HOMEWORK 2 SUBMISSION

Jinen Setpal

September 29, 2025

Checklist

1. Cross-checked independent work with Kunal Kapur, brief discussion of approach with Diego Aguilar.
2. No use of AI tools.
3. Code is included!

Problem 1

1. Here's the approach:
   a. Copy HW1 Q7 to get Poisson's Equation as a linear system w/o boundary conditions for $n = 10$.
   b. Flip the signs and consider the equivalent problem: $\boldsymbol{A}\mathbf{x} = \mathbf{b} \iff -\boldsymbol{A}\mathbf{x} = -\mathbf{b}$
   c. Check if new $\boldsymbol{A}' := -\boldsymbol{A}$ is PD by diagonal dominance.
   d. Solve $\boldsymbol{A}'$ with Richardson's Method ($\hat{\mathbf{x}}$) setting $\alpha = 1$, and with Julia's solver ($\mathbf{x}^*$) as the designated oracle. Then compare the difference in solutions (relative error): $\|\hat{\mathbf{x}} - \mathbf{x}^*\|_2 / \|\mathbf{x}^*\|$

Code

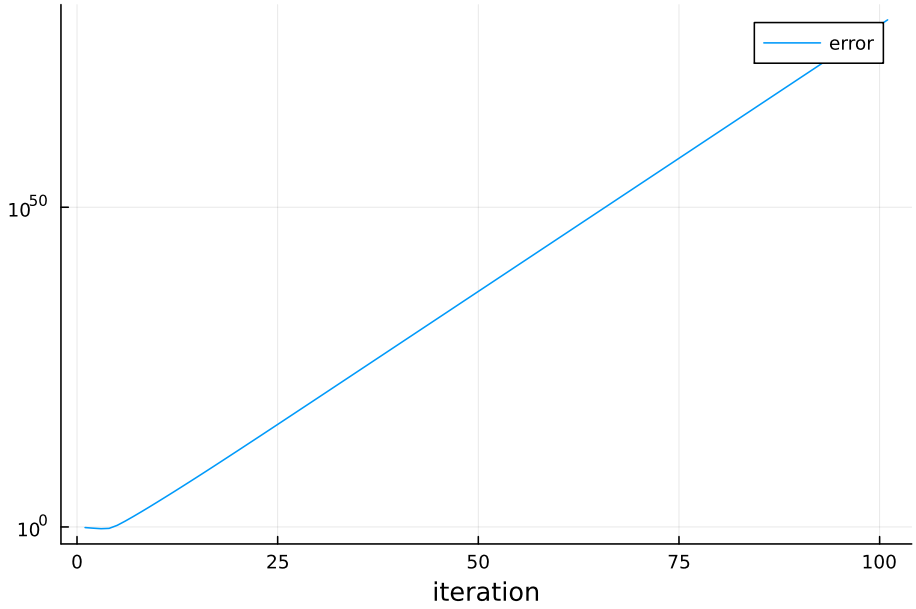
```
using LinearAlgebra, SparseArrays, Plots

function map_index(i::Integer, j::Integer, n::Integer)
        if 1 < i < n+1 && 1 < j < n+1
                return 4n + (i - 2)*(n-1) + j-1
        elseif i == 1
                return j
        elseif i == n+1
                return n + 1 + j
        elseif j == 1
                return 2(n+1) + i - 1
        elseif j == n+1
                return 2(n+1) + n - 2 + i
        end
end

function laplacian(n::Integer, f::Function)
        A = sparse(1I, (n+1)^2, (n+1)^2)
        A[diagind(A)[4n+1:end]] .= -4

        fvec = zeros((n+1)^2)

        global row_index = 4n + 1
        for i in 2:n
                for j in 2:n
                        A[row_index, map_index(i-1, j, n)] = 1
                        A[row_index, map_index(i+1, j, n)] = 1
                        A[row_index, map_index(i, j-1, n)] = 1
                        A[row_index, map_index(i, j+1, n)] = 1
                        fvec[row_index] = f(i, j)

                        global row_index += 1
```

```
                end
        end

        return A, fvec/n^2
end
n = 10
A, fv = laplacian(n, (x, y) -> 1)

A = A[4n+1:end,4n+1:end]
fv = fv[4n+1:end]

# equivalent system of equations
A = -A
fv = -fv

# check if new A is PD
is_pd = all((2 .* Array(diag(A)) - sum(A, dims=2)) .>= 0)  # diagonally
   dominant implies PD
println("Is `A` definitely PD? ", is_pd)

alpha = 1
T = 100
oracle_sol = A\fv
norm_oracle_sol = norm(oracle_sol)

function richardson(A::SparseMatrixCSC, fv::Vector, T::Integer, alpha::
   Number, oracle_sol::Vector)
        norm_oracle_sol = norm(oracle_sol)
        x_hat = 1 * fv  # copy-by-value, not reference (https://
           stackoverflow.com/a/76109083/10671309)
        err = [norm(x_hat - oracle_sol) / norm_oracle_sol]   # initial
           error
        for itr in 1:T
                x_hat = (I - (alpha * A)) * x_hat + (alpha * fv)
                push!(err, norm(x_hat - oracle_sol) / norm_oracle_sol)
        end
        return (x_hat, err)
end

function richardson(A::SparseMatrixCSC, fv::Vector, T::Integer, alpha::
   Number)
        x_hat = 1 * fv  # copy-by-value, not reference (https://
           stackoverflow.com/a/76109083/10671309)
        res = [norm(A * x_hat - fv) / norm(fv)]
        for itr in 1:T
                x_hat = (I - (alpha * A)) * x_hat + (alpha * fv)
                push!(res, norm(A * x_hat - fv) / norm(fv))
        end
        return (x_hat, res, minimum(push!(findall(x->x < 1e-5, res), T+1)
           ))
end

x_hat, err = richardson(A, fv, T, alpha, oracle_sol)
println(norm(x_hat - A\fv))
```

Output

```
Is `A` definitely PD? true
8.294254291121953184390746064420433471290093240340010870951240578726456583749248
    e+78
```

Richardson's Method produces a solution that diverges dramatically as we iterate, with a final error reading $8.29 \cdot 10^{78}$. Even though y is $\log_{10}$-scale it increases sharply:



