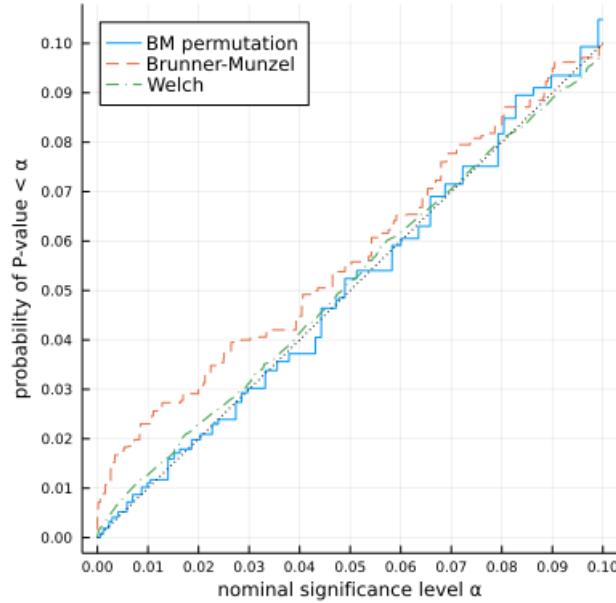
In [86]:

```
1 plot_pvals_with_perm()
2     distx = Uniform(-1, 1), disty = Exponential(0.5773502691896257),
3     m = 7, n = 7, L = 10^4)
```

```
3.938512 seconds (10.18 k allocations: 1.629 MiB)
0.001489 seconds (125 allocations: 248.078 KiB)
0.004222 seconds (97 allocations: 244.891 KiB)
a = -0.5370568188698568
Δμ = -0.5773502691896257
```

Out[86]:

X: Uniform(a=-1.0, b=1.0), m=7
Y: Exponential(θ=0.57735)+(a, Δμ), n=7
a=-0.5371, Δμ=-0.5774



In [87]:

```

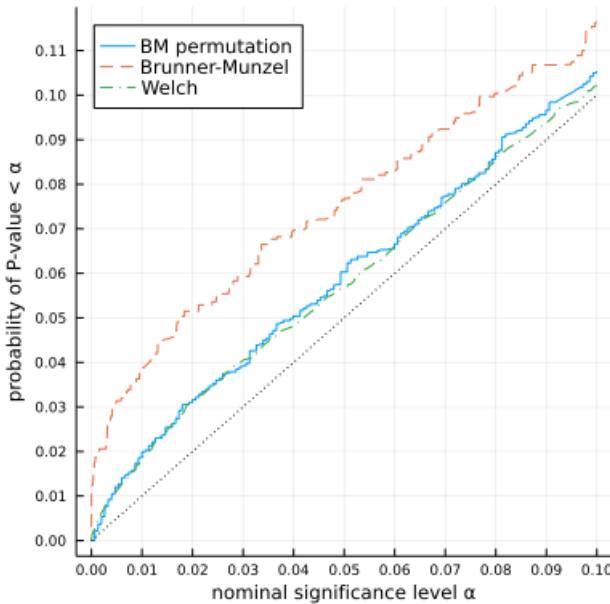
1 plot_pvals_with_perm(
2     distx = Uniform(-1, 1), disty = Exponential(0.5773502691896257),
3     m = 5, n = 10, L = 10^4)

```

3.505088 seconds (10.19 k allocations: 1.438 MiB)
 0.017648 seconds (131 allocations: 248.562 KiB)
 0.014763 seconds (104 allocations: 245.250 KiB)
 $a = -0.5370568188698568$
 $\Delta\mu = -0.5773502691896257$

Out[87]:

$X: \text{Uniform}(a=-1.0, b=1.0), m=5$
 $Y: \text{Exponential}(\theta=0.57735) + (a, \Delta\mu), n=10$
 $a=-0.5371, \Delta\mu=-0.5774$



