VideoGames

November 21, 2025

1 Dleto on Toy data

This notebook explores a data set about toys, specifically video game data. This data is available from VG Chartz, all rights reserved.

First we need to load Julia and perhaps some necessary packages.

[ ]:
```
## Uncomment if you do not have iJulia installed
## it will take only one round to install, you can re-comment after that
# using Pkg
# Pkg.add("IJulia")
# This installs Julia's Jupyter kernel without Python dependencies
# println("IJulia installed! Restart VS Code and select Julia kernel.")

# Ensure Julia kernel is properly recognized
# This notebook requires Julia kernel for execution and export
using IJulia
println("Julia kernel is active!")
println("Julia version: ", VERSION) # Fix Jupyter/Julia setup - Install IJulia␣
 ↪for Julia notebooks

include("../Dleto.jl")
```

1.1 Load some toy data.

We are using a Comma Separated Value (CSV) file of toy data. We load this with Julia's DataFrames (Julia's version of R's data frame, similar to Python Pandas) and print a few values. This may require you to install a couple packages, uncomment the necessary commands if that happens. Once those are install you can remove those steps or comment them out again.

[ ]:
```
## Uncomment if you do not have these packages installed
## it will take only one round to install, you can re-comment after that
# import Pkg;
# Pkg.add("CSV")
# Pkg.add("DataFrames")

using CSV, DataFrames
```

```
# Load the CSV file
df = CSV.read("Video_Games.csv", DataFrame)
println("Loaded Video_Games.csv with ", nrow(df), " rows")

# Inspect the structure
println("Columns: ", names(df))
println(first(df, 5))
```

1.2 Creating a data tensor

This is a good place to demonstrate creating a tensor from a data set.
* We treat every row of the CSV/DataFrame as contributing to an entry in the tensor.
* That axes of the tensor are individual columns. We will demonstrate using “Platform”, “Genre”, “Critic Score” * For columns that are categories, the corresponding axis will have one basis vector for each category. For example, in Platform we have Wii and NES. These could be mapped to e_1 and e_2. Users familiar with one-hot encoding will recognize this encoding strategy only now we apply this just to individual axes. * For numeric columns we take actual values, or push them into a range of scores as the units. For example, if scores are out of 100 we might take units of 10 similar to A,B,C,D, F grading.

[104]:
```
# Get unique values for each dimension
platforms = unique(skipmissing(df.Platform))
genres = unique(skipmissing(df.Genre))
scores = unique(skipmissing(df.Critic_Score))


println("Unique Platforms: ", length(platforms))
println("Unique Genres: ", length(genres))
println("Unique User Scores: ", length(scores))
```

```
Unique Platforms: 31
Unique Genres: 12
Unique User Scores: 82
```

Now we build up the tensor with entires being total sales. Since each row of the data frame contributes to the data it is possible that several rows have the same platform, genre, and score. This will then be used to add the total sales. Our data is pretty course so we wont need full 64 bit floating points.

[105]:
```
# Initialize tensor
t = zeros(Float16, length(platforms), length(genres), length(scores))
# Use Critic_Score instead of Year for the third dimension
critic_scores = unique(skipmissing(df.Critic_Score))
# Filter out "tbd" or non-numeric scores
critic_scores = filter(x -> tryparse(Float64, string(x)) !== nothing,␣
 ↪critic_scores)
critic_scores = sort([parse(Float64, string(x)) for x in critic_scores])
```

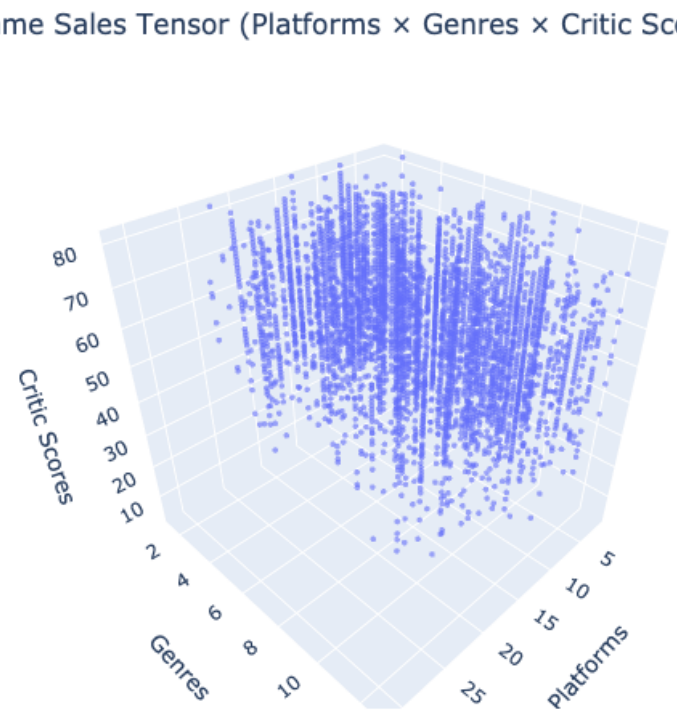
```
# Reinitialize tensor with new dimensions
t = zeros(Float64, length(platforms), length(genres), length(critic_scores))
# Fill tensor with aggregated sales data
for row in eachrow(df)
    if !ismissing(row.Platform) && !ismissing(row.Genre) && !ismissing(row.
 ↪Critic_Score)
        # Parse critic score
        score_str = string(row.Critic_Score)
        parsed_score = tryparse(Float64, score_str)

        if parsed_score !== nothing
            p_idx = findfirst(==(row.Platform), platforms)
            g_idx = findfirst(==(row.Genre), genres)
            s_idx = findfirst(==(parsed_score), critic_scores)

            if !isnothing(p_idx) && !isnothing(g_idx) && !isnothing(s_idx)
                # Aggregate global sales
                sales = get(row, :Global_Sales, 0.0)
                t[p_idx, g_idx, s_idx] += ismissing(sales) ? 0.0 : sales
            end
        end
    end
end

println("Created tensor with dimensions: ", size(t))
println("(", length(platforms), " platforms × ", length(genres), " genres × ",␣
 ↪length(critic_scores), " critic scores)")
# t = loadTensorFromFile("../lstm_hidden_states_tensor.txt")
# t = loadTensorFromFile("../gnn_adjacency_tensor.txt")
println("Tensor loaded with size: ", size(t))

# plotTensor(t)
```

```
Created tensor with dimensions: (31, 12, 82)
(31 platforms × 12 genres × 82 critic scores)
Tensor loaded with size: (31, 12, 82)
```