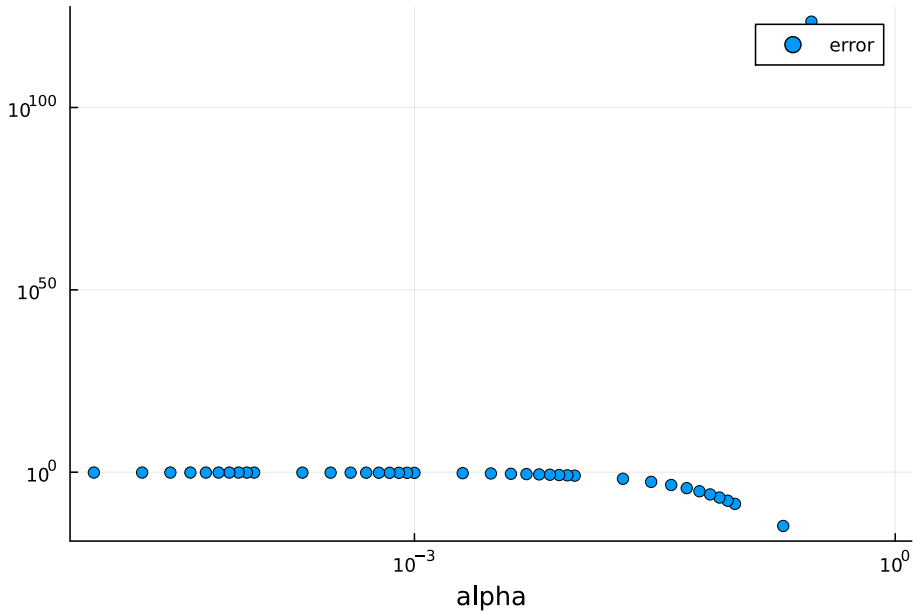
Setting $T = 1000$ we get a loss of `Inf` and at this point we can confidently claim that this method will not converge even as $T \to \infty$.

2. We can try a simple grid search with $\alpha := c \cdot 10^p \quad \forall c \in \{1, 2, \ldots, 9\}, p \in \{-1, -2, \ldots, -5\}$:

Code

```
alphs = []
errs = []
ress = []
tols = []
for alph_pow in 0:2
        for alph_coeff in 9:-.1:1
                local alpha = alph_coeff * 10^(-1. *alph_pow)
                local (x_hat, res, min_tol_iter) = richardson(A, fv, T,
                    alpha)
                local err = norm(x_hat - oracle_sol) / norm_oracle_sol

                push!(alphs, alpha)
                push!(errs, err)
                push!(ress, res)
                push!(tols, min_tol_iter)
        end
end
```

Output Here's the relative error progression:



Picking the largest α with relative error < 1, we get $\alpha = 0.2$ as a feasible choice with relative error 0.014 and relative residual 0.013 for $T = 100$.

3. Good news, with $n = 20$, we still have $\alpha = 0.2$ as a feasible solution. Bad news, the error is higher: relative error is 0.352 and relative residual is 0.3. I suspected I'm being too stingy with iterations so I set $T = 1000$ and now it is a more respectable relative error $= 4.75 \cdot 10^{-5}$ and relative residual $= 4.06 \cdot 10^{-5}$.

4. Very informally, from the figure in part 2, error w.r.t. α seems convex, and our previous feasible $\alpha = 0.2$ seems close to the minima, so I'll just do a more focused search around that:

Code

```
T = 1000

alphs = []
errs = []
ress = []
tols = []
for alph_pow in 0:2
        for alph_coeff in 9:-.1:1
                local alpha = alph_coeff * 10^(-1. *alph_pow)
                local (x_hat, res, min_tol_iter) = richardson(A, fv, T,
                   alpha)
                local err = norm(x_hat - oracle_sol) / norm_oracle_sol

                push!(alphs, alpha)
                push!(errs, err)
                push!(ress, res)
                push!(tols, min_tol_iter)
        end
end

tol = 1e-5
largest_alpha = maximum(alphs[errs .< tol])
println("largest feasible alpha: ", largest_alpha)
println("relative error: ", errs[alphs .== largest_alpha][end])
println("relative residual: ", ress[alphs .== largest_alpha][end][end])
```

```
println("fastest alpha: ", alphs[findall(x-> x == minimum(tols), tols)][
    end])
```

Output

a. With $n = 10$:

```
largest feasible alpha: 0.25
relative error:
    1.513596834021183337855890852231162135341721304455506905165879707855506594343
    e-16
relative residual: 9.717969227135053e-16
fastest alpha: 0.25
```

b. With $n = 20$:

```
largest feasible alpha: 0.25
relative error:
    3.943854431512536591214718270280764027830931906794095621191708359298462677555
    e-06
relative residual: 3.3672320687686646e-6
fastest alpha: 0.25
```

In either case, $\alpha = 0.25$ converges the quickest.

5. Reading $\alpha < 0.25$ is encouraging. We know that our method will converge for $\alpha < 2/\rho(\boldsymbol{A})$. T.P.T.

$$\alpha < 0.25 \implies \alpha < \frac{2}{\rho(\boldsymbol{A})}$$

Or, equivalently:

$$\rho(\boldsymbol{A}) \leq 8 \quad \text{where} \quad \rho(\boldsymbol{A}) = \max\{|\lambda| : \boldsymbol{A}\mathbf{x} = \lambda\mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^n \setminus \mathbf{0}_n\}$$

We can massage the eigenvector definition a bit:

$$\begin{aligned} \boldsymbol{A}\mathbf{x} = \lambda\mathbf{x} &\implies \|\boldsymbol{A}\mathbf{x}\| = \|\lambda\mathbf{x}\| \\ &\implies \|\boldsymbol{A}\mathbf{x}\| = |\lambda|\|\mathbf{x}\| \\ &\implies |\lambda| = \frac{\|\boldsymbol{A}\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\|\boldsymbol{A}\|\|\mathbf{x}\|}{\|\mathbf{x}\|} = \|\boldsymbol{A}\| \end{aligned}$$

Importantly, we need our choice of matrix norm to be submultiplicative for the last step. $\therefore |\lambda| < \|\boldsymbol{A}\| \ \forall\lambda$, which further implies $\rho(\boldsymbol{A}) < \|\boldsymbol{A}\|$. We choose $\|\cdot\|_\infty$ as the norm; in addition to being submultiplicative, this is very important because we know the structure of $\boldsymbol{A}$: each row is sparse and has one 4 and *at-most four* -1's. Picking the largest possible absolute case allows us to calculate $\|A\|_\infty = \max_{i=1}^{m} \sum_{j=1}^{n} |\boldsymbol{A}_{ij}| = 8$.

$$\therefore \rho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|_\infty = 8$$

Which satisfies the desired property, proving that $\alpha < 0.25$ will converge for any $n \in \mathbb{N}$.