In [88]:

```

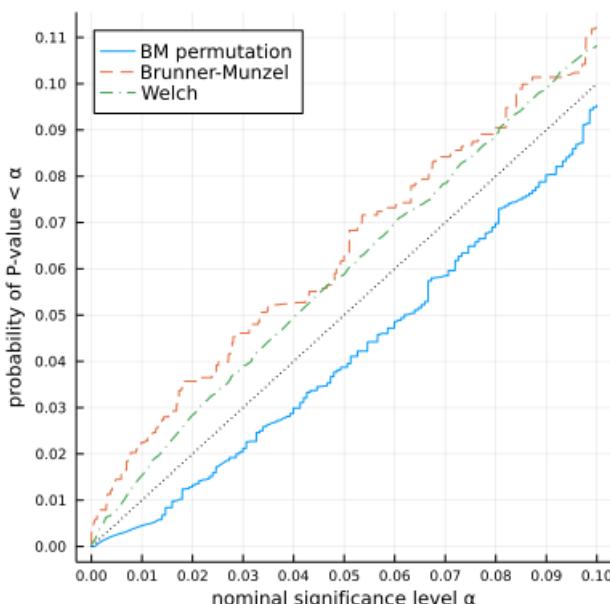
1 plot_pvals_with_perm(
2     distx = Uniform(-1, 1), disty = Exponential(0.5773502691896257),
3     m = 10, n = 5, L = 10^4)

```

3.527738 seconds (10.18 k allocations: 1.895 MiB)
 0.023244 seconds (127 allocations: 248.484 KiB)
 0.032098 seconds (99 allocations: 245.344 KiB)
 $a = -0.5370568188698568$
 $\Delta\mu = -0.5773502691896257$

Out[88]:

$X: \text{Uniform}(a=-1.0, b=1.0), m=10$
 $Y: \text{Exponential}(\theta=0.57735) + (a, \Delta\mu), n=5$
 $a=-0.5371, \Delta\mu=-0.5774$



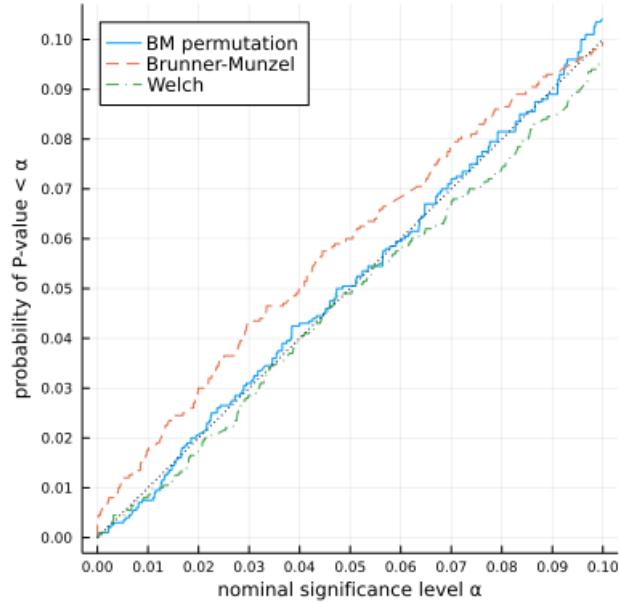
In [89]:

```
1 plot_pvals_with_perm()
2     distx = Uniform(-1, 1), disty = Exponential(0.5773502691896257),
3     m = 10, n = 10, L = 2000)
```

```
53.395587 seconds (2.22 k allocations: 17.257 MiB)
0.016941 seconds (130 allocations: 62.453 KiB)
0.014504 seconds (102 allocations: 58.750 KiB)
a = -0.5370568188698568
Δμ = -0.5773502691896257
```

Out[89]:

X: Uniform(a=-1.0, b=1.0), m=10
Y: Exponential(θ=0.57735)+(a, Δμ), n=10
a=-0.5371, Δμ=-0.5774