1.2.1 Visualization tools

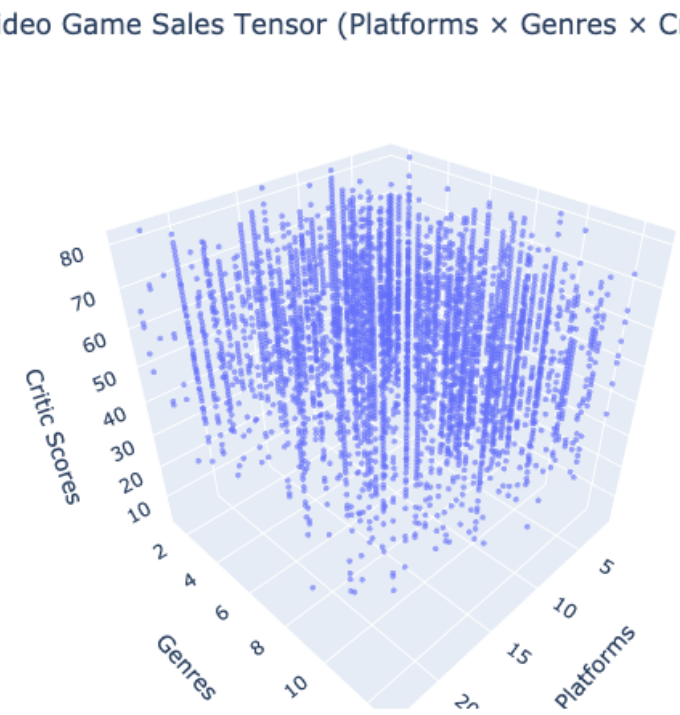
This is a good place to look at the tensor we created. You will need to have PlotlyJS active in your notebook, uncomment the command to install if you do not.

[106]:
```
# import Pkg; Pkg.add("PlotlyJS") # Uncomment if PlotlyJS is not installed
plotTensor(t; xlabel="Platforms", ylabel="Genres", zlabel="Critic Scores",
    title="Video Game Sales Tensor (Platforms × Genres × Critic Scores)")
```



We see here that not all the platforms are actually being used. There is a lot of missing data. There are tensor tools to detect such structure but it would be far more efficient to first go through some preliminary data processing to remove obvious issues. In this case having built the tensor we will simply drop the platforms beyond 23.

[107]:
```
# Safely slice the tensor based on actual dimensions
dim1, dim2, dim3 = size(t)
t_trimmed = t[1:min(23, dim1), :, :]
plotTensor(t_trimmed; xlabel="Platforms", ylabel="Genres", zlabel="Critic␣
 ↪Scores",
    title="Sliced Video Game Sales Tensor (Platforms × Genres × Critic Scores)")
```



When tensors get larger we will want some general statistics to guide our analysis. Often the data is wide ranging so we might drop small values or renormalize etc. To explore one option we will simply drop small sales volumes.

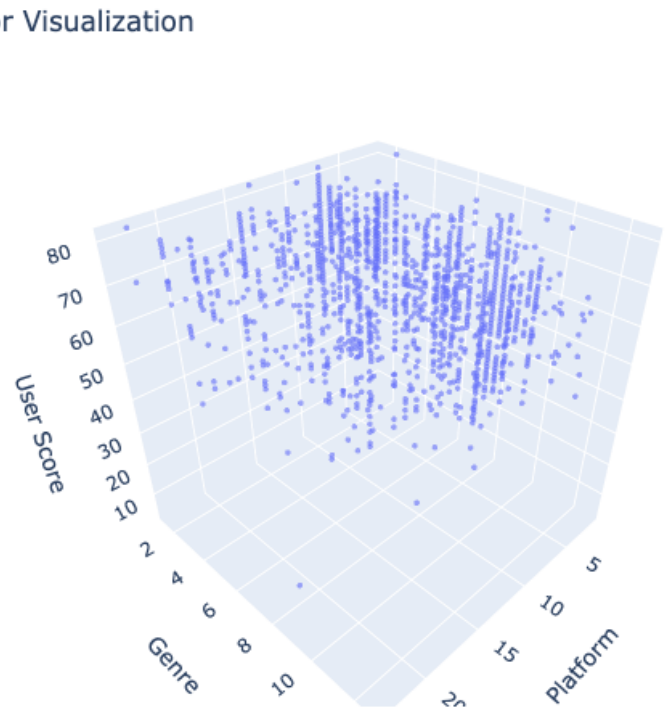
[108]:
```
dropSmall = x -> abs(x) < 1.0 ? 0 : x
t_trimmed_big = t_trimmed .|> dropSmall

# Check current tensor dimensions
println("Current tensor size: ", size(t_trimmed_big) )

nonzeros_count = count(!iszero, t_trimmed_big); nonzeros_count_orig = count(!
 ↪iszero, t)
total_length = length(t_trimmed_big); total_length_orig = length(t)
result = nonzeros_count / total_length; result_orig = nonzeros_count_orig /␣
 ↪total_length_orig
println("Number of nonzeros: ", nonzeros_count, " down from ",␣
 ↪nonzeros_count_orig)
println("Total length: ", total_length, " down from ", total_length_orig)
println("Ratio (nonzeros/length): ", result, " down from ", result_orig)
```

```
Current tensor size: (23, 12, 82)
Number of nonzeros: 1268 down from 4089
```

```
Total length: 22632 down from 30504
Ratio (nonzeros/length): 0.05602686461647225 down from 0.1340479937057435
```

For this size of data we can plot the resulting trimmed large value tensors. You might notice for top sellers the scores do not go below 30. We could trim this off as well.