Problem 2

1. We have:

$$f(\boldsymbol{A}) = \max_{i,j} |\boldsymbol{A}_{i,j}| = \max_{i} |vec(\boldsymbol{A})_i| = \|vec(\boldsymbol{A})\|_\infty$$

f is the well-known max norm, with just an extra 'flatten' step added to the input.

2. T.P.T:

$$\exists \boldsymbol{A} \in \mathbb{R}^{m\times n},\ \mathbf{x} \in \mathbb{R}^n : f(\boldsymbol{A}\mathbf{x}) > f(\boldsymbol{A})f(\mathbf{x})$$

Proof by example:

$$\text{Let} \quad \boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$$

Then, we have:

$$f(\boldsymbol{A}\mathbf{x}) = f\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}^T\right) = f\left(\begin{bmatrix} 2 & 2 \end{bmatrix}^T\right) = \max\{2,2\} = 2$$

$$f(\boldsymbol{A})f(\mathbf{x}) = f\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right) f\left(\begin{bmatrix} 1 & 1 \end{bmatrix}^T\right) = \max\{1,1,1,1\} \cdot \max\{1,1\} = 1 \cdot 1 = 1$$

$$f(\boldsymbol{A})f(\mathbf{x}) = 1 < 2 = f(\boldsymbol{A}\mathbf{x}) \implies f(\boldsymbol{A})f(\mathbf{x}) < f(\boldsymbol{A}\mathbf{x})$$

3. Simplifying σ, T.P.T.

$$\exists \sigma > 0 : f(\boldsymbol{A}\mathbf{x}) \le \sigma f(\boldsymbol{A})f(\mathbf{x}) \qquad \forall \boldsymbol{A} \in \mathbb{R}^{m\times n},\ \mathbf{x} \in \mathbb{R}^m,\ m,n \in \mathbb{N}$$

We have:

$$\begin{aligned} f(\boldsymbol{A}\mathbf{x}) &= \max_i \left| \sum_{j=1}^{n} \boldsymbol{A}_{i,j}\mathbf{x}_j \right| \\ &\le \max_i \left| \sum_{j=1}^{n} |\boldsymbol{A}_{i,j}||\mathbf{x}_j| \right| \end{aligned}$$

All summands are positvie, so we can drop the outer absolute:

$$\begin{aligned} &= \max_i \sum_{j=1}^{n} |\boldsymbol{A}_{i,j}||\mathbf{x}_j| \\ &\le \max_i \sum_{j=1}^{n} \max_{k,l} |\boldsymbol{A}_{k,l}| \max_k |\mathbf{x}_k| \\ &= \max_i \sum_{j=1}^{n} f(\boldsymbol{A})f(\mathbf{x}) \end{aligned}$$

i has no role to play in this equation, we can drop the max:

$$\begin{aligned} &= \sum_{j=1}^{n} f(\boldsymbol{A})f(\mathbf{x}) \\ &= n \cdot f(\boldsymbol{A})f(\mathbf{x}) \\ f(\boldsymbol{A}\mathbf{x}) &\le n \cdot f(\boldsymbol{A})f(\mathbf{x}) \end{aligned}$$

Since $n > 0$, we can set $\sigma := n$ which gives us the desired sub-multiplicativity.

Problem 3

1. T.P.T:

$$\forall \varepsilon > 0,\ \alpha_p \in (0,1) \quad \exists \alpha_r \in (0,1) : \limsup_{k\to\infty} \|(\boldsymbol{I} - \alpha_p \boldsymbol{P})\mathbf{x}^{(k)} - (1-\alpha_p)\mathbf{v}\| < \varepsilon$$
$$\text{where} \quad \mathbf{x}^{(k)} := (\boldsymbol{I} - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P}))\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v}$$

Let's first inspect the evolution of the residual:

$$\begin{aligned}
(\boldsymbol{I} - \alpha_p \boldsymbol{P})\mathbf{x}^{(k)} - (1-\alpha_p)\mathbf{v} &= (\boldsymbol{I} - \alpha_p \boldsymbol{P})(\boldsymbol{I} - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P}))\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} - (1-\alpha_p)\mathbf{v} \\
&= \left(\boldsymbol{I} - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P}) - \alpha_p \boldsymbol{P} + \alpha_p^2 \alpha_r \boldsymbol{P}^2\right)\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} - (1-\alpha_p)\mathbf{v} \\
&= (\boldsymbol{I} - \alpha_p \boldsymbol{P})\,\mathbf{x}^{(k-1)} - (1-\alpha_p)\mathbf{v} + \left(\alpha_p^2 \alpha_r \boldsymbol{P}^2 - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P})\right)\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
&= (\boldsymbol{I} - \alpha_p \boldsymbol{P})\,\mathbf{x}^{(k-1)} - (1-\alpha_p)\mathbf{v} + \left(\alpha_p^2 \alpha_r \boldsymbol{P}^2 - \alpha_r \alpha_p \boldsymbol{P} + \alpha_r \boldsymbol{I}\right)\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
&= (\boldsymbol{I} - \alpha_p \boldsymbol{P})\,\mathbf{x}^{(k-1)} - (1-\alpha_p)\mathbf{v} + \alpha_r\left(\left(\alpha_p^2 \boldsymbol{P}^2 - \alpha_p \boldsymbol{P} + \boldsymbol{I}\right)\mathbf{x}^{(k-1)} + (1-\alpha_p)\mathbf{v}\right) \\
&= (\boldsymbol{I} - \alpha_p \boldsymbol{P})\,\mathbf{x}^{(k-1)} - (1-\alpha_p)\mathbf{v} + \alpha_r \alpha_p^2 \boldsymbol{P}^2 \mathbf{x}^{(k-1)} - \alpha_r\left((\boldsymbol{I} - \alpha_p \boldsymbol{P})\,\mathbf{x}^{(k-1)} - (1-\alpha_p)\mathbf{v}\right)
\end{aligned}$$

