1 Writing

By Jojo Aboaf and Marty Wells

I am a visual human. Lets begin!

```julia
using GRUtils
using StatsBase
```

2 Abstract

- We Construct a general methods to analyze permutations.
- We present an empirical method for analyzing ordered in order to test for nationality bias in International Surf Competitions.
- We show this method can be applied in other settings, particularly for random graphs.

To state what we are setting out to do in its full generality simply:

- We are interested in a (finite) set of things, $E = \{e_1, e_2, \dots, e_d\}$
- The number of things we are interested in is $|E| = d$
- We observe some data, $Y = (y_i)_i^N$
- Each data point provides some information about a subset $\{e_a, e_b, \dots, e_c\} \subseteq E$.
- The number of things we observe is $|\{e_a, e_b, \dots, e_c\}| = k$
- We can observe
  - a full ordering, for example $[e_c \, e_a \dots e_b]$
  - or a partial ordering, for example $[e_c \, \{e_a, \dots, e_b\}]$, equivalently $[e_c \, \{e_b, \dots, e_a\}]$
  - or no ordering at all.

2.1 Concrete

Does nationality influence the World Surf League Championship Tour? If so, how?, to what degree?, and does this vary depending on nationality?

Ordered Data: Recommendation systems attempt to leverage information regarding one′s preferences to suggest new content (e.g. music, movies) or products (e.g. books). Ranked-choice voting is used for local, provincial/state, and national level elections across the globe. Many institutions, such as Cornell University, use ranked-choice for its elections. In sports, team or athlete rankings frequently determine season schedules or tournament structures. And in games or general forms of competition, outcomes can be naturally expressed as an ordering of the set of participants.

Graphical Data: Graphs are everywhere and everyone loves graphs and so on...

3 Introduction

The World Surf League (WSL) is the most prominent organizer of international surf compeitions. Each year the WSL organizes a variety of ”tours” which include Mens and Womens versions of Big Wave events, the ”Longboard Tour”, the ”Qualifying Series”, and the ”Championship Tour” (CT).

3.1 Format

Each year, the 32 highest ranked (shortboard) surfers are invited to participate in the ”Championship Tour” (CT), which constists of 11 surf compeitions in 7 different countries. Each competition has 7 rounds, each consisting of 1 to 16 heats, and each heat has 2 to 3 surfers. Within a heat, a surfer may attempt to ride any number of waves, but their final heat score is the sum of their two highest scoring waves. The surfer with the highest heat score places 1st in the heat, the surfer with the next highest heat score places 2nd in the heat. In some rounds, heats consist of 3 surfers, in which case the surfer with the third highest heat score will place 3rd in the heat. Each round has a rule that determines which surfers advance, and what (round,heat) they advance to. Surfers that do not advance ”exit” the event, and are given some number of points (the closer the exit is to the compeitions final round, the higher the amount of points).

INSERT 2019 Event Format Drawing Here: Note to self: add number of points given at exit to fig.

3.2 Data

We collected some incredibly rich data on surf compeitions from the 2017, 2018, and 2019 seasons of the Mens World Championship Tour (WCT). Each year the World Surf League (WCT) holds 10 to 11 surf competitions, which are called ”events”. While the format of events have changed slightly between the 2017 and 2019 seasons, they are all very similar. Each event consists of 7 rounds, and within each round there are some number of heats. A heat is the level at which intra-athlete competition takes place and may consist of 2 or 3 surfers. Throughout a timed heat (usually between 22 and 35 minutes), each athlete may surf any number of waves and their ”heat score” is the sum of the scores of their two highest scoring waves.

Question: How are waves scored? In any given heat, there is judging panel comprised of 5 judges. Anytime a surfer attempts to ride a wave, each judge analyzes the surfer’s ride with respect to the following criteria and write down a score:

- Commitment and degree of difficulty
- Innovative and progressive maneuvers
- Combination of major maneuvers
- Variety of maneuvers
- Speed, power, and flow

Note: that Different elements of this list may be emphasized more or less depending on the location, conditions, and changes of conditions. (Chapter 13 Article 182) Additionally, the General Judging Rules (Chapter 13; Article 183; Section 1,2) state that judges should be visually separated, should not discuss scores, and may not change their scores.

Though there are 5 judges and 5 scores, the score a surfer receives, called the ”wave score”, is the trimmed mean of the scores given by the 5 judges. This is very important information and particularly interesting because it distinguishes the *existence* of biased judges from the effect of biased judging (if it exists).

However, there are some intricacies. The judging process does not only consist of 5 judges. For each mens CT event, there is 1 international Head Judge, 7 international judges, and 1 international priority Judge (Chapter 13, Article 179.01). And Chapter 13, article 179.17 states that ”At CT Events, the number of judges from any one regional area is limited to 3”. For each heat, the WSL Head Judge must assure there are at least 5 judges on the panel for each heat and that they are a subset of the 7 International Judges and 1 International Head Judge (Chapter 13, Article 179.13).

5 judges are selected from a pool of 8 judges to form a panel for a heat. During that heat, when a surfer rides a wave, each judge independently observes the ride and writes down a score (based on some broadly defined criteria), which is some number in {0.0, 0.1, 0.2, ..., 9.8, 9.9, 10.0}.

3.3 Judging Panel Data

Question: What is the definition of a panel? What type of data is it? For any given heat, there are 5 judges on the judging panel. Anytime a surfer rides a wave, each of them observes the way

What does a panel look like? A lot of very different things that may seem very similar BUT ARE NOT. For example:

```julia
include("EquivPanelData.jl")
display(eqPanels[1])
display(eqPanels[3])
display(eqPanels[5])
```

```
1-element Array{Pair,1}:
 Float16[3.5, 4.5, 4.0, 4.0, 4.0] => [:AUS, :AUS, :USA, :BRA, :BRA]
5-element Array{Pair,1}:
 Float16[1.0] => [:BRA]
 Float16[1.1] => [:USA]
 Float16[1.2] => [:AUS]
 Float16[1.3] => [:PRT]
 Float16[1.5] => [:ESP]
4-element Array{Pair,1}:
 Float16[0.5, 0.5] => [:ESP, :FRA]
      Float16[0.8] => [:AUS]
      Float16[1.0] => [:BRA]
      Float16[1.2] => [:USA]
```

4 Motivation

Our goal is to determine if judges have a tendency to give higher scores to surfers that share their same nationality.

The straight forward approach is to identify all of the ”matching judges” for each wave, which are the judges scoring a surfer that have the same nationality as the surfer.

Though we will use the term ”matching judge(s)” throughout the paper, it is important to keep in mind that it is both the Athlete origin and Judge origin which determine if a judge is a ”matching judge” or a ”non-matching judge”.

Some instinctual approaches:

- H_0: Mean of matching judges = mean of non-matching judges
  - This approach can work, but not always. It will fail if

Country	Judges (n)	Waves	Per Wave	Mean	SD	T-Value	P-Value
Australia	4972	3250	1.529846	0.017198	0.351363	3.451332	0.000562
Brazil	8340	5234	1.593428	0.009411	0.330016	2.604384	0.009220
France	682	682	1	0.048167	0.320114	3.929514	0.000094
French Polynesia	10	10	1	0.010000	0.425617	0.074299	0.942398
Hawaii	122	122	1	0.024795	0.298571	0.917272	0.360825
Portugal	203	203	1	0.021552	0.358535	0.856443	0.392767
South Africa	257	257	1	0.044261	0.341772	2.076101	0.038883
United States	1277	1131	1.129089	0.026625	0.329370	2.888684	0.003934

5 Methods

5.1 Lets explore the Data!

We have lots of missing Judge Origins from panels in the 2017 World Surf League season so we will omit the 2017 season ... for now (This begs an intersting question which we should return to later).

