Zack Owens

Project 2

7/1/2023

1

1.1

Create list of names of images that men. Then create a list of names of images that are women.

In [2]:
```
cd("..\\img_align_celeba")
pwd()
```

```
"c:\\Users\\Firmz\\Desktop\\ParallelProgramClas\\ParallelProgrammingClass\\img_align_celeba"
```

In [3]:
```
# read in the CSV
using CSV
using DataFrames

df = CSV.read("./list_attr_celeba_2.csv",DataFrame)
```

202599×41 DataFrame

Row	id	5_o_Clock_Shadow	Arched_Eyebrows	Attractive	Bags_Under_Eyes	Bald	Bangs
	String15	Int64	Int64	Int64	Int64	Int64	Int64
1	000001.jpg	0	1	1	0	0	0
2	000002.jpg	0	0	0	1	0	0
3	000003.jpg	0	0	0	0	0	0
4	000004.jpg	0	0	1	0	0	0
5	000005.jpg	0	1	1	0	0	0
6	000006.jpg	0	1	1	0	0	0
7	000007.jpg	1	0	1	1	0	0
8	000008.jpg	1	1	0	1	0	0
9	000009.jpg	0	1	1	0	0	1
10	000010.jpg	0	0	1	0	0	0
11	000011.jpg	0	0	1	0	0	0
12	000012.jpg	0	0	1	1	0	0
13	000013.jpg	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
202588	202588.jpg	0	0	0	0	0	0
202589	202589.jpg	1	0	1	0	0	0
202590	202590.jpg	0	0	0	0	0	0
202591	202591.jpg	0	1	1	0	0	0
202592	202592.jpg	0	1	0	0	0	0
202593	202593.jpg	0	0	1	0	0	1
202594	202594.jpg	0	1	1	0	0	0
202595	202595.jpg	0	0	1	0	0	0
202596	202596.jpg	0	0	0	0	0	1
202597	202597.jpg	0	0	0	0	0	0
202598	202598.jpg	0	1	1	0	0	0
202599	202599.jpg	0	1	1	0	0	0

In [4]:
```
# split list into men list (hint use lpad)
males = filter(row->row.Male==1,df)
males.id
```

```
84434-element Vector{String15}:
 "000003.jpg"
 "000007.jpg"
 "000008.jpg"
 "000012.jpg"
 "000013.jpg"
 "000015.jpg"
 "000016.jpg"
 "000020.jpg"
 "000021.jpg"
 "000023.jpg"
 ⋮
 "202570.jpg"
 "202581.jpg"
 "202585.jpg"
 "202586.jpg"
 "202588.jpg"
 "202589.jpg"
 "202590.jpg"
 "202596.jpg"
 "202597.jpg"
```

In [5]:

```
# split list into women list (hint use lpad)
females = filter(row->row.Male==0,df)
females.id
```

```
118165-element Vector{String15}:
 "000001.jpg"
 "000002.jpg"
 "000004.jpg"
 "000005.jpg"
 "000006.jpg"
 "000009.jpg"
 "000010.jpg"
 "000011.jpg"
 "000014.jpg"
 "000017.jpg"
 ⋮
 "202584.jpg"
 "202587.jpg"
 "202591.jpg"
 "202592.jpg"
 "202593.jpg"
 "202594.jpg"
 "202595.jpg"
 "202598.jpg"
 "202599.jpg"
```