Let $\mathbf{r}^{(k)}$ be the residual at step k. Then, we have:

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + \alpha_r \alpha_p^2 \boldsymbol{P}^2 \mathbf{x}^{(k-1)} - \alpha_r \mathbf{r}^{(k-1)} = (1-\alpha_r)\mathbf{r}^{(k-1)} + \alpha_r \alpha_p^2 \boldsymbol{P}^2 \mathbf{x}^{(k-1)} \tag{1}$$

Next, note that each $\mathbf{x}^{(k)}$ on unravel reveals factor $\alpha_r, (1-\alpha_r) \in (0,1)$ quantifying reduction in magnitude:

$$\begin{aligned}
\mathbf{x}^{(k)} &= (\boldsymbol{I} - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P}))\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
&= ((1-\alpha_r)\boldsymbol{I} + \alpha_r \alpha_p \boldsymbol{P}))\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
&= (1-\alpha_r)\mathbf{x}^{(k-1)} + \alpha_r \alpha_p \boldsymbol{P}\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
\mathbf{x}^{(k)} &= (1-\alpha_r)\mathbf{x}^{(k-1)} + \alpha_r\left(\alpha_p \boldsymbol{P}\mathbf{x}^{(k-1)} + (1-\alpha_p)\mathbf{v}\right)
\end{aligned}$$

Thus the addition of each step $\mathbf{x}^{(k)}$ adds such a factor. We induct this into Eq. (1) as some $\gamma < 1$ in the limit:

$$\begin{aligned}
\lim_{k\to\infty} \mathbf{r}^{(k)} &= \lim_{k\to\infty} (1-\alpha_r)\mathbf{r}^{(k-1)} + \alpha_r \alpha_p^2 \boldsymbol{P}^2 \mathbf{x}^{(k-1)} \\
&= \lim_{k\to\infty} (1-\alpha_r)^{k-1}\mathbf{r}^{(0)} + \gamma^{k-1}\alpha_r \alpha_p^2 \boldsymbol{P}^2 \mathbf{x}^{(0)} = 0
\end{aligned}$$

In the limit, $\mathbf{r}^{(k)}$ the residual is 0, implying the convergence of PageRank linear system using the Richardson.

2. Restating the update rule:

$$\begin{aligned}
\mathbf{x}^{(k)} &= (\boldsymbol{I} - \alpha_r(\boldsymbol{I} - \alpha_p \boldsymbol{P}))\mathbf{x}^{(k-1)} + \alpha_r(1-\alpha_p)\mathbf{v} \\
&= (1-\alpha_r)\mathbf{x}^{(k-1)} + \alpha_r\left(\alpha_p \boldsymbol{P}\mathbf{x}^{(k-1)} + (1-\alpha_p)\mathbf{v}\right)
\end{aligned}$$

Also, relative error:

$$\frac{\|(\boldsymbol{I} - \alpha_p \boldsymbol{P})\mathbf{x} - (1-\alpha_p)\mathbf{v}\|}{\|(1-\alpha_p)\mathbf{v}\|} = \frac{\|\mathbf{x} - \alpha_p \boldsymbol{P}\mathbf{x} - (1-\alpha_p)\mathbf{v}\|}{\|(1-\alpha_p)\mathbf{v}\|}$$

The major sparse interaction for both of these is a single matmul.

Code

```
using SparseMatricesCSR, LinearAlgebra, SparseArrays, Random
using ZipFile, DelimitedFiles

function row_projection(rowptr, colval, nzval, row_idx, x)
```

```
        res = .0

        for row_ptr in rowptr[row_idx]:(rowptr[row_idx+1]-1)
                res += x[colval[row_ptr]] * nzval[row_ptr]
        end

        return res
end

function pagerank(rowptr, colval, nzval, n::Integer, v::Vector,
   alpha_richardson::Number, alpha_pagerank::Number, T_max::Integer,
   resid_tol::Number)
        v *= alpha_pagerank
        x_hat = 1 * v

        # create alpha * P
        nzval *= alpha_pagerank

        # csr_matmul
        p_matmul = (x) -> map((i) -> row_projection(rowptr, colval, nzval
           , i, x), 1:n)

        itr = 0
        for itr in 1:T_max
                p_matmul_xhat = p_matmul(x_hat)
                if norm(x_hat - p_matmul_xhat - (1 - alpha_pagerank) * v)
                   /norm((1 - alpha_pagerank) * v) < resid_tol
                        return (x_hat, itr)
                end
                x_hat = (1 - alpha_richardson) * x_hat + alpha_richardson
                    * (p_matmul_xhat + (1 - alpha_pagerank) * v)
        end
        return (x_hat, T_max)
end

function load_data()
        r = ZipFile.Reader("wikipedia-2005.zip")
        try
                @assert length(r.files) == 1
                f = first(r.files)
                data = readdlm(f,'\n',Int)
                n = data[1]
                colptr = data[2:2+n] # colptr has length n+1
                rowval = data[3+n:end] # n+2 elements before start of
                   rowval
                A = SparseMatrixCSC(n,n,colptr,rowval,ones(length(rowval)
                   )) |>
                A->begin ei,ej,ev = findnz(A); d = sum(A;dims=2);
                return sparse(ej,ei,ev./d[ei], size(A)...) end
        finally
                close(r)
        end
end

alpha_pagerank = 0.85
alpha_richardson = 1.0
T_max = 1000
```

```
tol = 1e-5

P = SparseMatrixCSR(load_data())
rowptr,colval,nzval = P.rowptr, P.colval, P.nzval

v = ones(size(P)[1])/size(P)[1]
x_hat, itrs = pagerank(rowptr, colval, nzval, size(P)[1], v,
   alpha_richardson, alpha_pagerank, T_max, tol)
println(itrs)
```