We have constructed a multidimensional array, aka an m-way, cross classified, contingency table. We have m classification factors:

- WSL Season
- Event
- Round
- Heat
- Ahtlete Origin
- Judge Origin
- Size of Partition of Panel (Max Rank)

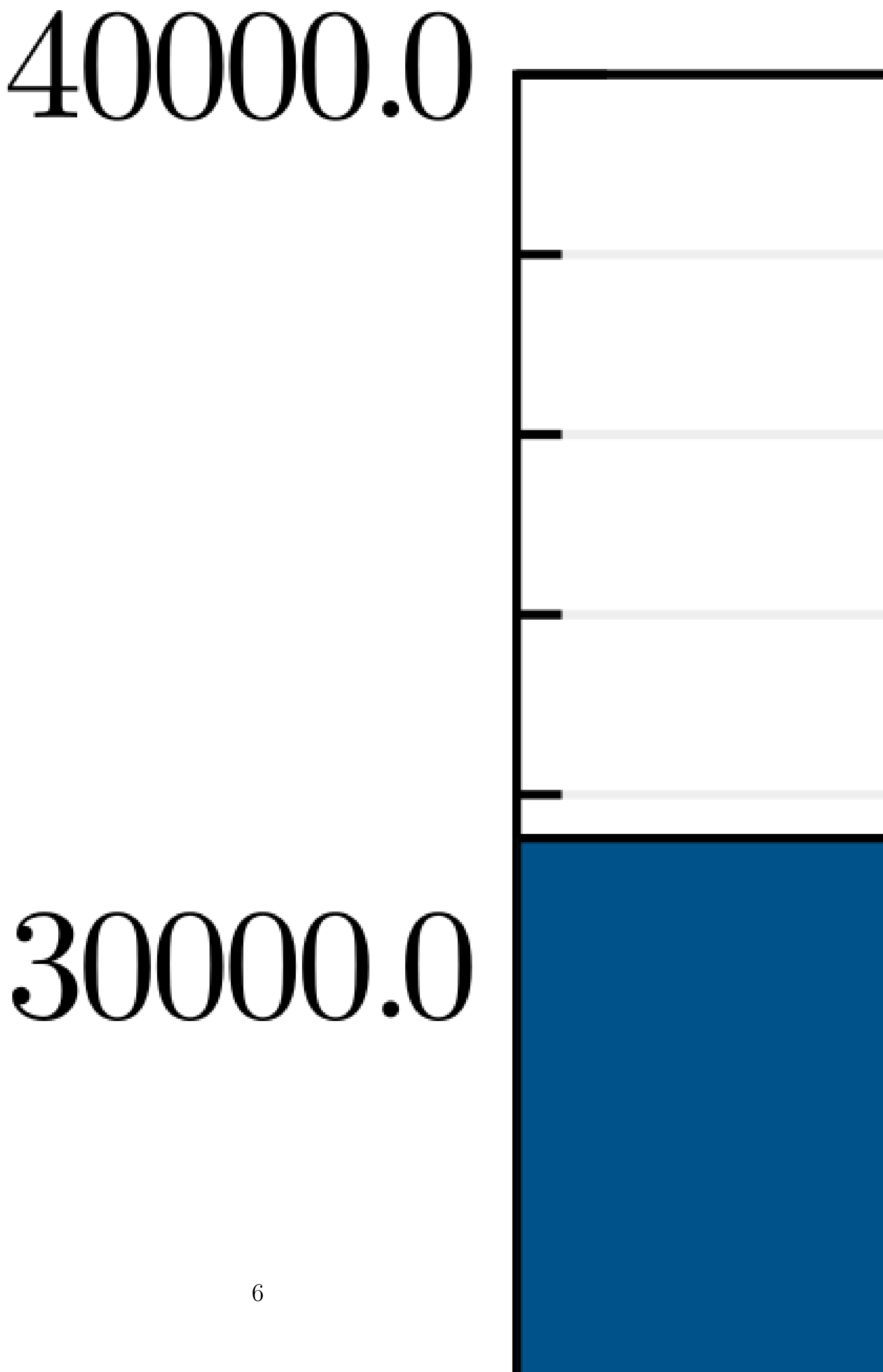
- Rank of Judge

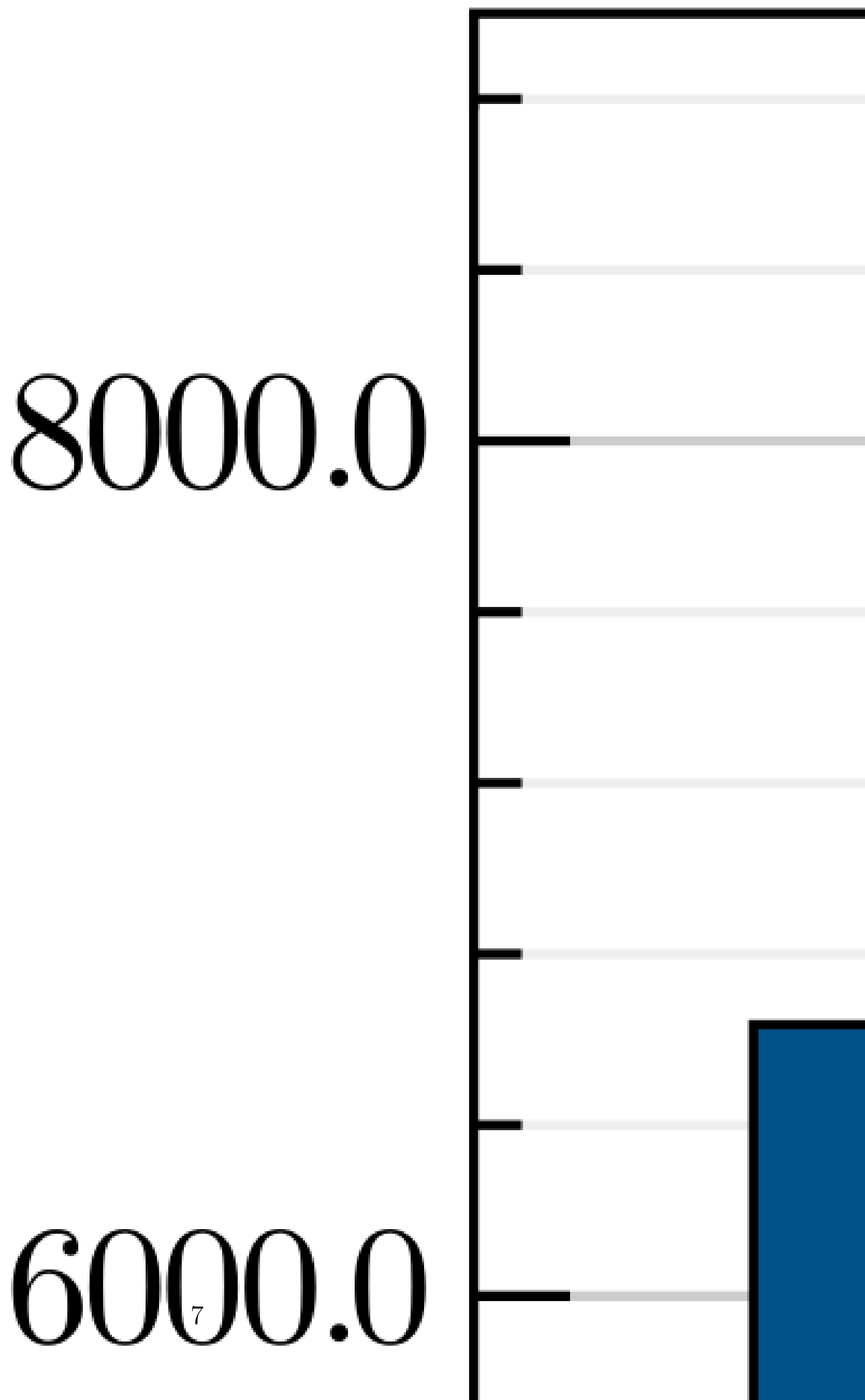
```
include("RankingTensor.jl")
println("m = $(ndims(info))")
println(size(info))

function marginalBarPlot(M::Array{T,N}, d::Integer) where {T,N}
        dIndex_(h::Integer) = [ i==d ? h : Colon() for i in 1:N]
        marginal_d = [sum( M[ dIndex_(h)... ] ) for h in 1:size(M)[d] ]
        savefig(
                "visuals/marginal_$(vars[d]).png",
                barplot(
                        map(x-> "$(x)", sort(collect(keys(ToInd[vars[d]]))) ),
                        marginal_d
                )
        )
        return marginal_d
end

for d in length(vars) marginalBarPlot(info, d) end
```