1.2

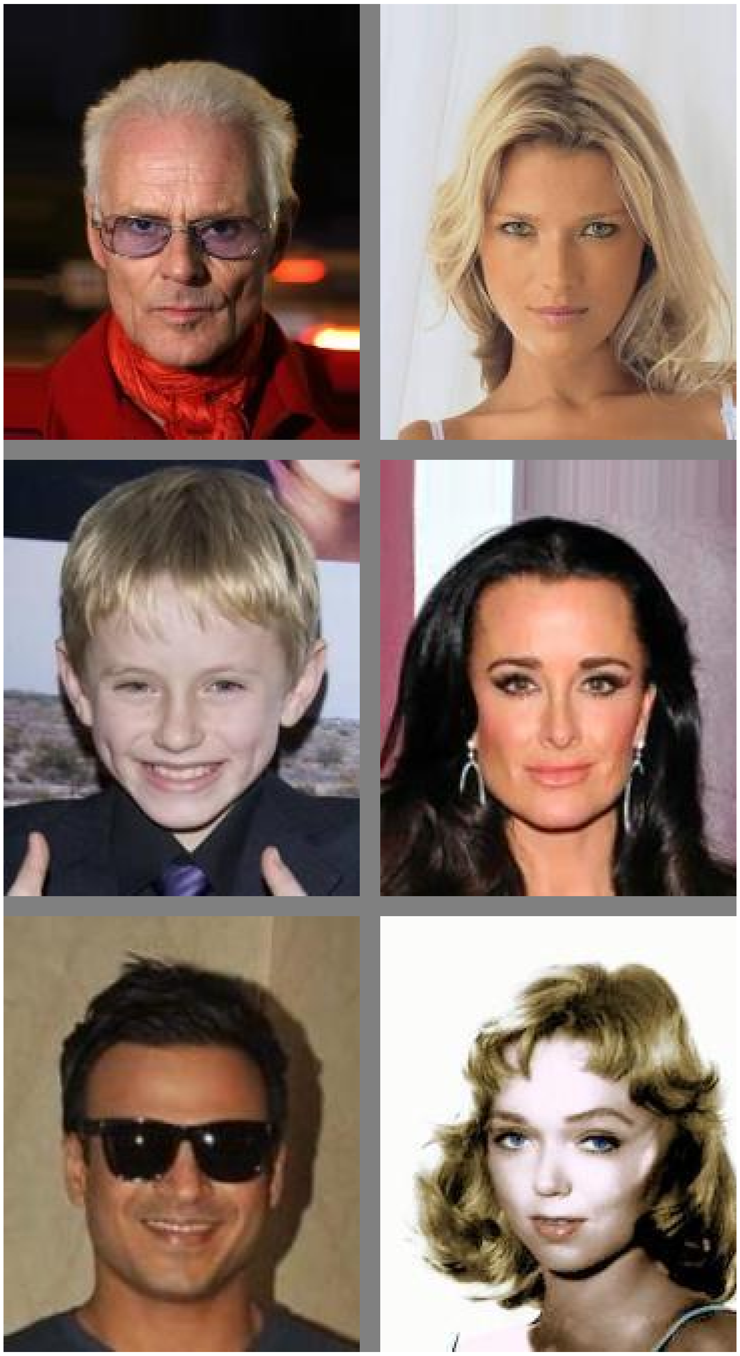
Display the last 3 faces of men and the last 3 faces of women.

In [9]:

```
#show last 3 mens and womens faces
using Images, FileIO, Colors

last_3_males = males.id[length(males.id)-2:length(males.id)]
last_3_females = females.id[length(females.id)-2:length(females.id)]
```

```julia
@time begin
    vector_N = Vector{Matrix{RGB{N0f8}}}([])
    for (i, f) in enumerate(last_3_males)
            push!(vector_N, load(f))
    end
end


@time begin
    vector_N_2 = Vector{Matrix{RGB{N0f8}}}([])
    for (i, f) in enumerate(last_3_females)
            push!(vector_N_2, load(f))
    end
end


mosaicview([vector_N ; vector_N_2];
fillvalue=0.5, npad=10, ncol=2)
```

```
  0.826752 seconds (1.13 M allocations: 76.142 MiB, 11.96% gc time, 84.01% compilatio
n time: 7% of which was recompilation)
  0.004178 seconds (254 allocations: 720.203 KiB)
```



2

Compute the means face for men and women seperately.

2.1 (5 points) Use the Hwloc.jl package to display the configuration of the computer you use for the project. Displayed info should include: number of CPU physical and virtual cores, RAM size, and cache info.