[109]:
```
plotTensor(t_trimmed_big, 1.0; xlabel="Platform", ylabel="Genre", zlabel="User␣
↪Score")
```



1.3 Stratification

It is time to see what Dleto can chisel form this data. Depending on the size of the data and the strategy selected this could take some time. It is our present research question to improve on this timing. So for now treat this as a demonstration of what you will get for this investment in time.

We will start with the full tensor, which takes about 120 seconds on an Apple M2 and uses 2.3 GB.

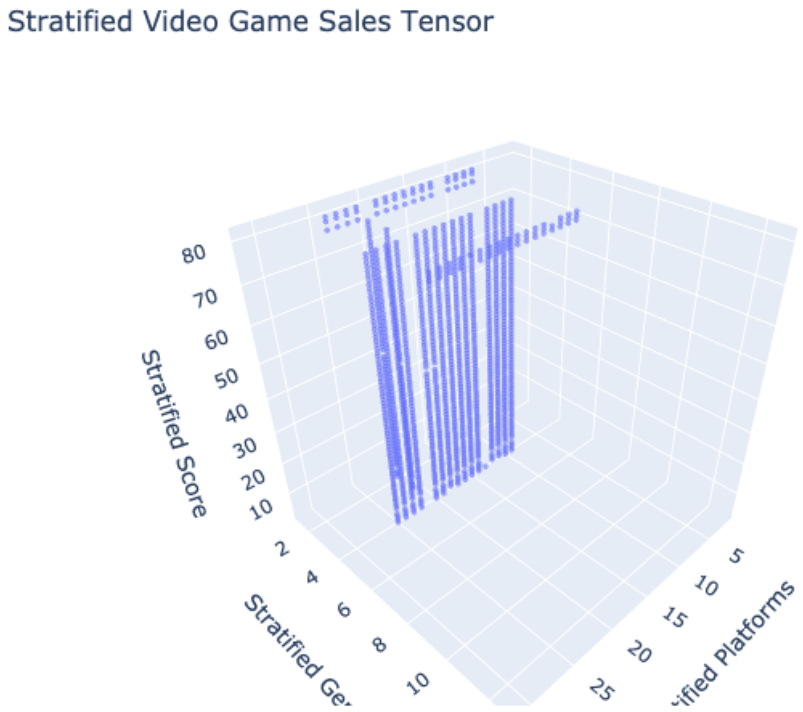
[110]:
```
@time u = stratify(t)
```

```
      Building linear system…
  0.608174 seconds (91.52 k allocations: 1.813 GiB, 21.47% gc time)

      Computing singular vectors for (7829, 30504)…
```

```
110.182942 seconds (48.63 k allocations: 2.291 GiB, 0.14% gc time)

        Extracting matrices…
  0.000048 seconds (2 allocations: 15.500 KiB)
  0.000002 seconds (2 allocations: 2.438 KiB)
  0.000035 seconds (4 allocations: 105.344 KiB)
110.793966 seconds (140.61 k allocations: 4.105 GiB, 0.26% gc time)
```

```
(tensor = [-5.306175732112984e-21 6.928803297596588e-27 … -4.819207764151554e-21␣
↪0.0; -1.0402272770455921e-20 -9.65677752310357e-27 … -9.740908075523617e-22 0.
↪0; … ; 1.5860213809999075e-22 7.553474130859457e-27 … -1.0093628943442131e-20␣
↪0.0; 1.2685315767245114e-30 7.9580234837497e-28 … 8.181778867458388e-30 0.0;;;␣
↪-1.3478158745195227e-20 5.310678902013431e-17 … -8.693146976509959e-20 0.0; -4.
↪5210131028547645e-20 -7.259638812425472e-17 … -2.9159644221987214e-19 0.0; … ;␣
↪1.2565578436026036e-20 5.740473250100093e-17 … 8.104550646248294e-20 0.0; 9.
↪677038695503155e-21 5.879453795032771e-18 … 6.241499395930247e-20 0.0;;; 1.
↪3478158746587176e-20 -5.310678902013278e-17 … 8.693146976370759e-20 0.0; 4.
↪521013102584685e-20 7.259638812426113e-17 … 2.9159644222257283e-19 0.0; … ; -1.
↪25655784354486e-20 -5.740473250100222e-17 … -8.104550646306025e-20 0.0; -9.
↪677038695503128e-21 -5.879453795033934e-18 … -6.241499395930245e-20 0.0;;; … ;;
↪; -8.122455354485157e-22 -4.9359222199025523e-17 … -5.238823754218083e-21 0.0;␣
↪-1.5196398852461182e-20 -1.1098809973156442e-16 … -9.801378008545819e-20 0.0;␣
↪… ; 3.900077375664087e-21 1.8660554495517642e-17 … 2.5154731060030074e-20 0.0;␣
↪2.3943307639163124e-21 -3.208153884269868e-30 … 1.5442961929650365e-20 0.0;;;␣
↪-1.4871777922151452e-21 -3.7895846069722986e-16 … -1.4916025594593605e-21 0.0;␣
↪-7.246882405752326e-21 -1.5910005646023203e-15 … -1.4836280683504918e-20 0.0;␣
↪… ; 1.388058925006287e-22 3.213480723966178e-16 … -6.310359985849182e-21 0.0;␣
↪8.663900334826052e-22 2.8290819049054544e-16 … 8.694515813119488e-22 0.0;;; -2.
↪7421509144704904e-33 -3.675476758182608e-16 … -1.7686333407109765e-32 0.0; -9.
↪185511232115402e-33 -1.5694146178364588e-15 … -5.924473861327039e-32 0.0; … ;␣
↪2.553321252215119e-33 3.180933102148958e-16 … 1.6468419270372572e-32 0.0; 1.
↪966984250263707e-33 2.801059666972029e-16 … 1.268666107073668e-32 0.0],␣
↪Xchange = [-9.972640722605748e-17 -1.865793278238257e-15 … 4.
↪788462202639284e-16 2.9397268975163856e-16; 0.0 0.0 … 0.0 0.0; … ; 0.0 0.0 … 0.
↪0 0.0; 0.0 0.0 … 0.0 0.0], Ychange = [0.0 1.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ;
↪ 0.00012161288456923982 0.0 … 0.0 0.0; 1.3203173628735626e-35 0.0 … 0.0 0.0],␣
↪Zchange = [0.0 0.0 … 0.0 0.0; 4.4782206390848134e-27 3.6559836831323824e-17 …␣
↪8.323957794480505e-29 -2.8155358516807254e-47; … ; 0.0 0.0 … 0.0 0.0; 0.0 0.0␣
↪… 0.0 0.0], Xes = [-0.04789693653995108, -0.020337987033814838, -0.
↪017120208703085473, -0.011276220492168556, -0.010425618604770797, -0.
↪008165295774271084, 0.0, 0.0, 5.863117436494866e-45, 5.868435327374634e-45  … ␣
↪5.918562485837084e-45, 5.941747676862598e-45, 5.99707622794497e-45, 6.
↪442608509182039e-45, 0.011928175470739145, 0.023460894476072428, 0.
↪025201779665159938, 0.04919912915615237, 0.06038878746649544, 0.
↪23682980240105175], Yes = [-5.986538442540361e-49, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0, 0.0, 7.441844403700714e-49, 1.402101247319834e-48, 3.86121776425297e-48, 1.
↪6079169818792883e-16], Zes = [-0.008344233348309367, -1.33902019439334e-14, -8.
↪126044521089872e-16, -7.408144515670094e-16, -6.938893903907228e-18, -6.
↪943437445754127e-29, -4.4672007548683845e-30, -1.3235482330397998e-42, -1.
↪4235372453615364e-44, -1.314075356658948e-44  …  1.2350592839501208e-44, 1.
↪345081796442528e-44, 1.591445233501582e-44, 2.995892227350657e-44, 1.
↪0868945794878018e-43, 3.791374638500385e-16, 0.010907883516232033, 0.
↪011928175470739145, 0.01670602055841929, 0.024774749956228647])
```