3. For $\alpha_r = 0.5$ it takes 138 iterations, and for $\alpha_r = 1.0$ it takes 67 iterations to converge.

Problem 4

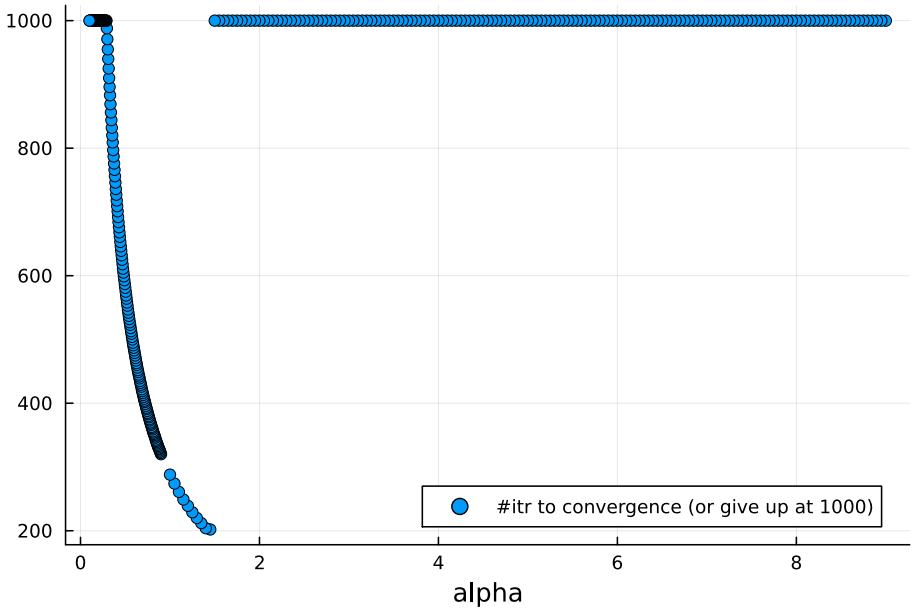
1. The overall strategy is consistent; develop the code to solve with Richardson's method. We set $T = 1000$, so if we do not get below our acceptable tolerance $\varepsilon = 10^{-5}$ by then it is considered not converged (even if the guess is improving).

Code

```
using DelimitedFiles, LinearAlgebra, SparseArrays, Plots


coords = readdlm("chutes-and-ladders-coords.csv",',')
data = readdlm("chutes-and-ladders-matrix.csv",',')

xc = coords[:, 1]
yc = coords[:, 2]

TI = Int.(data[:,1])
TJ = Int.(data[:,2])
TV = data[:,3]
T = sparse(TI, TJ, TV, 101, 101)
A = I - T'

is_pd = all((2 .* Array(diag(A)) - sum(A, dims=2)) .>= 0)  # diagonally
    dominant implies PD
println("Is `A` definitely PD? ", is_pd)

y = ones(101)
y[100] = 0

function richardson(A, fv::Vector, T_max::Integer, alpha::Number, tol::
    Number)
        x_hat = 1 * fv  # copy-by-value, not reference (https://
            stackoverflow.com/a/76109083/10671309)
        for itr in 1:T_max
                x_hat = (I - (alpha * A)) * x_hat + (alpha * fv)
                if norm(A * x_hat - fv) / norm(fv) < tol
                        return (x_hat, itr)
                end
        end
        return (x_hat, T_max)
end

T_max = 1000
alpha = .2
tol = 1e-5

alphs = []
itrs = []
for alph_pow in 0:1
        for alph_coeff in 9:-.05:1
                local alpha = alph_coeff * 10^(-1. *alph_pow)
                local (x, itr) = richardson(A, y, T_max, alpha, tol)

                push!(alphs, alpha)
                push!(itrs, itr)
        end
end
```

```
p = scatter(alphs, itrs, label="#itr to convergence (or give up at 1000)"
    , xlabel="alpha")
savefig(p, "q4-1.pdf")
println(alphs[findall(x-> x == minimum(itrs), itrs)][end])
```

The following plot summarizes each of the desired dimensions. The choice of alpha is denoted by the x-axis. Choices that converge include alpha where the y-axis is less than 1000, and the speed of convergence is ranked in order of fewest steps required (fastest). The best choice obtained by the search routine is $\alpha = 1.45$.



2. First, let's write out the linear equation we solved for in HW6:

$$(\boldsymbol{I} - \boldsymbol{T}^T)\mathbf{x} = \mathbf{y}$$

Julia's solver gets us:

$$\mathbf{x} = (\boldsymbol{I} - \boldsymbol{T}^T)^{-1}\mathbf{y}$$

We only want the 100th index. Using ideas from Q1-5, we bound $\rho(\boldsymbol{T}^T) \leq \|\boldsymbol{T}^T\|_\infty = 1$ as every row sums to 1 by definition. We know that for $\rho(\boldsymbol{T}^T) < 1$, $\lim_{\ell\to\infty} \sum_{k=0}^{\ell} (\boldsymbol{T}^T)^k = (\boldsymbol{I} - \boldsymbol{T}^T)^{-1}$

$$\ell = [(\boldsymbol{I} - \boldsymbol{T}^T)^{-1}\mathbf{y}]_{100}$$

Using a one-hot encoded vector as selection:

$$= \mathbf{e}_{100}(\boldsymbol{I} - \boldsymbol{T}^T)^{-1}\mathbf{y}$$

Applying Neumann Series:

$$= \mathbf{e}_{100} \sum_{k=0}^{\infty} (\boldsymbol{T}^T)^k \mathbf{y}$$

We can extract the first term:

$$= \langle \mathbf{e}_{100}, \mathbf{y} \rangle + \mathbf{e}_{100} \sum_{k=1}^{\infty} (\boldsymbol{T}^T)^{k-1} \boldsymbol{T}^T \mathbf{y}$$

Since $\mathbf{y}_{100} = 0$, these are perpendicular vectors and nullify each other:

$$= 0 + \mathbf{e}_{100} \sum_{k=1}^{\infty} (\boldsymbol{T}^T)^{k-1} \boldsymbol{T}^T \mathbf{y}$$

$$= \sum_{k=1}^{\infty} \mathbf{e}_{100} (\boldsymbol{T}^T)^{k-1} k \boldsymbol{T}_{101,:}$$