In [6]:
```julia
using Hwloc

topology()
```

```
Machine (9.45 GB)
    Package L#0 P#-1 (9.45 GB)
        NUMANode (9.45 GB)
        L3 (16.0 MB)
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#0 P#-1
                PU L#0 P#0
                PU L#1 P#1
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#1 P#-1
                PU L#2 P#2
                PU L#3 P#3
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#2 P#-1
                PU L#4 P#4
                PU L#5 P#5
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#3 P#-1
                PU L#6 P#6
                PU L#7 P#7
        L3 (16.0 MB)
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#4 P#-1
                PU L#8 P#8
                PU L#9 P#9
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#5 P#-1
                PU L#10 P#10
                PU L#11 P#11
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#6 P#-1
                PU L#12 P#12
                PU L#13 P#13
            L2 (512.0 kB) + L1 (32.0 kB) + Core L#7 P#-1
                PU L#14 P#14
                PU L#15 P#15
```

In [6]:
```julia
print(Sys.total_memory() / 2^20 /1000, " GB of RAM")
```

```
16.33305859375 GB of RAM
```

2.2 (65 points) Compute mean face of female celebA faces and male celebA faces, respectively. To do this assignment, follow these steps:

1. Get the vectors of male_faces and female_faces as described in Problem 1.
2. To best use the available resources of the compute you use, consider:

- to compute color or gray mean face,
- to decide the number, N, of celebA faces to be included for the mean face calculation (the large the N the better for your project merit).

1. Document the data structures used, how the face files are read, and how the mean faces are computed in your project notebook.
2. Measure the runtimes of files read and men faces computation.
3. Display mean faces.

In [7]:
```
function mean_RGB(args...)

    c = RGB{Float64}(0,0,0)

    for arg in args
       c += arg
    end

    return RGB{N0f8}(c/length(args))
end
```

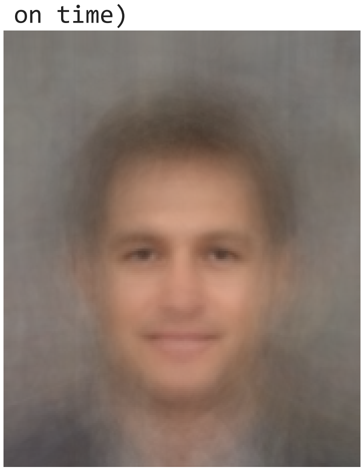
```
mean_RGB (generic function with 1 method)
```

In [27]:
```
# read in the images for men at once
cd("..\\img_align_celeba")
pwd()


@time begin
    vector_N = Vector{Matrix{RGB{N0f8}}}([])
    for (i, f) in enumerate(males.id[1:1_000])
            push!(vector_N, load(f))
    end
end
```

```
  1.623829 seconds (83.99 k allocations: 234.232 MiB, 7.13% gc time)
```

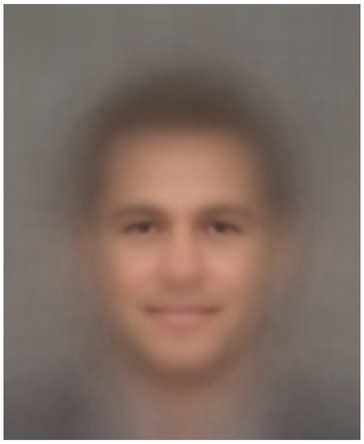
In [10]:
```
@time vector_mean = mean_RGB.(vector_N[1:100]...)
```

```
 21.479756 seconds (378.39 M allocations: 17.144 GiB, 8.29% gc time, 12.65% compilati
on time)
```



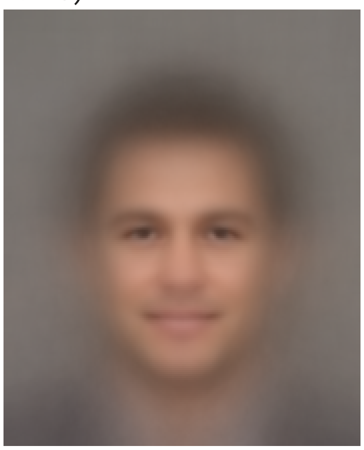
In [11]:
```
@time vector_mean = mean_RGB.(vector_N[1:500]...)
```

```
537.823437 seconds (10.04 G allocations: 460.272 GiB, 9.40% gc time, 8.78% compilatio
n time)
```



In [12]:
```
@time vector_mean = mean_RGB.(vector_N[1:1_000]...)
```

2088.332861 seconds (39.94 G allocations: 2.003 TiB, 9.19% gc time, 9.08% compilation time)