[111]:
```
plotTensor(u.tensor,0.01;
           xlabel="Stratified Platforms",
           ylabel="Stratified Genre",
           zlabel="Stratified Score",
           title="Stratified Video Game Sales Tensor")
```



You will see in this that essentially concentrated one Genre combination but spread along many platforms and scores. Some of these effects are the problem of using data with so many degeneracies. Lets recompute on the trimmed tensor to compare. This is smaller and the performance improves to about 90 seconds and 2 GB on an Apple M2.

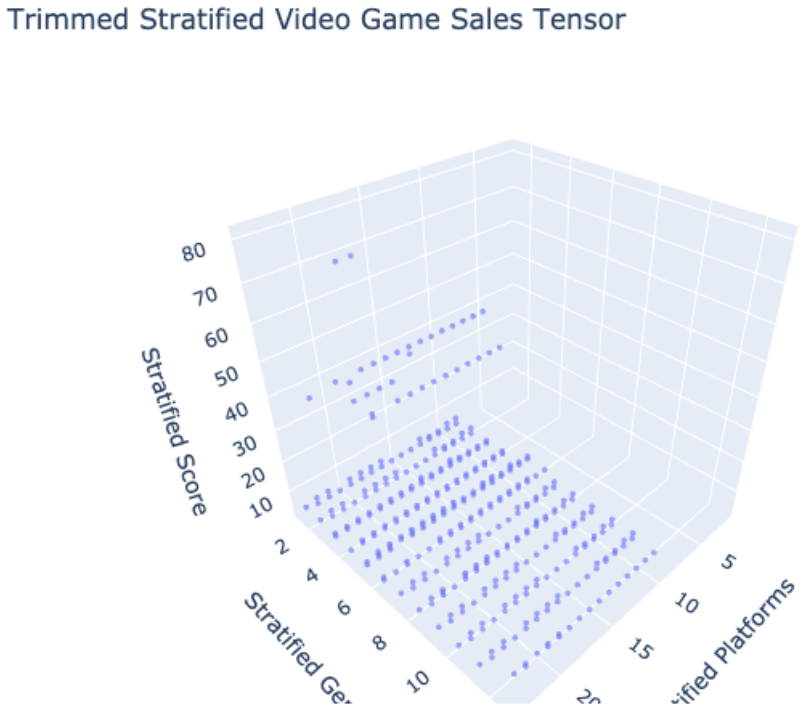
[112]:
```
@time v = stratify(t_trimmed)
```

```
      Building linear system…
 0.360429 seconds (67.90 k allocations: 1.271 GiB, 24.92% gc time)

      Computing singular vectors for (7397, 22632)…

92.096162 seconds (48.66 k allocations: 2.045 GiB, 0.32% gc time)

      Extracting matrices…
 0.000039 seconds (2 allocations: 8.750 KiB)
```

```
  0.000001 seconds (2 allocations: 2.438 KiB)
  0.000022 seconds (4 allocations: 105.344 KiB)
 92.460003 seconds (117.03 k allocations: 3.318 GiB, 0.42% gc time)

(tensor = [0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; -0.9203189482031027 -0.
 ↪9203189482031255 … 0.9203189482031249 0.0002616722141290434; -0.
 ↪0009569660528229152 -0.0009569660527667679 … 0.0009569660527790795 0.
 ↪00964858572696222;;; 0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; -1.
 ↪1757928017764524e-11 -1.1907994640826e-11 … 1.1872909223852289e-11 -0.
 ↪01192623676481066; -0.016568840342074787 -0.01656884034208043 … 0.
 ↪016568840342078766 -0.0008593012571089603;;; 0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0␣
 ↪0.0; … ; -0.09711357031929994 -0.0971135703192983 … 0.09711357031929457 1.
 ↪3508255459859816e-15; -0.0015321916833303812 -0.0015321916833323128 … 0.
 ↪0015321916833323799 1.280560820197343e-16;;; … ;;; 0.0 0.0 … 0.0 0.0; 0.0 0.0␣
 ↪… 0.0 0.0; … ; 7.453800690943017e-26 1.7441539877204274e-26 … -3.
 ↪1320887793788693e-26 7.950789723558117e-25; -1.9326884085394833e-15 3.
 ↪847361757187688e-26 … -2.789334858319408e-26 5.701580989347786e-27;;; 0.0 0.0␣
 ↪… 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; -4.863790190159324e-23 -4.
 ↪863790190159444e-23 … 4.863790190158987e-23 1.6752131364383503e-36; -1.
 ↪9326884055356e-15 3.212009937994826e-24 … -3.1656272949471846e-24 6.
 ↪496251642382062e-38;;; 0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; 0.0 0.0 … 0.0␣
 ↪0.0; -2.561682872298673e-15 6.422890801659495e-16 … 2.6792698797169533e-18 0.
 ↪0], Xchange = [0.0 0.0 … -0.23852473529359491 -0.017186025142143173; 1.0 0.0 …␣
 ↪0.0 0.0; … ; 0.0 0.0 … 0.0 0.007864554302861223; 0.0 0.0 … 0.0 0.
 ↪9681567176768177], Ychange = [0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0; … ; 2.
 ↪951815858138778e-13 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0], Zchange = [3.
 ↪660504157168336e-14 0.0 … 0.0 0.0; -8.921078596933535e-7 2.7579586967328114e-9␣
 ↪… 1.0 -1.0763689323359074e-6; … ; 0.0 0.0 … 0.0 0.0; 0.0 0.0 … 0.0 0.0], Xes =␣
 ↪[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07898300146667261, 0.07898300146668945, 0.
 ↪07898300146668988, 0.0789830014666905  …  0.07898300146669193, 0.
 ↪07898300146669204, 0.07898300146669217, 0.07898300146669222, 0.
 ↪07898300146669245, 0.07898300146669252, 0.07898300146669272, 0.
 ↪07898300146669293, 0.07898300146669351, 0.07898300146669582], Yes = [-3.
 ↪400058012914542e-16, -1.196959198423997e-16, 0.0, 0.0, 0.0, 0.0, 0.0, 1.
 ↪0625181290357943e-17, 1.5005358067199381e-16, 2.938187887435717e-16, 3.
 ↪7816971776294395e-16, 0.07898300146669252], Zes = [-0.1039731758972772, -0.
 ↪10397317589679088, -1.4383519012731855e-6, -5.806324388304973e-13, -4.
 ↪9113308092738756e-14, -3.099565959799322e-14, -1.553257509049298e-14, -1.
 ↪491160083730075e-14, -1.1015494072452725e-14, -9.258436031722717e-15  …  8.
 ↪963538123004343e-15, 1.1129396286935905e-14, 1.1599004905420846e-14, 1.
 ↪2125717097077882e-14, 2.936927333003613e-14, 3.0408753856972504e-14, 1.
 ↪0388420335007316e-13, 1.7688012377481063e-13, 6.697520278176397e-13, 1.
 ↪8977157451689834e-10])
```