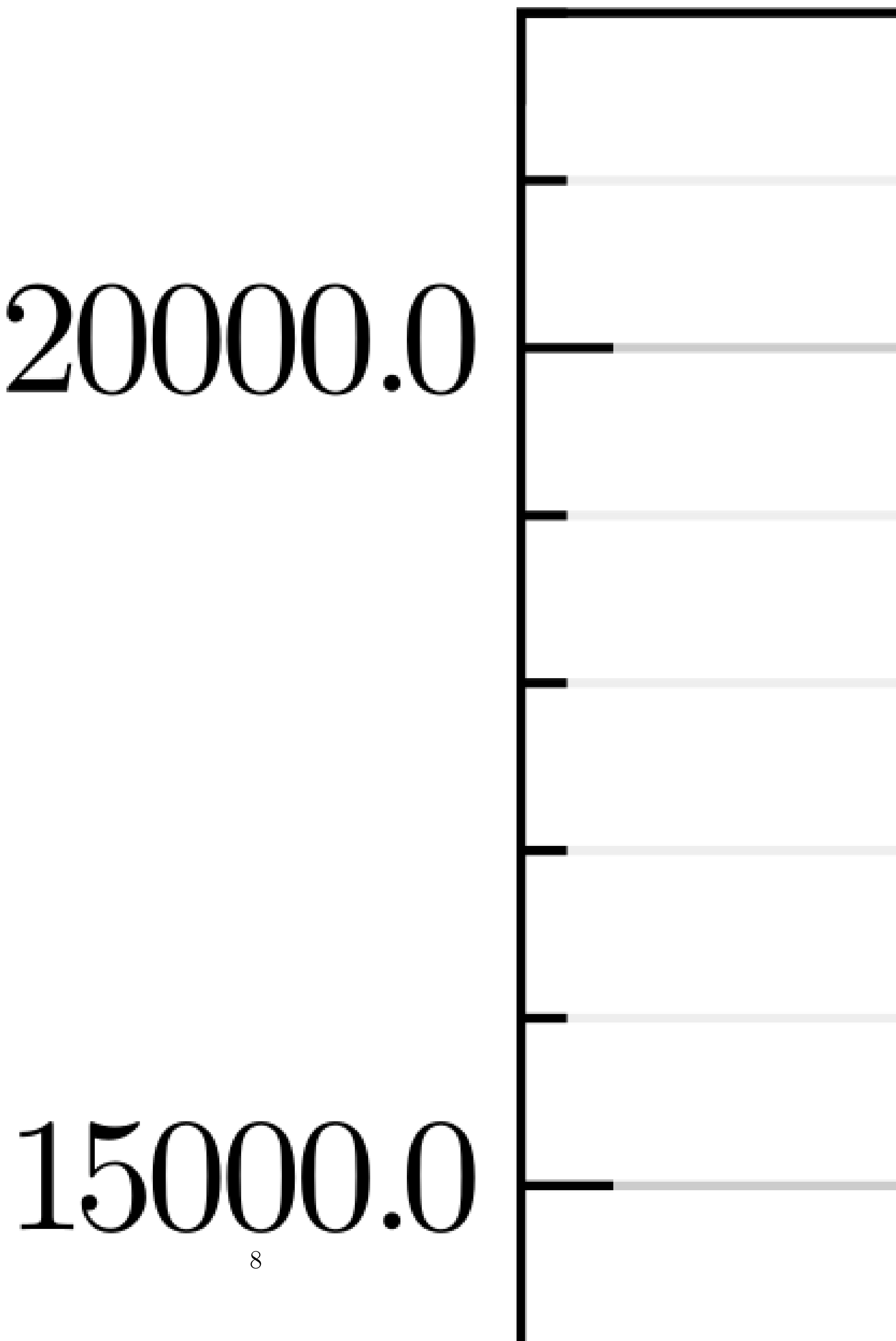
```
m = 8
(2, 10, 7, 16, 10, 7, 5, 5)
```