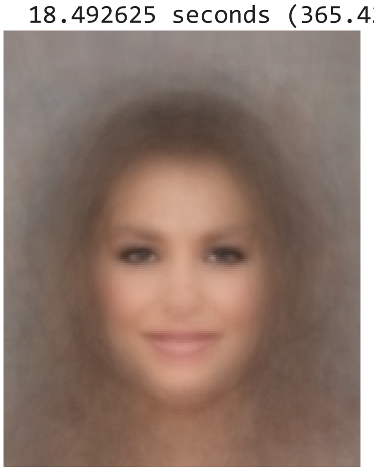


In [10]:
```
# read in the images for women at once
@time begin
    vector_N_2 = Vector{Matrix{RGB{N0f8}}}([])
    for (i, f) in enumerate(females.id[1:1_000])
            push!(vector_N_2, load(f))
    end
end
```

1.438743 seconds (83.99 k allocations: 234.713 MiB, 2.94% gc time)

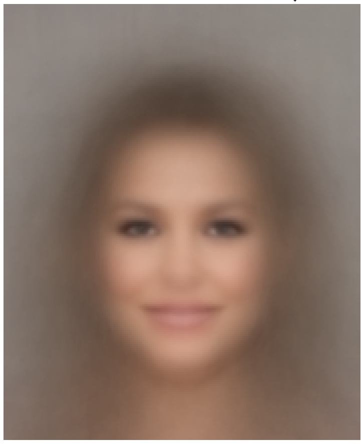
In [15]:
```
@time vector_mean = mean_RGB.(vector_N_2[1:100]...)
```

18.492625 seconds (365.42 M allocations: 16.484 GiB, 8.93% gc time)



In [16]:
```julia
@time vector_mean = mean_RGB.(vector_N_2[1:500]...)
```

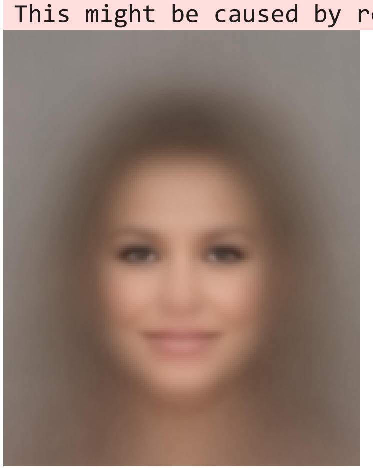
```
474.439248 seconds (9.77 G allocations: 448.683 GiB, 8.17% gc time)
```



In [11]:
```julia
@time vector_mean = mean_RGB.(vector_N_2[1:1_000]...)
```

```
2513.133556 seconds (40.17 G allocations: 2.012 TiB, 35.19% gc time, 0.02% compilatio
n time)
```

```
Internal error: stack overflow in type inference of materialize(Base.Broadcast.Broadc
asted{Base.Broadcast.DefaultArrayStyle{2}, Nothing, typeof(Main.mean_RGB), NTuple{100
0, Array{ColorTypes.RGB{FixedPointNumbers.Normed{UInt8, 8}}, 2}}}).
This might be caused by recursion over very long tuples or argument lists.
```



In [31]:
```julia
@time begin

    array_N = Array{RGB{N0f8}}(undef, 218, 178, 1_000)
    # requires (number_of_faces)x218x178x3 bytes of memory space

    for (i,f) in enumerate(males.id[1:1_000])
            array_N[:,:,i] = load(f)
    end
end
```

```
  1.789166 seconds (85.96 k allocations: 345.260 MiB)
```

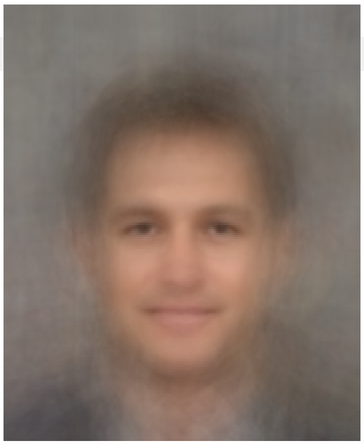
In [32]:
```julia
@time v = [array_N[:,:,i] for i in 1:1000]
```

```
  0.174579 seconds (30.84 k allocations: 112.873 MiB)
```

(a vector displayed as a row to save space)