Our result is a scalar, we can transpose the result and change nothing:

$$= \sum_{k=1}^{\infty} k \boldsymbol{T}^{k-1} \boldsymbol{T}_{:,101} \mathbf{e}_{100}$$

$$\ell = \sum_{k=1}^{\infty} k \left[\boldsymbol{T}^{k-1} \boldsymbol{T}_{:,101} \right]_{100}$$

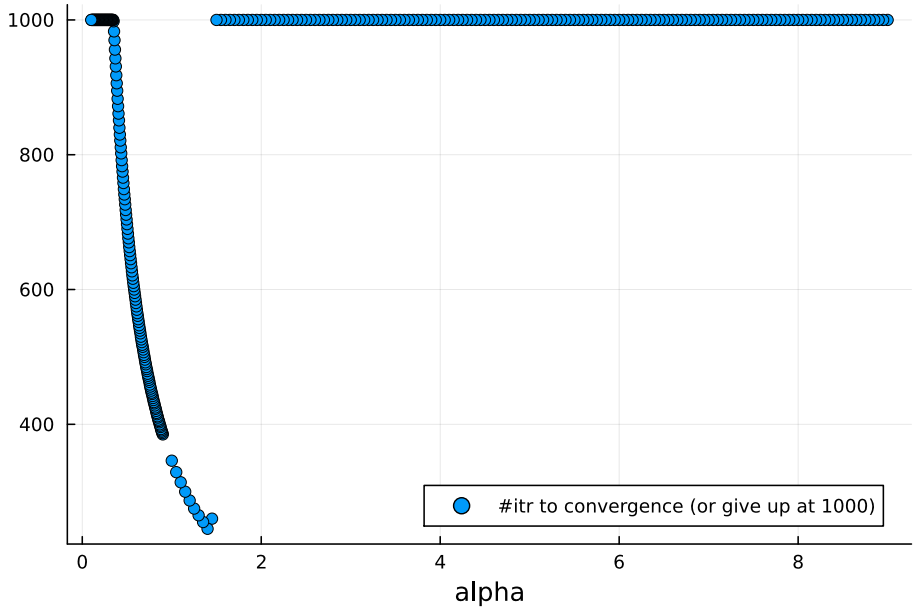
Which is the desired equivalent markovian computation.

Problem 5

Since we are working with a convex problem, any local minima is automatically a global minima. We can check for converge to the global optimizer not by comparing the residual but by checking for the gradient norm, as $\|\mathbf{g}^{(k)}\| < \varepsilon$ for some $\varepsilon > 0$ is a valid convergence criteria. In this case, we'll set $\varepsilon = 10^{-5}$.

```
function steepest_descent(A, fv::Vector, T_max::Integer, tol::Number)
        x_hat = 1 * fv  # copy-by-value, not reference (https://stackoverflow
            .com/a/76109083/10671309)
        for itr in 1:T_max
                g_k = A * x_hat - fv    # matrix-vector product
                x_hat -= ((g_k' * g_k) / (g_k' * A * g_k)) * g_k    # extra (
                    not-allowed) matrix product, included only for discussion;
                     please see main implementation below
                if norm(g_k) < tol
                        return (x_hat, itr)
                end
        end
        return (x_hat, T_max)
end

function steepest_descent(A, fv::Vector, T_max::Integer, alpha::Number, tol::
    Number)
        x_hat = 1 * fv  # copy-by-value, not reference (https://stackoverflow
            .com/a/76109083/10671309)
        for itr in 1:T_max
                g_k = A * x_hat - fv    # matrix-vector product
                x_hat -= alpha * g_k
                if norm(g_k) < tol
                        return (x_hat, itr)
                end
        end
        return (x_hat, T_max)
end
```

Solving chutes and ladders using steepest descent with fixed step sizes results in a best-choice $\alpha = 1.4$ which solves it in 245 steps. Interestingly, with adaptive stepsizes, this took 257 iterations.



Problem 6

Code

```
using SparseMatricesCSR, DelimitedFiles, LinearAlgebra, SparseArrays, Plots,
    Random

function partial_row_projection(rowptr, colval, nzval, row_idx, skip_col_idx,
     x)
        res = .0
        skip_val = 1

        for row_ptr in rowptr[row_idx]:(rowptr[row_idx+1]-1)  # only work on
            k non-zero entries
                if colval[row_ptr] != skip_col_idx
                        res += x[colval[row_ptr]] * nzval[row_ptr]
                else
                        skip_val = nzval[row_ptr]
                end
        end

        return res, skip_val
end

function random_coordinate_descent(rowptr, colval, nzval, y::Vector, T_max::
    Integer, tol::Number)
        x_hat = 1 * y  # copy-by-value, not reference (https://stackoverflow.
            com/a/76109083/10671309)
        for epoch in 1:T_max
                x_hat_new = similar(x_hat)

                for _ in 1:length(y)
                        itr = rand(1:length(y))
                        proj_val, skip_val = partial_row_projection(rowptr,
                            colval, nzval, itr, itr, x_hat)
                        x_hat_new[itr] = (y[itr] - proj_val) / skip_val
                end

                if norm(x_hat_new - x_hat) < tol
                        return (x_hat, epoch * size(y)[1])
                end
                x_hat = 1 * x_hat_new
        end
        return (x_hat, T_max * size(y)[1])
end

function cyclic_coordinate_descent(rowptr, colval, nzval, y::Vector, T_max::
    Integer, tol::Number)
        x_hat = 1 * y  # copy-by-value, not reference (https://stackoverflow.
            com/a/76109083/10671309)
        for epoch in 1:T_max
                x_hat_new = similar(x_hat)

                for itr in 1:length(y)
                        proj_val, skip_val = partial_row_projection(rowptr,
                            colval, nzval, itr, itr, x_hat)
                        x_hat_new[itr] = (y[itr] - proj_val) / skip_val
```

```
                end

                if norm(x_hat_new - x_hat) < tol
                        return (x_hat, epoch * size(y)[1])
                end
                x_hat = 1 * x_hat_new
        end
        return (x_hat, T_max * size(y)[1])
end


coords = readdlm("chutes-and-ladders-coords.csv",',')
data = readdlm("chutes-and-ladders-matrix.csv",',')

xc = coords[:, 1]
yc = coords[:, 2]

TI = Int.(data[:,1])
TJ = Int.(data[:,2])
TV = data[:,3]
T = sparse(TI, TJ, TV, 101, 101)
A = SparseMatrixCSR(I - T')

is_dd = all((2 .* Array(diag(A)) - sum(A, dims=2)) .>= 0)
println("Is `A` diagonally dominant? ", is_dd)

y = ones(101)
y[100] = 0

Random.seed!(0)
T_max = 1000000
tol = 1e-10

x_oracle = A\y

x_r, itr_r = random_coordinate_descent(A.rowptr, A.colval, A.nzval, y, T_max,
    tol)
x_c, itr_c = cyclic_coordinate_descent(A.rowptr, A.colval, A.nzval, y, T_max,
    tol)

println(norm(x_oracle - x_r))
println(norm(x_oracle - x_c))
```