We can plot the result and we see now there are 2 possibly 3 relevant score groups but not much separation in platforms.

[113]:
```
plotTensor(v.tensor;
           xlabel="Stratified Platforms",
           ylabel="Stratified Genre",
           zlabel="Stratified Score",
           title="Trimmed Stratified Video Game Sales Tensor")
```



We might now ask a number of questions: 1. Is this reliable features or if I run this again will it find other structure? 2. How do I see the actual combinations that lead to these clusters? 3. Are their other clustering targets?

We can start by looking a the data labels the new combinations. These are included in the output under cryptic labels (a future feature is to fix this). Use `u.Xchange` (Platforms groups), `u.Ychange` (Genre groups), and `u.Zchange` (Score groups) and replace `u` with `v` to look a the second stratified tensor.

If you print `v.Xchange` directly you get a matrix. Each column corresponds to the vector of combinations that corresponds to the new coordinates in the resulting tensor. So it may be more instructive to read this column-by-column.

[117]:
```
v.Xchange[:,1]
```

```
23-element Vector{Float64}:
 0.0
 1.0
```

```
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0

0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
```

This column is (e_2) which means this column did not change and it represents therefore 100% of the value of that original category. Since this is the platform category this is the 2nd platform. We can look this up.

[119]:
```
println("Platform at index 2: ", platforms[2])
```

```
Platform at index 2: NES
```

Perhaps unsurprisingly for Video Game experts, Nintendo NES stands apart from other platforms. But we should not jump to conclusions. Perhaps what we are seeing is that no platforms changed. Here inspecting the whole matrix can help us.

[121]:
```
v.Xchange
```

```
23×23 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0  0.0  …  -0.651445  -0.238525  -0.017186
 1.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.0
 0.0  1.0  0.0  0.0  0.0  0.0       0.0        0.0        0.0
 0.0  0.0  0.0  0.0  0.0  0.0       0.276399   0.971136   0.0153219
 0.0  0.0  0.0  0.0  0.0  0.0      -0.647963   0.0        0.0314286
 0.0  0.0  0.0  0.0  0.0  0.0  …    0.281723   0.0       -0.0654746
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.00923796
 0.0  0.0  1.0  0.0  0.0  0.0       0.0        0.0        0.0
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.0330348
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0       -0.00391016

 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.00779529
 0.0  0.0  0.0  0.0  1.0  0.0  …    0.0        0.0        0.0
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.0129722
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0        0.129937
 0.0  0.0  0.0  0.0  0.0  0.0       0.0        0.0       -0.0377176
```

```
 0.0  0.0  0.0  0.0  0.0  0.0      0.0        0.0        0.073544
 0.0  0.0  0.0  0.0  0.0  1.0  …   0.0        0.0        0.0
 0.0  0.0  0.0  0.0  0.0  0.0      0.0        0.0        0.00786455
 0.0  0.0  0.0  0.0  0.0  0.0      0.0        0.0        0.968157
```

We see now that while the first several columns are unchanged, the final 3 columns are in combination. Lets extract that combination.