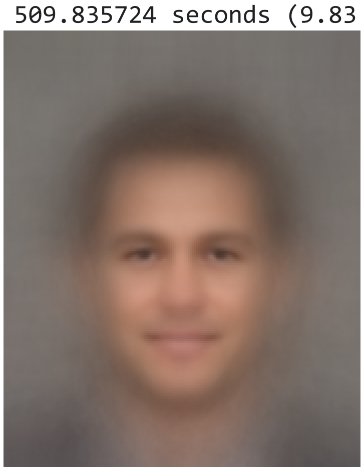
In [14]:
```
@time array_mean = mean_RGB.(v[1:100]...)
```

```
 19.857926 seconds (370.39 M allocations: 16.756 GiB, 7.52% gc time)
```



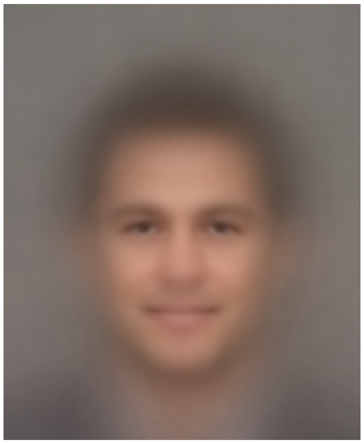
In [15]:
```
@time array_mean = mean_RGB.(v[1:500]...)
```

```
509.835724 seconds (9.83 G allocations: 451.449 GiB, 7.01% gc time)
```



In [16]:
```
@time array_mean = mean_RGB.(v[1:1_000]...)
```

```
1869.561592 seconds (39.03 G allocations: 1.966 TiB, 6.90% gc time)
```

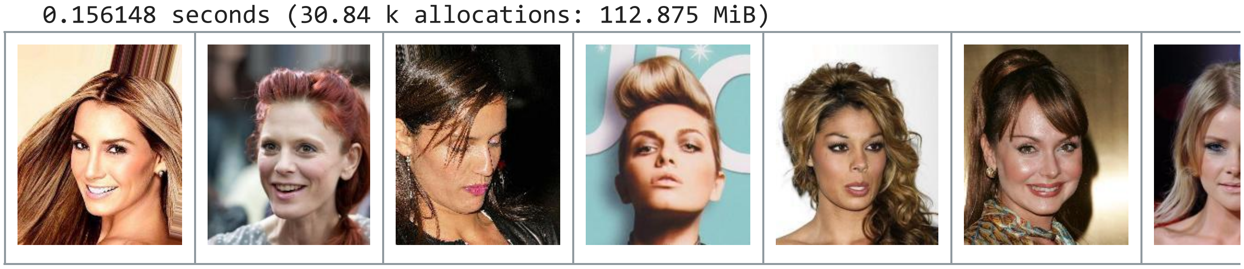


In [22]:
```
@time begin

    array_N = Array{RGB{N0f8}}(undef, 218, 178, 1_000)
    # requires (number_of_faces)x218x178x3 bytes of memory space

    for (i,f) in enumerate(females.id[1:1_000])
            array_N[:,:,i] = load(f)
    end
end
```

```
  1.479575 seconds (85.96 k allocations: 345.741 MiB, 3.03% gc time)
```

In [29]:
```
@time v = [array_N[:,:,i] for i in 1:1000]
```

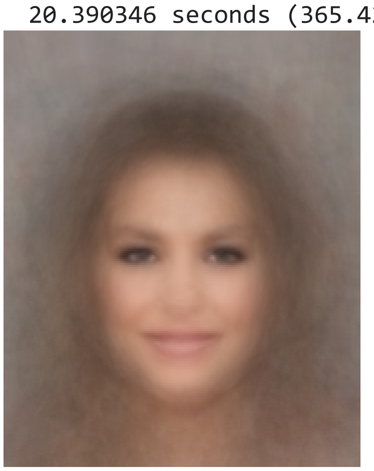
```
  0.156148 seconds (30.84 k allocations: 112.875 MiB)
```



(a vector displayed as a row to save space)

