UCB Math 124, Spring 2022: Final Exam

Prof. Persson, May 12, 2022

SOLUTIONS

Name: ______________________________

SID: ______________________________

Instructions:

- One sheet of notes, no books, no calculators.
- Exam time 180 minutes, do all of the problems.
- You must justify your answers for full credit.
- All computer codes must be written in the Julia programming language, using only the functionality and the packages covered in the course (unless otherwise specified).
- Write your answers next to or below each problem.
- If you need more space, use reverse side or scratch pages.
  Indicate clearly where to find your answers.

1. (4 points) Describe briefly (in words or in mathematical notation) what the following Julia commands do.

a)
```
function f(x)
    y,j = x[1],1
    for i = 2:length(x)
        if x[i] > y
            y,j = x[i],i
        end
    end
    return j
end
```

Solution: Find the index of largest element in the vector x.

b)
```
# Given a matrix A
B = A[:, sum(A .< 0, dims=1)[:] .% 2 .== 0]
```

Solution: Create a matrix B containing only the columns of A that have an even number of negative numbers.

(2 points) Find the output of the following Mathematica commands, with justification.

c)
```
f[x_ y_] := f[x] - f[y]
Sum[f[x[k + 1] x[k]], {k, 1, n}]
```

Solution: With the given rules, we have that

$$\begin{aligned}\sum_{k=1}^{n} f(x_{k+1}x_k) &= \sum_{k=1}^{n} f(x_{k+1}) - f(x_k) \\ &= f(x_2) - f(x_1) + f(x_3) - f(x_2) + \cdots + f(x_{n+1}) - f(x_n) \\ &= f(x_{n+1}) - f(x_1)\end{aligned}$$

2. (4 points) A box contains 20 balls: 5 red, 5 yellow, 3 green, and 7 brown. You draw `n` balls from the hat randomly (without putting them back). You win the game if the draw results in an even number of red balls, more brown balls than yellow balls, and at least one ball of each color. Write a Julia function `game_sim(n,ntrials)` which uses Monte Carlo simulation with `ntrials` trials to estimate the probability that you win the game.

Solution: This problem can be solved in a way similar to the poker problem in the homework. For example, draw `n` random integers between 1 and 20 (without replacement), using the `randperm` function. Let 1 to 5 represent the red balls, 6 to 10 the yellow balls, etc.

```
using Random

function game_sim(n, ntrials)
    nwin = 0
    for i = 1:ntrials
        p = randperm(20)[1:n]
        nred = count(1 .<= p .<= 5)
        nyellow = count(6 .<= p .<= 10)
        ngreen = count(11 .<= p .<= 13)
        nbrown = count(14 .<= p .<= 20)
        if nred % 2 == 0 && nbrown > nyellow &&
           nred > 0 && nyellow > 0 && ngreen > 0 && nbrown > 0
            nwin += 1
        end
    end
    return nwin / ntrials
end
```

3. (4 points) Define the *quality* of a triangle T by

$$q(T) = \frac{(b+c-a)(a+c-b)(a+b-c)}{abc}$$

where a,b,c are the side lengths of T. Consider a triangular mesh represented by an array of points `p` and an array of triangle indices `t` (same format as in the lecture notes and in the mesh generation project). Write a Julia function `triqual(p,t)` which computes and returns an array containing the quality of each triangle of the mesh.

Solution:

```julia
using LinearAlgebra

function triqual(p,t)
    function quality(ptri)
        a = norm(ptri[2] - ptri[1])
        b = norm(ptri[3] - ptri[2])
        c = norm(ptri[1] - ptri[3])
        return (b+c-a) * (a+c-b) * (a+b-c) / (a*b*c)
    end
    return [ quality(p[tri]) for tri in t ]
end
```

4. A number is called a k-stepping number if all adjacent decimal digits have an absolute difference of k ($0 \leq k < 10$). For example, 321 is a 1-stepping number, 747 is a 3-stepping number, but 421 is not a stepping number. Any single-digit number is considered a stepping number for any k.

a) (3 points) Write a Julia function `isstepnumber(n,k)` which returns `true` if n is a k-stepping number. You function should only use $\mathcal{O}(1)$ memory (that is, independent of the number of digits in n).

b) (1 points) Write a *one-line* definition of the form

```
allstepnumbers(nmax,k) = < your one-line code >
```

which defines a function that returns an array of all k-stepping numbers between `1` and `nmax`.