[124]:
```
new_platform = [(platforms[i], v.Xchange[i,21]) for i in 1:size(v.Xchange,1)]
```

```
23-element Vector{Tuple{String7, Float64}}:
 ("Wii", -0.65144497056809)
 ("NES", 0.0)
 ("GB", 0.0)
 ("DS", 0.27639862696314355)
 ("X360", -0.647962563073312)
 ("PS3", 0.2817228535101357)
 ("PS2", 0.0)
 ("SNES", 0.0)
 ("GBA", 0.0)
 ("PS4", 0.0)

 ("PC", 0.0)
 ("2600", 0.0)
 ("PSP", 0.0)
 ("XOne", 0.0)
 ("WiiU", 0.0)
 ("GC", 0.0)
 ("GEN", 0.0)
 ("DC", 0.0)
 ("PSV", 0.0)
```

We see that platform group 23 is now a combination of "Wii" (down by 65%), "DS" (up 28%), "X360" (down 64%), "PS3" (up 28%). How to interpret this data will take a subject matter expert with experience in data science. However, we might conjecture that the algorithm is identifying a habit in sales which would be perceived as stable across this combination of groups. For example, that the sales Wii and XBox 360 sales negatively offset with "DS" and "PS2". We might look at the entire tensor slice here to learn more.

[125]:
```
v.tensor[21, :, :]
```

```
12×82 Matrix{Float64}:
  0.311924   -3.3562e-12   -0.0276398    …   8.31594e-15  -8.95102e-21
  0.311924   -3.3341e-12   -0.0276398       -1.3843e-23    0.0
  0.475084   -0.607006      0.0             -1.11854e-20   0.0
 -0.404618   -0.37527       0.0              0.0           0.0
 -2.01826    -0.7862       -0.0781498        1.35351e-12   0.0193479
  0.816651    0.0957858     0.0          …   0.0           0.0
 -0.311924    3.34531e-12   0.0276398        1.3843e-23    0.0
 -0.330004   -0.0219805     0.0275832       -3.66079e-23   0.0
```

```
 -0.311924    3.34475e-12   0.0276398        1.3408e-18   -1.44318e-24
 -0.311924    3.34507e-12   0.0276398        1.3843e-23    0.0
 -0.311924    3.3452e-12    0.0276398    …   1.3843e-23    0.0
 -0.0733776  -0.0325722     5.02517e-15      7.30764e-36   0.0
```

Since scores are the most pronounced clusters we should look at the Z-change (dropping small values to see better.)

[129]:
```
v.Zchange .|> dropSmall
```

```
82×82 Matrix{Real}:
 0  0  0   0     0    0     0     0    …   0    0     0    0    0    0    0
 0  0  0  -1.0  -1.0  1.0  -1.0  -1.0      -1.0  1.0  -1.0  1.0  1.0  1.0  0
 0  0  0   0     0    0     0     0         0    0     0    0    0    0    0
 0  0  0   0     0    0     0     0         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   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    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         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   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    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         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   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    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         0    0     0    0    0    0    0
```

We should see what row 2 is about.

[130]:
```
critic_scores[2]
```

```
17.0
```

1.4 Other chisels

We close with other chisels.