Output

```
Is `A` diagonally dominant? true
2.4148946143466656e-9
2.4148946143466656e-9
```

The random case depends on the mercy of the gods and is *much* slower to converge as a result:

```
julia> itr_r
20094152

julia> itr_c
63024
```

Problem 7

```
function gen_coord_descent(A, y::Vector, T_max::Integer, tol::Number,
    n_coords::Integer)
        x_hat = 1 * y  # copy-by-value, not reference (https://stackoverflow.
            com/a/76109083/10671309)
        for itr in 1:T_max
                g_k = A * x_hat - y   # matrix-vector product
                update = ((g_k' * g_k) / (g_k' * A * g_k)) * g_k

                for start in 1:n_coords:length(y)-n_coords
                        for coord in start:start+n_coords
                                x_hat[coord] -= update[coord]
                        end
                end

                if norm(g_k) < tol
                        return (x_hat, itr * length(y) / n_coords)
                end
        end
        return (x_hat, T_max * length(y) / n_coords)
end
```

Problem 8

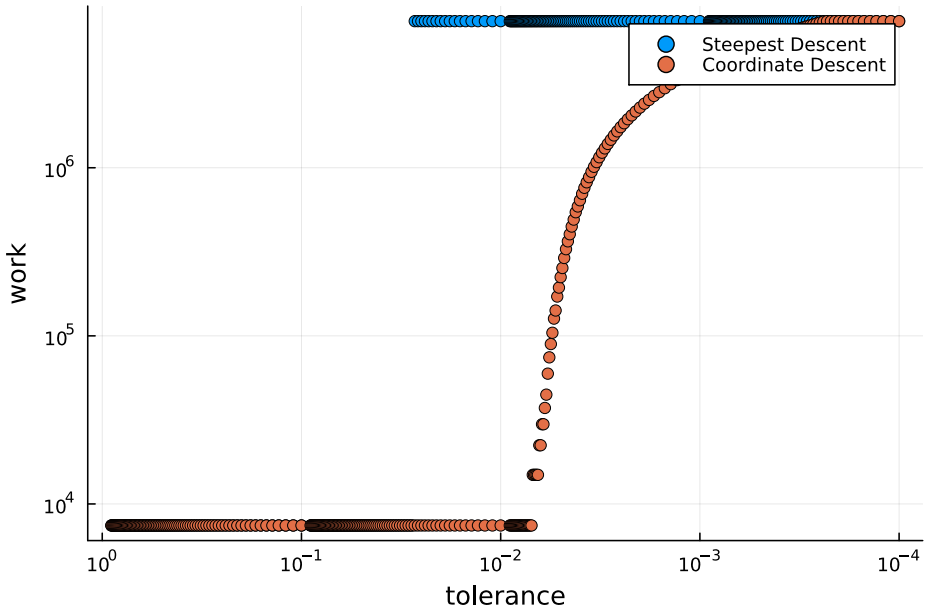
Code

```
include("q1.jl")
include("q5.jl")
include("q6.jl")

# 2d laplacian
n = 40
A, y = laplacian(n, (x, y) -> 1)

A = SparseMatrixCSR(A[4n+1:end,4n+1:end])
y = y[4n+1:end]

# chutes and ladders
coords = readdlm("chutes-and-ladders-coords.csv",',')
data = readdlm("chutes-and-ladders-matrix.csv",',')

xc = coords[:, 1]
yc = coords[:, 2]

TI = Int.(data[:,1])
TJ = Int.(data[:,2])
TV = data[:,3]
T = sparse(TI, TJ, TV, 101, 101)
A = SparseMatrixCSR(I - T')

y = ones(101)
y[100] = 0


T_max = 1000
alpha = 1.4
tol = 1e-4

tols = []
works_stpd = []
works_coord = []
for tol_pow in 1:4
        for tol_coeff in 9:-.1:1
                local tol = tol_coeff * 10^(-1. * tol_pow)
                local x_stpd, itr_stpd = steepest_descent(A, y, T_max, tol)
                local work_stpd = itr_stpd * nnz(A)

                local x_coord, itr_coord = cyclic_coordinate_descent(A.rowptr
                    , A.colval, A.nzval, y, T_max, tol)
                local work_coord = itr_coord / length(y) * nnz(A)

                push!(tols, tol)
                push!(works_stpd, work_stpd)
                push!(works_coord, work_coord)
        end
end

p = scatter(tols, works_stpd, label="Steepest Descent")
```

Tolerance	Work by Steepest Descent	Work by Coordinate Descent
0.9	7449	7449
0.65	7449	7449
0.4	7449	7449
0.15000000000000002	7449	7449
0.07100000000000001	7449	7449
0.046000000000000006	7449	7449
0.021000000000000005	-	7449
0.0077	-	7449
0.005200000000000001	-	171327
0.0027	-	1.556841e6
0.0008300000000000001	-	4.402359e6
0.00058	-	5.266443e6
0.00033	-	6.62961e6
0.0001	-	7.449e6