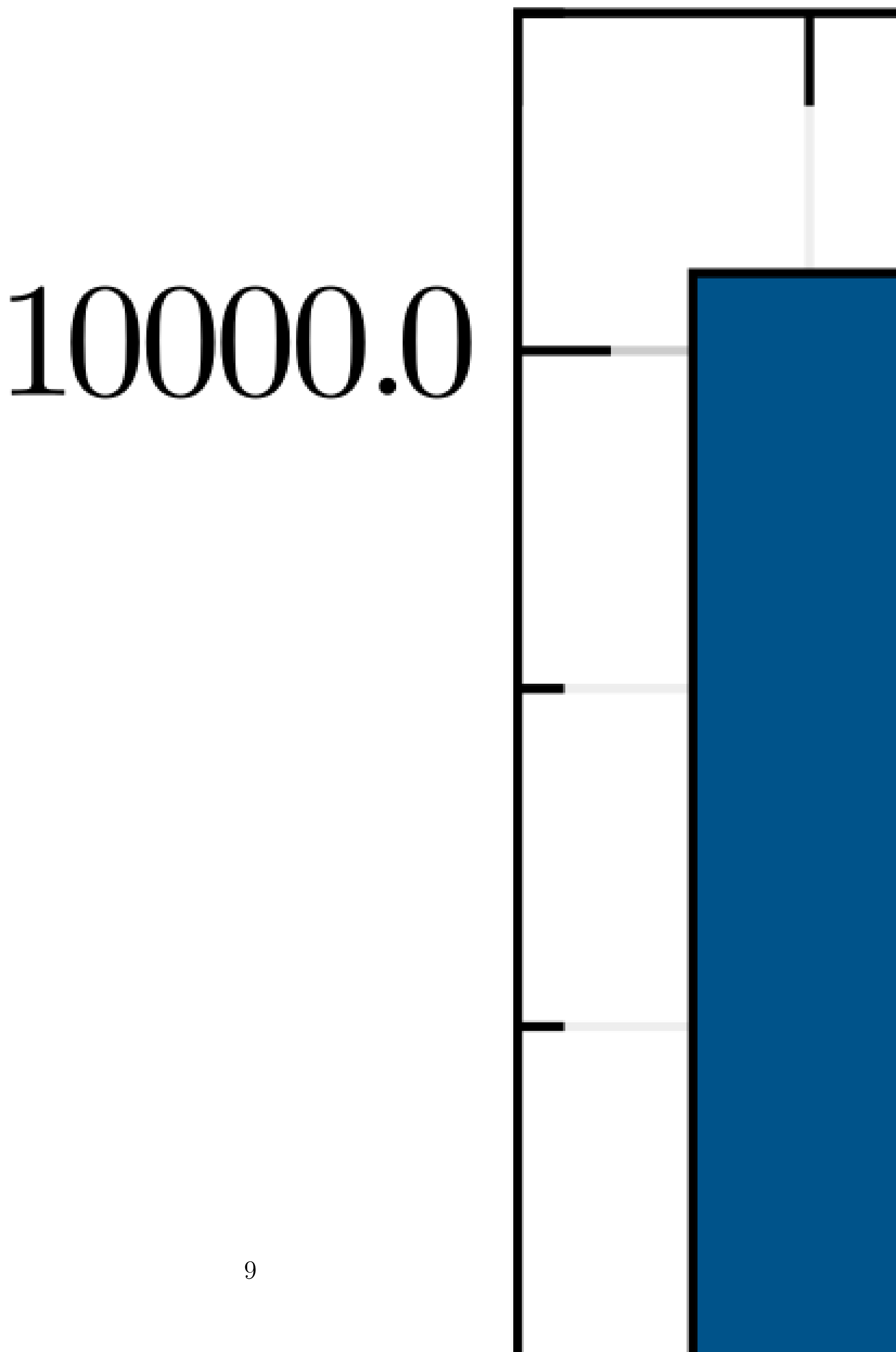
6 Definitions

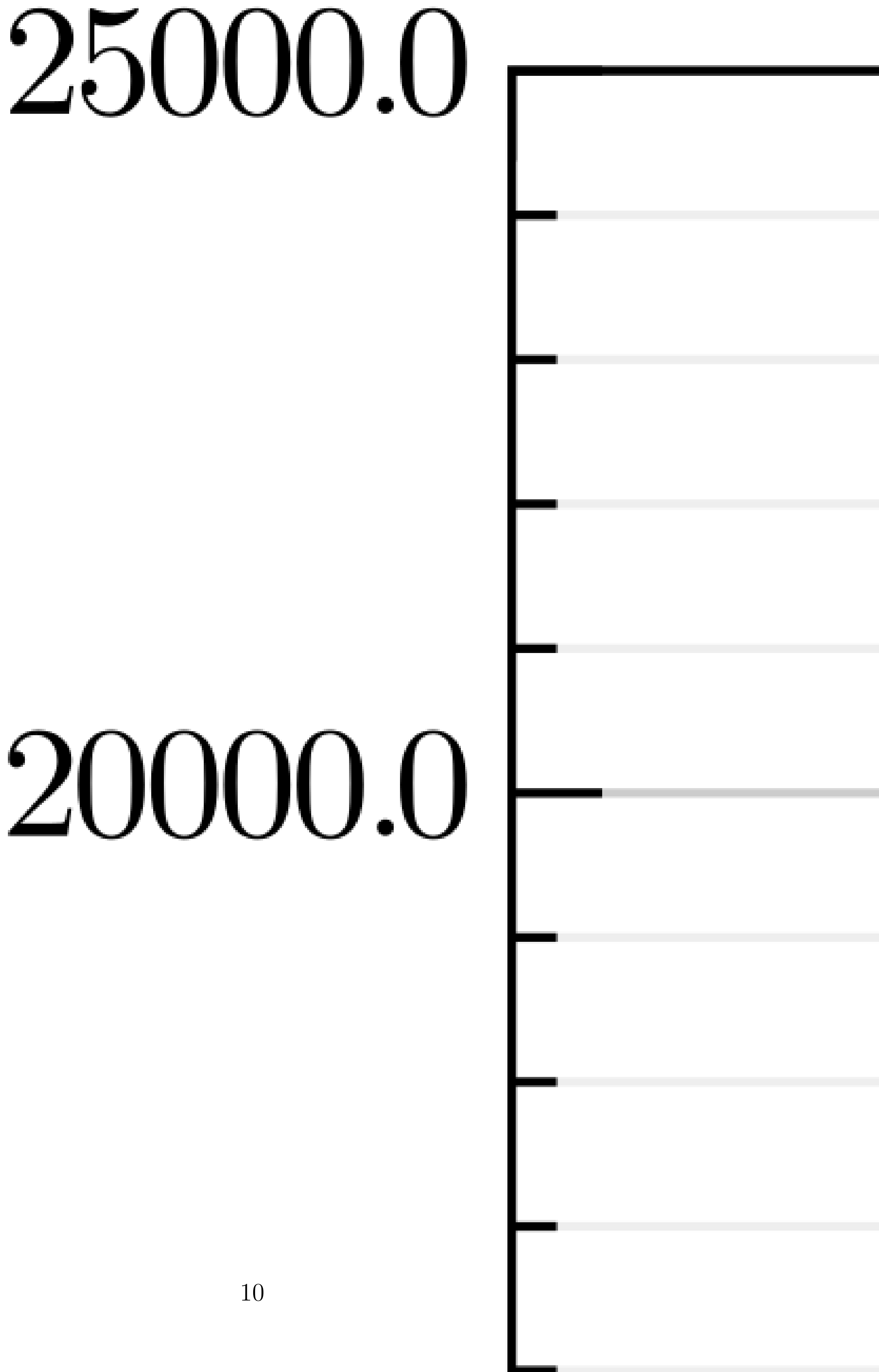
- **The Symmetric Group on a set,** X is $S_X := Isomorphisms(X, X)$. When $|X| < \infty$,$S_X = \{\tau : X \to X \mid Image(\tau) = X\} = \{\tau : X \to X \mid \{\tau(x) \mid x \in X\} = X\}$
- $S_d := S_{\{1,\dots,d\}}$.
- When G is a Group, and $\mathbb{F}$ is a Field, the **Group Algebra of G over** F, denoted $\mathbb{F}[G]$, is the space of formal linear combinations of elements of G. Elements of $\mathbb{F}[G]$ are of the form: $c_1g_1 + \cdots + c_ng_n = \sum_{i=1}^n c_ig_i$, where $c_i \in \mathbb{F}, g_i \in G$. Note that $i \neq j \implies g_i \neq g_j$ because G is a set of elements, so no element occurs with multiplicity. For example, $c_1g + c_2g \notin \mathbb{F}[G]$ whereas $(c_1 + c_2)g \in \mathbb{F}[G]$.
- $\mathbb{F}[S_d]$ is comprised on formal linear combinations of elements of S_d. This may be understood as two different ways:
  - There exists a function $a : S_n \to \mathbb{F}$ and $A := \sum_{\tau\in S_d} a(\tau)\tau \quad \in \mathbb{F}[S_n]$.
  - Or: $A = [\pi_1 \dots \pi_{d!}] \begin{bmatrix} a_{\pi_1} \\ \vdots \\ a_{\pi_{d!}} \end{bmatrix} = \sum_{\tau\in S_d} a_\tau\tau \in \mathbb{F}[S_d]$,where $\tau \in S_d, a_\tau \in \mathbb{F}$.
- **Addition in** $\mathbb{F}[S_n]$ is defined by $A + B = (\sum_\tau a_\tau\tau) + (\sum_\tau b_\tau\tau) := \sum_\tau(a_\tau + b_\tau)\tau$
- **Scalar Multiplication in** $\mathbb{F}[S_d]$ is $c(A) = c(\sum_\tau a_\tau\tau) = \sum_\tau ca_\tau\tau$.
- Multiplication in $\mathbb{F}[S_d]$ is *defined* by: $A*B = (\sum_\tau a_\tau\tau)*(\sum_\pi b_\pi\pi) := \sum_{\gamma\in S_d}(\sum_{\tau,\pi|\tau\pi=\gamma} a_\tau b_\pi)\gamma = \sum_{\gamma\in S_d}(\sum_{\tau\in S_d} a_\tau b_{\tau^{-1}\gamma})\gamma$.