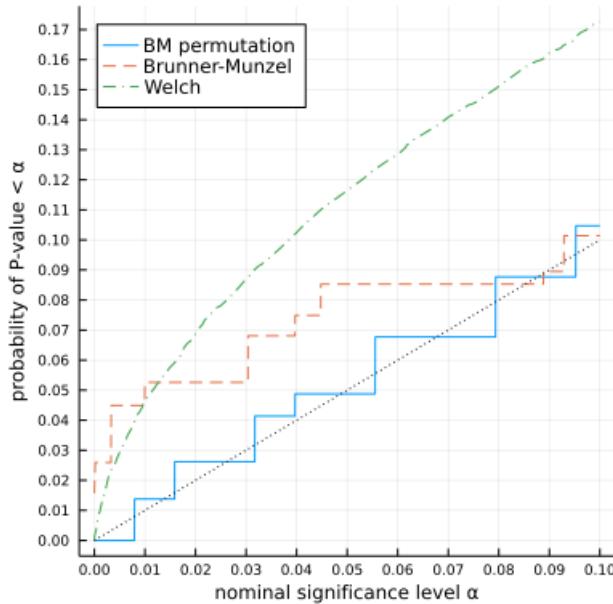
In [90]:

```
1 plot_pvals_with_perm(
2     distx = LogNormal(), disty = LogNormal(1), m = 5, n = 5, L = 10^4)
```

0.306429 seconds (35.60 k allocations: 2.486 MiB, 14.79% compilation time)
 0.001561 seconds (127 allocations: 247.188 KiB)
 0.017950 seconds (104 allocations: 244.688 KiB)
 $a = -1.4744426128871542$
 $\Delta\mu = -2.8329677996379363$

Out[90]:

X: $\text{LogNormal}(\mu=0.0, \sigma=1.0)$, m=5
 Y: $\text{LogNormal}(\mu=1.0, \sigma=1.0)+(a, \Delta\mu)$, n=5
 $a=-1.4744, \Delta\mu=-2.833$



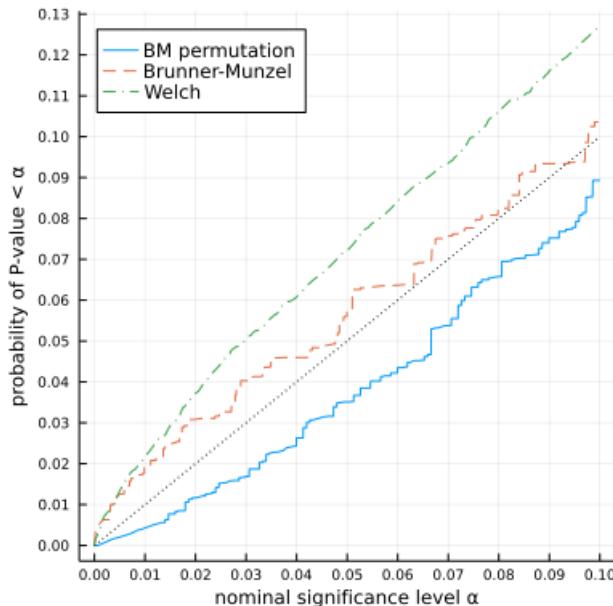
In [91]:

```
1 plot_pvals_with_perm(
2     distx = LogNormal(), disty = LogNormal(1), m = 5, n = 10, L = 10^4)
```

3.434650 seconds (10.18 k allocations: 1.438 MiB)
 0.024491 seconds (131 allocations: 248.484 KiB)
 0.015069 seconds (104 allocations: 245.250 KiB)
 $a = -1.4744426128871542$
 $\Delta\mu = -2.8329677996379363$

Out[91]:

X: $\text{LogNormal}(\mu=0.0, \sigma=1.0)$, m=5
 Y: $\text{LogNormal}(\mu=1.0, \sigma=1.0)+(a, \Delta\mu)$, n=10
 $a=-1.4744, \Delta\mu=-2.833$