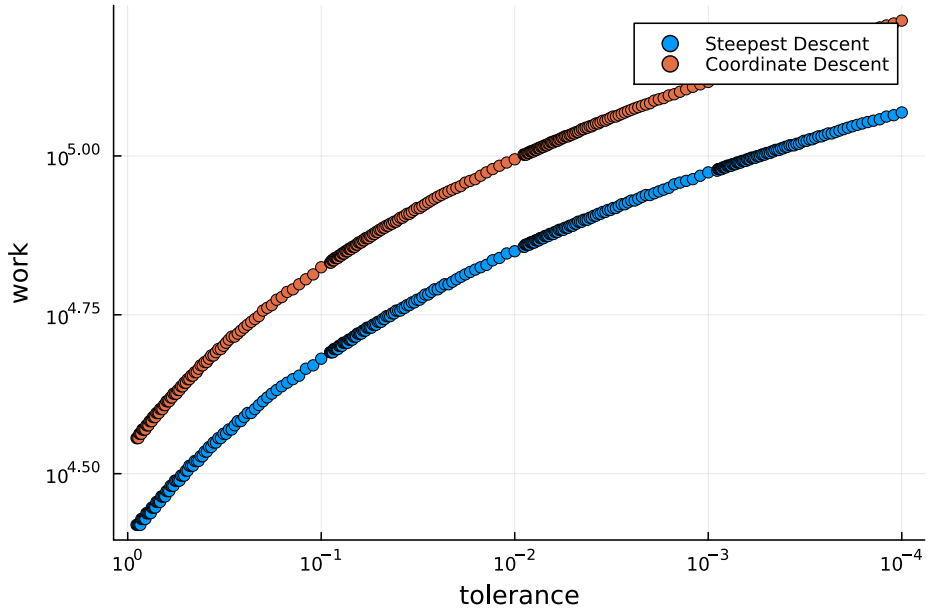
```
p = scatter!(tols, works_coord, label="Coordinate Descent", xlabel="tolerance
    ", ylabel="work", yscale=:log10, xscale=:log10)
p = xflip!(true)
```

1. For the 2D Laplacian, steepest descent does well in the start (when tolerances are high) but as we up the ante coordinate descent comes out a clear winner.



2. For chutes and ladders, steepest descent outperforms coordinate descent by across every convergence threshold.

Tolerance	Work by Steepest Descent	Work by Coordinate Descent
0.9	26266	35973
0.65	29121	40541
0.4	34260	47393
0.15000000000000002	43967	61097
0.07100000000000001	51390	71375
0.046000000000000006	55958	77656
0.021000000000000005	63381	88505
0.0077	73659	102209
0.005200000000000001	77656	107919
0.0027	83937	117055
0.0008300000000000001	95928	133614
0.00058	99354	138753
0.00033	105064	146176
0.0001	117055	163306



For the table, we sample from various tolerance levels to prevent overcrowding.

Problem 9

We have the following linear system:

$$\boldsymbol{A}\mathbf{x} = \mathbf{b} \iff (\boldsymbol{D} + \boldsymbol{N})\mathbf{x} = \mathbf{b}$$

We also have diagonal dominance, where for some $\gamma < 1$ it holds that:

$$\|\boldsymbol{A}\mathbf{x} - \boldsymbol{D}\mathbf{x}\| \leq \gamma \|\boldsymbol{D}\mathbf{x}\|$$

We update elements elementwise:

$$\mathbf{x}_i^{(k)} = \frac{1}{\boldsymbol{A}_{i,i}}(\mathbf{b}_i - \langle \boldsymbol{N}_{i,:}, \mathbf{x}^{(k-1)} \rangle)$$

Which can be rewritten for the entire guess $\mathbf{x}^{(k)}$ as:

$$\mathbf{x}^{(k)} = \boldsymbol{D}^{-1}(\mathbf{b} - \boldsymbol{N}\mathbf{x}^{(k-1)})$$

Since $\boldsymbol{A}$ is diagonally dominant, and diagonal dominance implies positive definiteness (we used this fact in the programming questions), we can approach this problem through the lens of convex minimization:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T \boldsymbol{A}\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

Under this formulation, we can show that the update at each iteration is the gradient scaled coordinate-wise:

$$\begin{aligned}
\mathbf{x}^{(k)} &= \boldsymbol{D}^{-1}(\mathbf{b} - \boldsymbol{N}\mathbf{x}^{(k-1)}) \\
&= \boldsymbol{D}^{-1}(\mathbf{b} - (\boldsymbol{A} - \boldsymbol{D})\mathbf{x}^{(k-1)}) \\
&= \boldsymbol{D}^{-1}\mathbf{b} - \boldsymbol{D}^{-1}\boldsymbol{A}\mathbf{x}^{(k-1)} + \boldsymbol{D}^{-1}\boldsymbol{D}\mathbf{x}^{(k-1)} \\
\mathbf{x}^{(k)} &= \mathbf{x}^{(k-1)} - \underbrace{\boldsymbol{D}^{-1}}_{\text{step size}} \underbrace{(\boldsymbol{A}\mathbf{x}^{(k-1)} - \mathbf{b})}_{\mathbf{g}^{(k-1)}}
\end{aligned}$$

A sufficient condition for the convergence of steepest descent is $\alpha < 2/\rho(\boldsymbol{A})$, for the elementwise case we will prove that $\boldsymbol{D}_{ii}^{-1} < 2/\rho(\boldsymbol{A})$ for every index i. By diagonal dominance, we have that:

$$\rho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|_\infty = \max_i \sum_j |\boldsymbol{A}_{i,j}| < 2 \max_i(\boldsymbol{D}_{i,i})$$

$$\therefore \frac{2}{\rho(\boldsymbol{A})} > \frac{1}{\max_i \boldsymbol{D}_{i,i}} = \left(\max_i \boldsymbol{D}_{i,i}\right)^{-1}$$

Since we have proven the desired statement for the max, this also follows for any i, and through this equivalence we have that Jacobi does converge.

Problem 10

In the linear algebraic approximation, we can decompose viral spread at time t with the following equation:

$$\mathbf{x}^{(t)} = \rho \boldsymbol{A}\mathbf{x}^{(t-1)} = (\rho \boldsymbol{A})^t \mathbf{x}^{(0)} = (\boldsymbol{V}\Lambda^t \boldsymbol{V}^T)\mathbf{x}^{(0)}$$

If the largest eigenvalue of $\rho\boldsymbol{A} > 1$, as $t \to \infty$ the probability of infection diverges. In contrast for a maximum eigenvalue < 1, spread as timesteps progress tends to 0, thus our approximate model (calculating cumulative probability) reduces to a convergent Nuemann series, which depends on the spectral radius to guarantee convergence and find a steady state for this suitable regime.