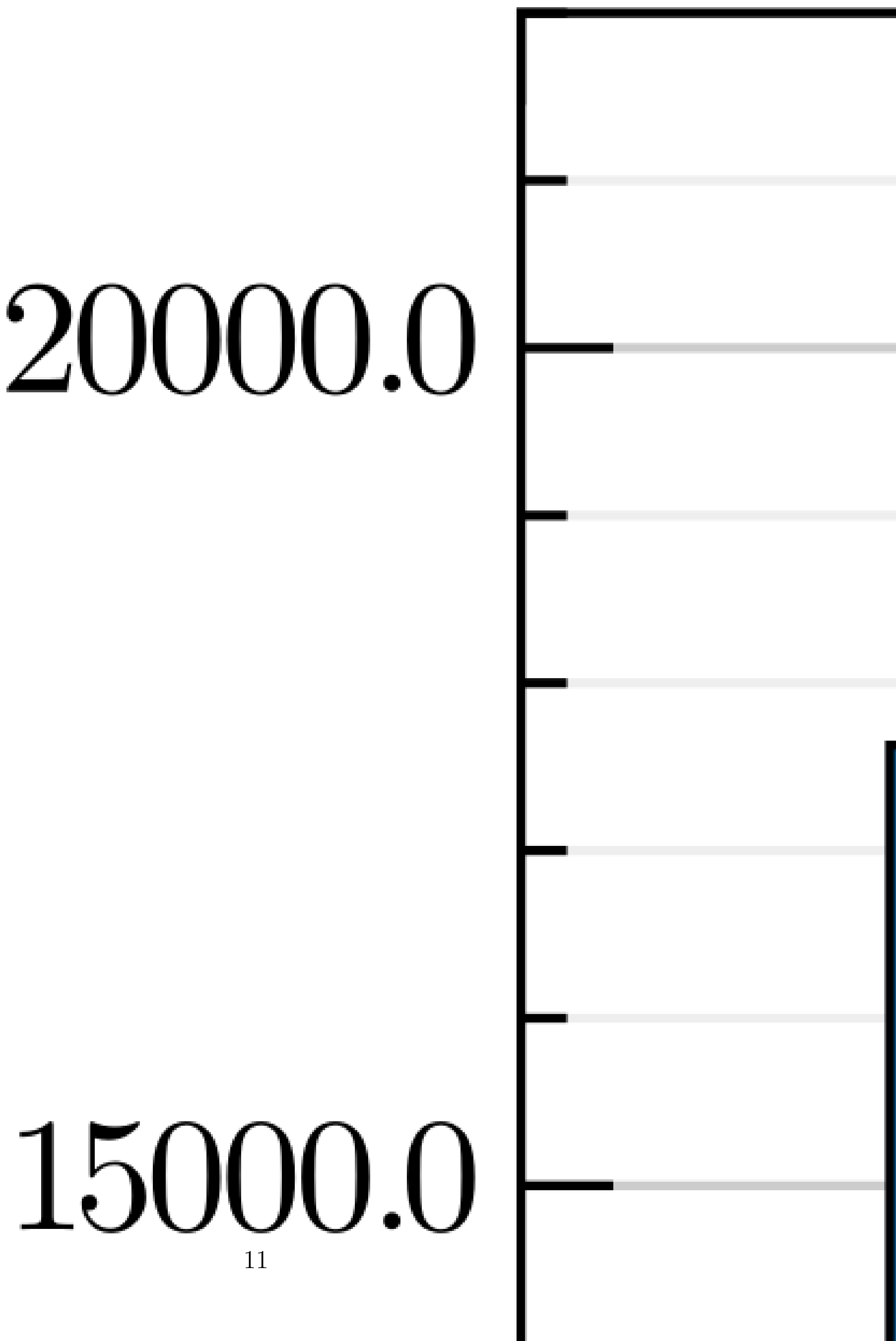

40000.0
30000.0

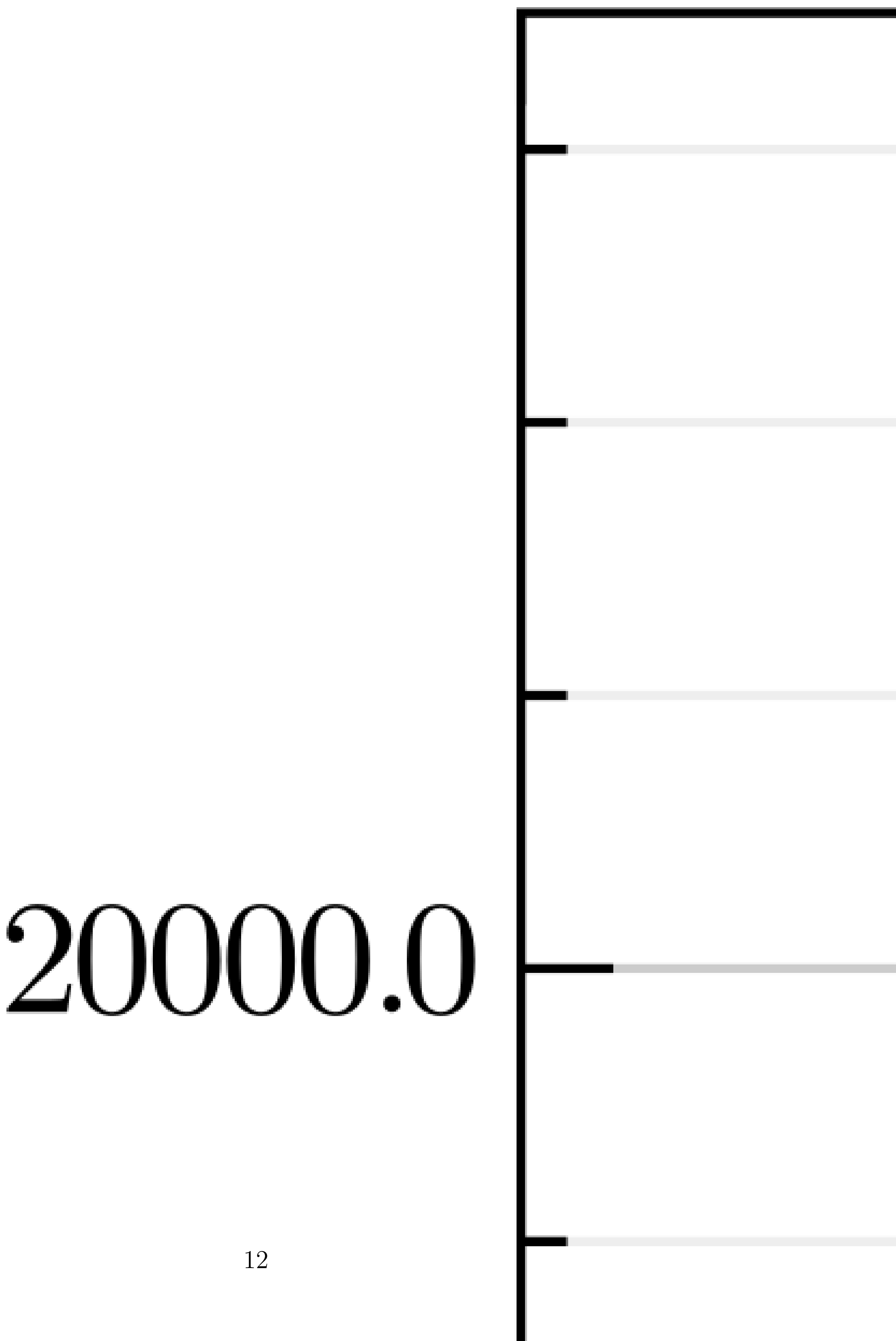

8000.0
6000.0

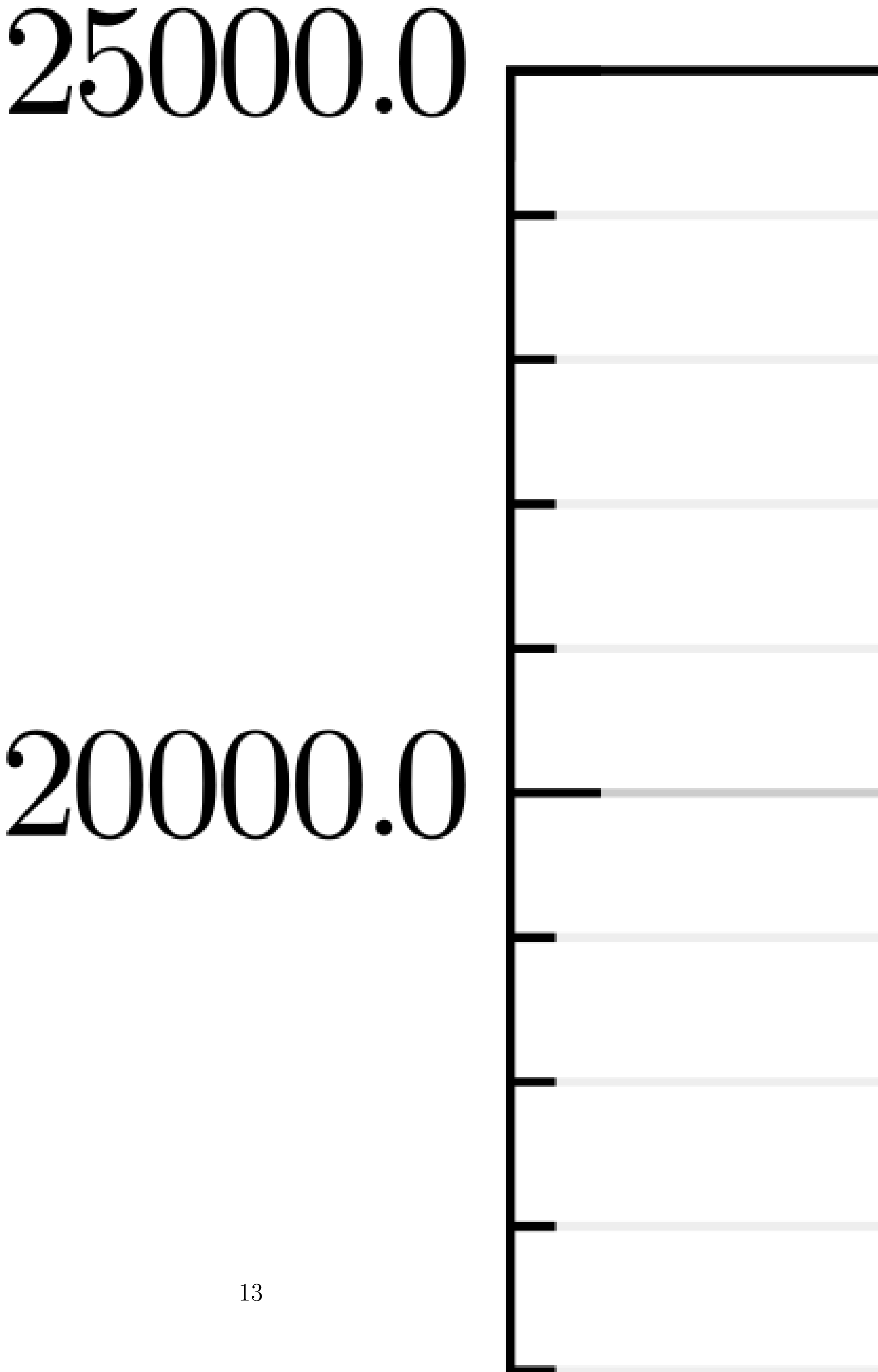

20000.0
15000.0


10000.0


25000.0
20000.0


20000.0
15000.0


20000.0


25000.0
20000.0

We should not over complicate $*$. The *definition* of $*$ may look odd, but it exactly the same as our basic understanding of multiplication: $(\sum_\tau a_\tau \tau) * (\sum_\pi b_\pi \pi) = \sum_\tau (a_\tau \tau) * (\sum_\pi b_\pi \pi) = \sum_\tau \sum_\pi (a_\tau \tau) * (b_\pi \pi) = \sum_\tau \sum_\pi a_\tau b_\pi \tau\pi = \sum_{\gamma \in S_d} (\sum_{\tau,\pi | \tau\pi = \gamma} a_\tau b_\pi)\gamma = (\sum_\tau a_\tau \tau) * (\sum_\pi b_\pi \pi)$ Even though $(\sum_\tau a_\tau \tau) * (\sum_\pi b_\pi \pi)$ is equal to the intuitive form, $\sum_\tau \sum_\pi a_\tau b_\pi \tau\pi$, the latter is not an element of the the group algebra because elements are repeated in the sum, hence our chosen definition. Also, $*$ merely extends multiplication in the Field,$\cdot : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$, and the operation in the group, $\circ$, by $a_\tau \tau * b_\pi \pi = a_\tau \cdot b_\pi \tau \circ \pi = a_\tau b_\pi \tau\pi$, where the last equality is simply notation-reduction.