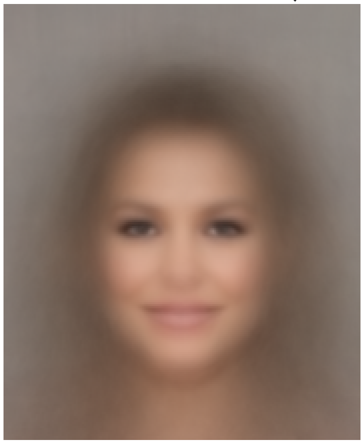
In [24]:
```
@time array_mean = mean_RGB.(v[1:100]...)
```

```
 20.390346 seconds (365.42 M allocations: 16.484 GiB, 8.93% gc time)
```



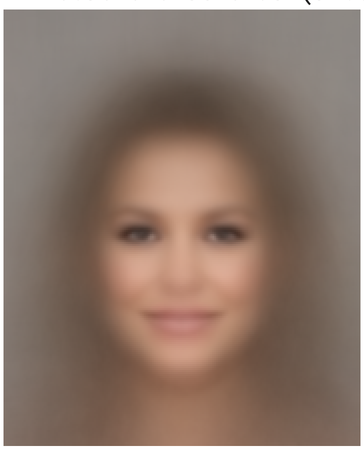
In [25]:
```
@time array_mean = mean_RGB.(v[1:500]...)
```

543.824524 seconds (9.77 G allocations: 448.683 GiB, 8.68% gc time)



In [26]:
```
@time array_mean = mean_RGB.(v[1:1_000]...)
```

1990.364610 seconds (39.03 G allocations: 1.966 TiB, 8.03% gc time)



2.2

Methods Discussion

For this project, I decided to use 1,000 images as the run times were under an hour while maintaining a colored mean face. This would mean faces were comprised of 1,000 images. This was due to using one function for computing the mean.

The following function was used to calculate the mean:

```
function mean_RGB(args...)

    c = RGB{Float64}(0,0,0)

    for arg in args
        c += arg
    end

    return RGB{N0f8}(c/length(args))
end
```

This function had many problems. the worst being that it needed to iteratively go pixel by pixel to calculate the mean face of each image. This would mean that each image would have its first pixel added to c before moving on to the next pixel. This is bad function design as this causes more misses on the CPU cache and even hits on the CPU cache are in the larger slower caches like L3. A better function would be to take an initial image then add each image pixels to the base image after being divided by the weight. Due to the time constraints, I was unable to create this function. This is one of the main reasons that the function used is so slow.

For this project I used two types of data structures. The two data structures were a 3 dimensional array and a vector of images. The files were read into memory using the similar linear loading method and used the same number of images (1,000).

```
@time begin
    vector_N_2 = Vector{Matrix{RGB{N0f8}}}([])
    for (i, f) in enumerate(females.id[1:1_000])
            push!(vector_N_2, load(f))
    end
end

@time begin

    array_N = Array{RGB{N0f8}}(undef, 218, 178, 1_000)
    # requires (number_of_faces)x218x178x3 bytes of memory space

    for (i,f) in enumerate(females.id[1:1_000])
            array_N[:,:,i] = load(f)
    end
end

@time v = [array_N[:,:,i] for i in 1:1000]
```

Results

Vector Mean Face Run Time

Size N	Male/Female	RunTime
100	Males	21.479756 seconds
500	Males	537.823437 seconds
1000	Males	2088.332861 seconds
100	Females	18.492625 seconds
500	Females	474.439248 seconds
1000	Females	2513.133556 seconds

Array Mean Run Time

Size N	Male/Female	RunTime
100	Males	19.857926 seconds

Size N	Male/Female	RunTime
500	Males	509.835724 seconds
1000	Males	1869.561592 seconds
100	Females	20.390346 seconds
500	Females	543.824524 seconds
1000	Females	1990.364610 seconds

Load Into Memory Times

DataType	Males/Females	Load Time
Vector	Males	1.438743 seconds
Vector	Males	1.623829 seconds
Array	Males	1.963745 seconds
Array	Females	1.635723 seconds

Discusssion

Overall the arrays took slightly longer than loading in the 1000 images. The run time of the arrays is overall better and would most likely be used instead of the vectors for my hardware. Comparing the 1000 of females run time the Array performed 500 seconds better than the vector. This is a considerable speedup that should be considered when scaling this up. The load times of the arrays had a slight increase in time as there was a required loading function to convert the array matrix into v. This on average added less than a quarter of a second but this time made the load times slower than the vectors.