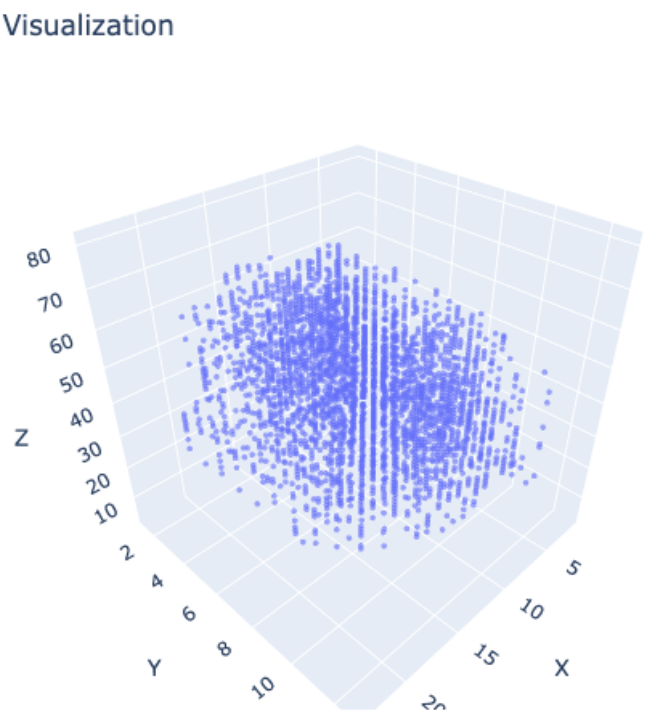
[131]:
```
@time s = toSurfaceTensor(t_trimmed)
```

```
        Building linear system…
  0.555177 seconds (124.96 k allocations: 677.101 MiB, 65.96% gc time, 9.52%
compilation time)

        Computing singular vectors for (3757, 22632)…
```

```
 23.100818 seconds (48.64 k allocations: 542.963 MiB, 0.10% gc time)

       Extracting matrices…
  0.014835 seconds (7.76 k allocations: 581.875 KiB, 99.68% compilation time)
  0.000006 seconds (2 allocations: 1.250 KiB)
  0.000015 seconds (3 allocations: 52.703 KiB)
 23.758618 seconds (448.99 k allocations: 1.210 GiB, 1.64% gc time, 0.63%
compilation time)
```

```
(tensor = [1.2775571871853449e-18 5.155402002490835e-18 … -5.
 ↪0969416206526955e-18 -2.246396309277731e-18; -4.0343313143763076e-17 1.
 ↪2875366159892006e-16 … -2.1655627644800206e-18 -4.581333016691393e-17; … ; -3.
 ↪061606721336904e-17 7.186409999153987e-17 … 8.311871266127921e-18 -1.
 ↪2133650360241247e-17; 6.575305295295524e-17 -2.0017088780764984e-17 … 3.
 ↪060236869201986e-17 4.4873030356578225e-17;;; 1.639480438753792e-17 -3.
 ↪350264694098166e-18 … -4.762273299843633e-18 3.521324502251401e-18; -5.
 ↪872936714836394e-17 9.827897115829746e-17 … 5.679666050251801e-17 3.
 ↪900096100886663e-17; … ; -6.987843368306937e-17 1.131742125238249e-16 … 1.
 ↪2638966416528276e-16 -3.478613641959115e-17; -4.417644958443909e-17 1.
 ↪3242607195832243e-17 … -3.374300755058791e-17 -8.423966342043285e-17;;; -1.
 ↪6782692351672664e-18 -5.475925389480474e-18 … 6.86020696550795e-18 -1.
 ↪7971103739165267e-19; -1.0431313442171118e-16 -8.444079243349558e-17 … 9.
 ↪518056017435879e-17 6.368449992800978e-17; … ; -6.209750221872327e-17 -1.
 ↪4292395417263234e-17 … 1.6895805677415487e-16 2.816636639314199e-17; 4.
 ↪457583736382168e-17 3.9991358072967055e-17 … 4.928630596901524e-17 -8.
 ↪928531602373522e-18;;; … ;;; -4.911896489662299e-18 1.5129914794360548e-17 …␣
 ↪-1.3684776000078835e-18 -4.689100123686225e-19; 1.5216045822637243e-16 1.
 ↪9188957762642435e-16 … -1.0505654862299012e-16 -8.785813467364504e-17; … ; 1.
 ↪2595701338673224e-16 -3.9274214610354566e-18 … -4.2161702809053934e-17 -4.
 ↪566926590658613e-17; -3.452204089765648e-17 -1.977857125120694e-16 … 4.
 ↪1909176402999566e-17 4.243747514818217e-17;;; 9.297471841223918e-19 5.
 ↪846063185491962e-18 … 5.3927252670519104e-18 1.6422553114980574e-18; -1.
 ↪9094578116739371e-16 2.0627705960219966e-16 … 4.500088022953551e-18 -4.
 ↪219704786746886e-17; … ; -1.954256043331478e-16 1.269888404501581e-16 … -4.
 ↪043616228287581e-17 -4.43085547569432e-17; 9.434067895971237e-17 -1.
 ↪1789834581602566e-16 … -3.777916585441685e-17 -8.119247999636491e-18;;; -7.
 ↪409803103643566e-18 -1.8270319179212185e-18 … 4.077519228909478e-18 -1.
 ↪1794255976611258e-20; -1.131404902916697e-16 -8.05914917572123e-17 … 5.
 ↪940837857305729e-17 4.467163345493606e-17; … ; -8.046124879811069e-17 -5.
 ↪2619333153769093e-17 … 1.1212755131283712e-17 3.896637829901692e-17; 7.
 ↪667899464406399e-17 -1.7285085732491494e-17 … -2.5859880234923178e-17 -4.
 ↪2355897264881645e-17], Xchange = [3.4392462857513516e-17 1.
 ↪7376495552374187e-16 … 3.223952477833042e-16 -9.815744527988927e-17; 1.
 ↪65274755142755e-5 -7.247872336222095e-6 … 1.2578670148486637e-5 1.
 ↪0548173982855457e-6; … ; 1.5543122344752192e-15 -2.498001805406602e-16 … 8.
 ↪743006318923108e-15 -4.718447854656915e-15; 1.2164756618029962e-15 2.
 ↪9196178471939656e-15 … -8.081250762079457e-15 7.371029841936065e-16], Ychange␣
 ↪= [-0.0012633779626063375 0.006519808442359781 … -0.35833541118204154 0.
 ↪0032564505797186127; 0.013470794293760496 0.777688045518198 … 0.
 ↪055391792719644575 0.0004966368251834142; … ; -0.9496971169254201 -0.
 ↪035668622821966964 … -0.0006433741688878294 0.21873815597522445; 0.
 ↪17453458125125132 0.04042166445804459 … -0.017717228521186287 0.
 ↪9608740552820954], Zchange = [-0.37887976743975965 0.4111430744370792 … -0.
 ↪32496060821553413 -0.4002432047997315; -0.2188792786791077 -0.
 ↪031153188464531557 … -0.04084326716868333 -0.16841654936536737; … ; 0.
 ↪0016831248404401214 0.0010942972818679397 … 0.0012594684160633085 -0.
 ↪001157831398151482; -0.0003099009609799421 0.004934908006995518 … 0.
 ↪0021009883542583057 -0.0005920132676607481], Xes = [-0.958387687718345, -0.
 ↪2476281234891276, -0.01176320374058093, -6.818988535165852e-10, 1.
 ↪1597699643786494e-9, 1.159769965451511e-9, 1.1597699718634471e-9, 1.
 ↪1597699741863084e-9, 1.159769975230977e-9, 1.1597699753496007e-9  …  1.
 ↪1597699870331364e-9, 1.1597700081179063e-9, 1.159770023576269e-9, 1.
```

[132]:
```
plotTensor(s.tensor, 1.0)
```



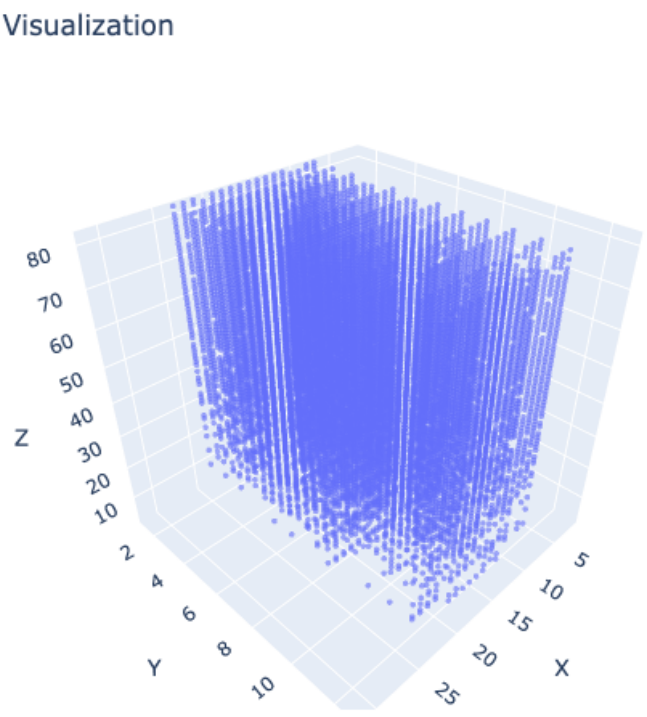
While surface chisels are great at recovering data like embedded surfaces that arise in PDEs, because they only use symmetric matrices and thus rotations, they are less capable of clustering in this setting.