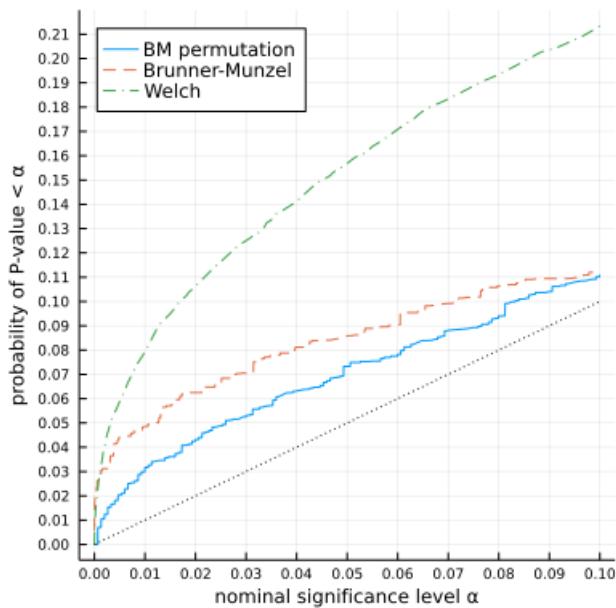
In [92]:

```
1 plot_pvals_with_perm()
2     distx = LogNormal(), disty = LogNormal(1), m = 10, n = 5, L = 10^4)

3.576869 seconds (10.18 k allocations: 1.895 MiB)
0.018933 seconds (125 allocations: 248.281 KiB)
0.005340 seconds (107 allocations: 245.781 KiB)
a = -1.4744426128871542
Δμ = -2.8329677996379363
```

Out[92]:

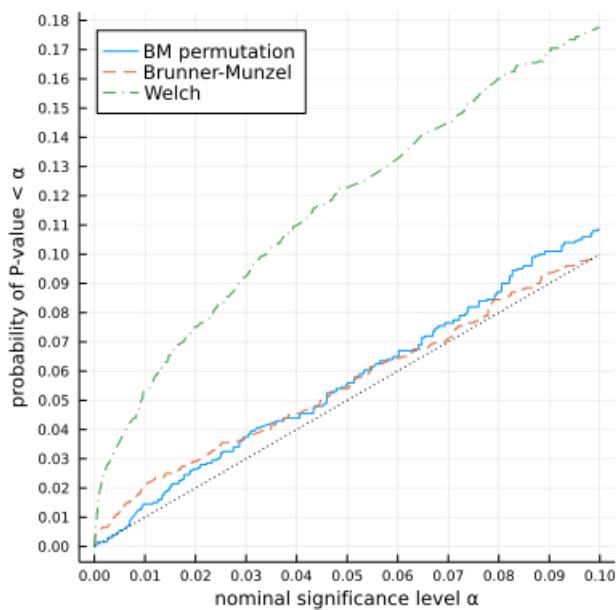
X: $\text{LogNormal}(\mu=0.0, \sigma=1.0)$, m=10
Y: $\text{LogNormal}(\mu=1.0, \sigma=1.0) + (a, \Delta\mu)$, n=5
 $a=-1.4744, \Delta\mu=-2.833$



```
In [93]: 1 plot_pvals_with_perm(  
2     distx = LogNormal(0), disty = LogNormal(1), m = 10, n = 10, L = 2000)
```

```
54.768207 seconds (2.23 k allocations: 17.257 MiB)  
0.000670 seconds (226 allocations: 70.969 KiB)  
0.000532 seconds (97 allocations: 58.469 KiB)  
a = -1.4744426128871542  
 $\Delta\mu$  = -2.8329677996379363
```

```
Out[93]: X: LogNormal( $\mu=0.0, \sigma=1.0$ ), m=10  
Y: LogNormal( $\mu=1.0, \sigma=1.0$ ) $+(a, \Delta\mu)$ , n=10  
a=-1.4744,  $\Delta\mu=-2.833$ 
```



```
In [ ]: 1
```