A measure on S_d, is an element of the group algebra $\mathbb{C}[S_n]$. A measure on S_d, $F = \sum_{\tau \in S_d} f_\tau \tau$, is a probability measure on S_d if and only if $\forall \tau \in S_d f_\tau \geq 0$ and $\sum_{\tau \in S_d} f_\tau = 1$.

A linear representation of a group G, is a group homomorphism, $\rho : (G, \circ) \to (GL(V), \cdot)$. A group homomorphism satisfies:

- $\forall x, y \in G \quad \rho(x \circ y) = \rho(x) \cdot \rho(y)$ where $\cdot$ is multiplication in $GL(V)$.
- $\rho(e) = I$, where e is the identity element in G and I is the identity element in $GL(V)$.
- $\forall x \in G, \rho(x^{-1}) = \rho(x)^*$, where * denotes involution in $GL(V)$.

The representation of a measure F, is: $\hat{F} := \sum_{\tau \in S_n} P(\tau)\rho(\tau)$. Note: This is sometimes called the Fourier transform at a representation, I avoid that lingo.

Convolution of two functions on S_d is a binary operation $A * B := \sum_{\tau \in S_n} a(\tau) b(\tau^{-1} g)$.

Note: $\widehat{A * B} = \sum_\gamma (A * B)(\gamma)\rho(\gamma) = \sum_\gamma \sum_\tau a(\gamma\tau^{-1}) b(\tau)\rho(\gamma) = \sum_\tau \sum_{\gamma\tau} a(\gamma\tau\tau^{-1}) b(\tau) \rho(\gamma\tau) = \sum_\tau \sum_{\gamma\tau} a(\gamma) b(\tau) \rho(\gamma)\rho(\tau) = \sum_\tau b(\tau)\rho(\tau) \sum_{\gamma\tau} a(\gamma)\rho(\gamma) = \sum_\tau b(\tau)\rho(\tau)\hat{A} = \hat{A} \sum_\tau b(\tau)\rho(\tau) = \hat{A} \cdot \hat{B}$

We take ρ_d the permutation representation acting on the vector space $V := \mathbb{R}^d$ with basis indexed by $\{1, \ldots, d\}$. So a typical element of V is of the form:

$$\begin{pmatrix} a^1 \\ a^2 \\ \vdots \\ a^n \end{pmatrix} = \begin{pmatrix} a^1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ a^2 \\ \vdots \\ 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a^n \end{pmatrix} = a^1 \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + a^2 \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} + \cdots + a^n \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} = a^1 e_1 + a^2 e_2 + \ldots a^n e_n$$

This could be rewritten as $\sum_{i \in \{1,\ldots,n\}} a^i e_i$. But remember, we are interested in functions that act on the basis.

Many authors call $*$ ”convolution”, however this terminology is superfluous.

7 Analysis

One thing I could change:

- instead giving every element of rth partition rank give them each:

{ N_r + i for i in 1:n_i} where N_r is Σrn_i ... but then any answer we give loses meaning

I wonder: M_r := max order for that wave B = 1*{order(J_i) = 1 }* *T = 1*{order(J_i) = M_r } B $\cup$ T only really meaninful for M_r>2

Wondering_1: if P(T | JUD*orig==ATH*orig) = P(T | JUD*orig != ATH*orig) Wondering_2: if Wondering_1 depends on [ ]*orig Wondering_3: if P(B ∪ T / M_r) = 2/M_r Wondering_4: if P(T / (JUD*orig==ATH_orig,M_r)) = 1/M_r

7.1 Method 1

```
ATHandJUD_orig = intersect(keys(ToInd[:ATH_orig]),keys(ToInd[:JUD_orig]) )

GivenMatchByM_r = [
sum( [ sum(info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],r,:]) for c in
ATHandJUD_orig ] )
for r in 1:5
]
println("Match conditional on Max Rank is:")
GivenMatchByM_r

TopGivenMatchByM_r = [
sum( [ sum(info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],r,1])
for c in ATHandJUD_orig])
for r in 1:5
]
println("Judge has Max Rank given Match (by Max Rank)")
TopGivenMatchByM_r

D_1 = TopGivenMatchByM_r ./ GivenMatchByM_r
```

```
Match conditional on Max Rank is:
Judge has Max Rank given Match (by Max Rank)
5-element Array{Float64,1}:
 1.0
 0.4969289265867213
 0.30612588354O8953
 0.21223286661753868
 0.18444995864350702
```

We would expect:

```
E_1 = [1/r for r in 1:5]
```

```
5-element Array{Float64,1}:
 1.0
 0.5
 0.3333333333333333
 0.25
 0.2
```

So we have:

```
χsq_1 = sum(GivenMatchByM_r)*sum( (D_1 .- E_1).^2 ./ E_1 )
```

```
137.795814443704
```

Now...

```
GivenMatchByM_randOrig = [
sum( info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],r,:] )
for r in 1:5, c in ATHandJUD_orig
]
println("Match conditional on Max Rank (rows) and Nationality (cols)")
```

```
display(GivenMatchByM_𝑟andOrig)

TopGivenMatchByM_𝑟andOrig = [
sum( info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],r,1] )
for r in 1:5, c in ATHandJUD_orig
]
println("Judge has Max Rank given Match (by Max Rank (rows) by Nationality (cols))")
println(TopGivenMatchByM_𝑟andOrig)

println(ATHandJUD_orig)
D_2 = TopGivenMatchByM_𝑟andOrig ./ GivenMatchByM_𝑟andOrig
```

```
Match conditional on Max Rank (rows) and Nationality (cols)
5×@*(6 Array(*@{Int64,2}:
  5    89    8   253   14   43
 30   917   73  1811  160  428
 72  1666  110  3061  282  751
 40  1133   50  2066  187  595
 16   360   16   581   57  179
Judge has Max Rank given Match (by Max Rank (rows) by Nationality (cols))
[5 89 8 253 14 43; 10 418 40 943 80 208; 25 474 28 984 88 220; 5 237 16 442
 39 125; 6 76 2 109 4 26]
Set([:PRT, :AUS, :ZAF, :BRA, :FRA, :USA])
5×@*(6 Array(*@{Float64,2}:
 1.0       1.0       1.0       1.0       1.0        1.0
 0.333333  0.455834  0.547945  0.520707  0.5        0.485981
 0.347222  0.284514  0.254545  0.321464  0.312057   0.292943
 0.125     0.209179  0.32      0.21394   0.208556   0.210084
 0.375     0.211111  0.125     0.187608  0.0701754  0.145251
```

We would expect:

```
E_2 = [ 1/r for r in 1:5, c in ATHandJUD_orig ]
```

```
5×@*(6 Array(*@{Float64,2}:
 1.0       1.0       1.0       1.0       1.0       1.0
 0.5       0.5       0.5       0.5       0.5       0.5
 0.333333  0.333333  0.333333  0.333333  0.333333  0.333333
 0.25      0.25      0.25      0.25      0.25      0.25
 0.2       0.2       0.2       0.2       0.2       0.2
```