[133]:
```
@time f = toFaceCurveTensor(t)
```

```
0.800605 seconds (337.43 k allocations: 178.064 MiB, 13.51% compilation time)
```

```
(tensor = [4.127357923217616e-19 -8.801944126500106e-19 … 3.933935585027017e-20
 8.54362245370243e-19; -6.579714908678993e-18 1.4031805351483372e-17 … -6.
 271366598224714e-19 -1.361999639443108e-17; … ; -6.011054759494205e-18 1.
 2819088898680641e-17 … -5.7293558401849e-19 -1.2442871049480771e-17; 4.
 520853916841984e-20 -9.64110802125223e-20 … 4.308990988673154e-21 9.
 358158355146673e-20;;; -1.7535649757181358e-17 -4.293105527883586e-18 … -1.
 6587158137189638e-17 1.529460337663417e-17; 2.7954826861914473e-16 6.
 843944957486663e-17 … 2.6442768889498214e-16 -2.438221538613721e-16; … ; 2.
 553879573680258e-16 6.25244839349371e-17 … 2.4157419279331005e-16 -2.
 2274951708094344e-16; -1.920747179283164e-18 -4.702403644706916e-19 … -1.
 8168552432043922e-18 1.6752767476947393e-18;;; 1.4490125525927546e-17 2.
 6049080764374815e-17 … 8.938151577358858e-18 1.242111476771801e-17; 3.
 2834457667110824e-16 5.69836924066561e-16 … 1.4610603007659975e-16 2.
 8021264198591536e-16; … ; 1.8031952703151018e-16 2.8496681894249035e-16 … 7.
 07450804593139e-17 1.5208849550038238e-16; -2.60555444975489e-17 -5.
 1386823752224144e-17 … -1.3531361355655874e-17 -2.226807217091813e-17;;; … ;;;
 -7.916435121816621e-16 1.660833665349279e-15 … -6.772762716410733e-16 -1.
 2399176196178983e-15; -1.2274947640329201e-14 2.6091081139511875e-14 … -1.
 1776941527074522e-14 -1.5368270285992926e-14; … ; -1.972766256220711e-14 3.
 635417427375443e-14 … -1.5193808941706794e-14 -2.218628347376001e-14; -2.
 385082195897226e-15 3.785319090622832e-15 … -1.4553505971784711e-15 -3.
 2064810488098375e-15;;; 3.985224891221581e-17 4.730541275290712e-16 … 1.
 349200751483904e-16 1.4348041807844716e-16; -2.6820794397177575e-15 1.
 435924603988759e-14 … 1.3579105041688075e-14 1.9663290404381317e-15; … ; 3.
 9068777065708174e-15 -5.0088530555319585e-15 … -2.4096560186498886e-14 -4.
 475191367562179e-15; -1.1670573116552092e-15 1.2907736085739095e-15 … 3.
 014930046085181e-15 2.0673081634774601e-16;;; -8.956766120353227e-16 1.
 0447124435988137e-15 … -1.9754726139126105e-16 -1.2810859470722 39e-15; -6.
 29396514790017e-14 7.938108917274549e-14 … -2.0490273541962838e-14 -9.
 1149779357867 57e-14; … ; 7.632724746332218e-14 -9.42608470715391e-14 … 2.
 5675495561076588e-14 1.11404161065333e-13; -1.0348578189061415e-14 1.
 2906220320630963e-14 … -3.252826069838377e-15 -1.497781950695077e-14], Xchange
 = [1.7884424961616162e-16 6.923350854725674e-16 … -1.1391314715608995e-15 -1.
 4535146219916475e-17; 0.0 0.0 … 0.0 0.0; … ; -9.266167450822094e-5 0.
 27546908172773205 … -0.09838219618203539 0.01901054740166566; -0.
 0020188284126388335 0.4114211245019512 … 0.18413025706502115 -0.
 00780488103667986], Ychange = [-0.009144690937836044 0.015868091913807288 … 0.
 03767432231162958 0.004510264438060285; 0.060327084230778635 0.
 06725509271727004 … -0.026135029539239747 0.047025895153832556; … ; -0.
 5890613672029326 -0.14421493624813725 … -0.5571994300526405 0.5137796489221664;
  -0.36971391071637566 0.4510478892579283 … 0.21812251394051452 0.
 2750836916652016], Zchange = [1.0 0.0 … 0.0 0.0; 0.0 1.0 … 0.0 0.0; … ; 0.0 0.
 0 … 1.0 0.0; 0.0 0.0 … 0.0 1.0], Xes = [-1.0474500260494097, -0.
 1202218575042644, -0.05024940808377587, -0.04650169926284131, -0.
 035034002736233694, -0.01529878350278746, -7.588133683383638e-6, -7.
 588133680186605e-6, -7.588133679403173e-6, -7.588133678624002e-6  …  -7.
 588133674056326e-6, -7.588133670909317e-6, 0.0, 0.0, 0.006392323931962984, 0.
 021386762657334413, 0.02651755567281855, 0.0746276541880686, 0.
 11664864431602728, 0.9187440830861642], Yes = [-7.588133679233131e-6, -7.
 588133678455144e-6, -7.588133678176591e-6, -7.588133678113651e-6, -7.
 588133678090425e-6, -7.588133678088927e-6, -7.588133678065136e-6, -7.
 588133678031028e-6, -7.588133677900009e-6, -7.588133677637644e-6, -7.
 588133677262273e-6, -7.588133676401365e-6], Zes = [1, 2, 3, 4, 5, 6, 7, 8, 9,
```

[134]:
```
plotTensor(f.tensor)
```



[ ]:
```
@time c = toCurveTensor(t)
plotTensor(c.tensor)
```