So we have a total χ^2 of:

```
χsq_2 = sum(GivenMatchByM_𝑟andOrig)*sum( (D_2 .- E_2).^2 ./ E_2)

W = (D_2 .- E_2).^2 ./ E_2
χsqbyctry = [ sum(GivenMatchByM_𝑟andOrig[:,c])*sum(W[:,c]) for c in 1:6]

M_𝑟GivenMatch =[
sum(info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],3:5,:] )
for c in ATHandJUD_orig
]
OrderIsM_𝑟 = [
sum( [ sum(info[:,:,:,:,ToInd[:ATH_orig][c],ToInd[:JUD_orig][c],r,1])
for r in 3:5 ] )
for c in ATHandJUD_orig
]
println(OrderIsM_𝑟 ./ M_𝑟GivenMatch)
```

```
[0.28125, 0.24912947135169358, 0.26136363636363635, 0.2689208128941836, 0.2
4904942965779467, 0.24327868852459017]
```

But!!!

```
include("PanelData.jl")

N = length(panels)
Ord_Parts = map(panel -> length.(panel) , panels)
Ord_Parts_Counts = countmap(Ord_Parts)
Ord_Parts_Props = Dict(
        [x=>Ord_Parts_Counts[x]/N
        for x in keys(Ord_Parts_Counts)]
)
```

```
Dict{Array{Int64,1},Float64} with 16 entries:
  [2, 2, 1]       => 0.0679787
  [1, 1, 1, 1, 1] => 0.0821078
  [2, 1, 2]       => 0.0538495
  [4, 1]          => 0.0557238
  [1, 2, 2]       => 0.0766292
  [3, 2]          => 0.0592561
  [3, 1, 1]       => 0.0537053
  [5]             => 0.0284025
  [1, 3, 1]       => 0.0864331
  [1, 1, 3]       => 0.0502451
  [1, 1, 2, 1]    => 0.0810265
  [1, 2, 1, 1]    => 0.0810986
  [2, 1, 1, 1]    => 0.0546424
  [1, 1, 1, 2]    => 0.0551471
  [2, 3]          => 0.0589677
  [1, 4]          => 0.0547866
```

$|\{g \in S_d | cycles(g) = 1^{k_1}, 2^{k_2}, \ldots, d^{k_d}\}| = \frac{d!}{\prod_{j=1}^{d} k_j! j_j^k} \implies P(1_1^k, \ldots, d_d^k) = \frac{\frac{d!}{\prod_{j=1}^{d} k_j! j_j^k}}{d!} = \frac{1}{\prod_{j=1}^{d} k_j! j_j^k}$
Panels is the Array of the observed Panels, each of which is ordered by score. We do not have a total order for every panel So a observed panel is an ordered partition. Y is an array of cycle counts. Below, λ, is the cycle counts.

```
Y = map(panel -> [count(==(i),length.(panel)) for i in 1:5], panels)
λ_Counts = countmap(Y)
λ_Obs = Dict([x=>λ_Counts[x]/N for x in keys(λ_Counts)])
λ_Thry = Dict(
        [K=> prod( [ 1//( factorial(K[j]) * j^K[j] ) for j in 1:5])
        for K in keys(λ_Counts) ]
)
χ_sq = length(keys(λ_Thry))*sum(
        [ (λ_Obs[x]-λ_Thry[x])^2 / λ_Thry[x]
        for x in keys(λ_Thry)]
)
println(χ_sq)
```

```
9.558900782521095
```

... $\chi 2$ is very large....

```
include("EquivPanelData.jl")
eqParts = map(eqPanel -> [count(==(i),length.(last.(eqPanel))) for i in 1:5], eqPanels )
eqParts_Counts = countmap(eqParts)
eqParts_Props = Dict([x=>eqParts_Counts[x]/N for x in keys(eqParts_Counts)])
Theory_Parts_Props = Dict(
        [K=> prod( [ 1//( factorial(K[j]) *  j^K[j] ) for j in 1:5])
        for K in keys(eqParts_Counts) ]
)
eqχ_sq = length(keys(eqParts_Props))*sum(
        [ (eqParts_Props[x]-Theory_Parts_Props[x])^2 / Theory_Parts_Props[x]
        for x in keys(eqParts_Props)]
)
```

```
0.567850258509503
```

Now eqPanels with No mulitplicity

```
Y = map(eqPanels) do panel
        x = unique.(last.(panel))
        return [ count( ==(i), length.(x) ) for i in 1:5]
end
nOrigs_Counts = countmap( map(x->sum([i*x[i] for i in 1:5]), Y))
nOrigs_Props = Dict([x=>nOrigs_Counts[x]/N for x in keys(nOrigs_Counts)])
λ_Counts = countmap(Y)
λ_Props = Dict([x=>λ_Counts[x]/N for x in keys(λ_Counts)])
λ_Thry_Cond_Origs = Dict(
        [ K => nOrigs_Props[sum([i*K[i] for i in 1:5])]/N * prod( [ 1//( factorial(K[j])
*  j^K[j] ) for j in 1:5])
        for K in keys(λ_Props) ]
)
eqNoM_cond_χ_sq = length(keys(λ_Thry_Cond_Origs))*sum(
        [ (λ_Props[x]-λ_Thry_Cond_Origs[x])^2 / λ_Thry_Cond_Origs[x]
        for x in keys(λ_Thry_Cond_Origs)]
)
```

```
324381.85070143273
```

what marty thought of

```
D = Dict(eqInfo)
χ_sq_byHT = []
χ_sq_byHT_nom = []
for heat in partitionBy("heatId")
        Panels = [ last.(D[x]) for x in heat[2] ]
        Panels_nom = [ unique.(last.(D[x])) for x in heat[2] ]
        n = length(Panels)
        λ_HT_Origs_Counts = countmap( map(x->sum(length.(x)), Panels_nom))
        λ_HT_nOrigs_Props = Dict([x=>λ_HT_Origs_Counts[x]/n for x in
keys(λ_HT_Origs_Counts)])
        λ_HT = map(x -> [count(==(i),length.(x)) for i in 1:5], Panels )
        λ_HT_nom = map(x -> [count(==(i),length.(x)) for i in 1:5], Panels_nom )
        λ_HT_Counts = countmap(λ_HT)
        λ_HT_Counts_nom = countmap(λ_HT_nom)
        λ_HT_Props = Dict([x=>λ_HT_Counts[x]/n for x in keys(λ_HT_Counts)])
        λ_HT_Props_nom = Dict([x=>λ_HT_Counts_nom[x]/n for x in keys(λ_HT_Counts_nom)])
        λ_HT_Thry = Dict(
                [K=> prod( [ 1//( factorial(K[j]) *  j^K[j] ) for j in 1:5])
                for K in keys(λ_HT_Counts) ]
        )
        λ_HT_Thry_Cond_Origs = Dict(
                [ K => λ_HT_nOrigs_Props[sum([i*K[i] for i in 1:5])]/n * prod( [ 1//(
factorial(K[j]) *  j^K[j] ) for j in 1:5])
                for K in keys(λ_HT_Props_nom) ]
        )
        eqχ_sq = length(keys(λ_HT_Thry))*sum(
                [ (λ_HT_Props[x]-λ_HT_Thry[x])^2 / λ_HT_Thry[x]
                for x in keys(λ_HT_Thry)]
```