Solution:

```julia
function isstepnumber(n,k)
    d = n % 10
    n ÷= 10
    while n > 0
        newd = n % 10
        if abs(newd - d) != k
            return false
        end
        n ÷= 10
        d = newd
    end
    return true
end

# Alt 1
allstepnumbers(nmax,k) = findall(isstepnumber.(1:nmax, k))
# Alt 2
allstepnumbers2(nmax,k) = [ n for n = 1:nmax if isstepnumber(n, k) ]
```

5. (4 points) Suppose you are given vectors x, y of length N, and want to approximate the data x_i, y_i, $i = 1, \ldots, N$, by a circle. More precisely, you want to find a circle center x_c, y_c and a radius r that minimize the sum of the squares of the distances between each point x_i, y_i and the closest point on the circle.

Write a function `fitcircle(x,y)` which solves this problem using the black-box optimizer in the `Optim` package and the following calling syntax:

```
res = optimize(fobj, xinit, GradientDescent(); autodiff=:forward)
```

where `fobj` is the function to minimize, and the initial vector `xinit` is all zeros. Return the minimizing vector `res.minimizer`.

Solution:

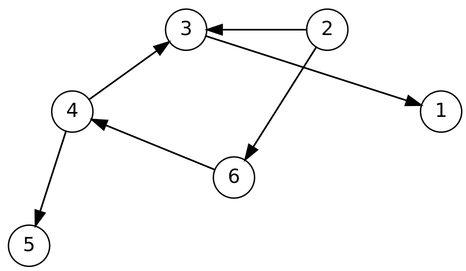
```
using Optim

function fitcircle(x, y)
    function fobj(α)
        xc,yc,r = α
        xyr2 = @. (x-xc).^2 + (y-yc).^2
        d = (sqrt.(xyr2) .- r).^2
        return sum(d)
    end

    res = optimize(fobj, zeros(3), GradientDescent(); autodiff=:forward)
    return res.minimizer
end
```

6. (4 points) We use the data structures for graphs from the lecture notes:

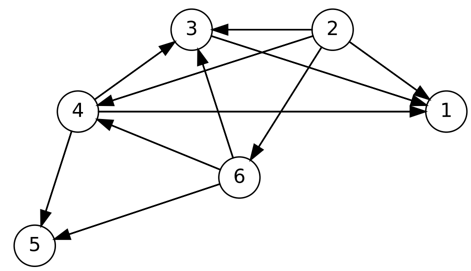
```
struct Vertex
    neighbors::Vector{Int}
end
```

```
struct Graph
    vertices::Vector{Vertex}
end
```

Create a function `nbnb_graph(g)` which creates and returns a new *neighbors-neighbors graph* `g1` such that there is an edge between vertices i, j either if j is a neighbor of i or if j is a neighbor of any of the neighbors of i in `g`. In other words, a vertex in `g1` has edges to all the neighbors of the node in `g` as well as to all of its neighbors' neighbors. See figure below for an example. The `neighbors` vectors should not contain any duplicate values.
Hint: Start with `g1 = deepcopy(g)` to create a duplicate of `g` which you can then modify.



Input graph `g`



Neighbors-neighbors graph `g1`

Solution: You can use any method for traversing the vertices of the graph, such as a for-loop over all the vertices.

```
function nbnb_graph(g::Graph)
    g1 = deepcopy(g)
    for iv = 1:length(g.vertices)
        for nb in g.vertices[iv].neighbors
            for nbnb in g.vertices[nb].neighbors
                if nbnb ∉ g1.vertices[iv].neighbors
                    push!(g1.vertices[iv].neighbors, nbnb)
                end
            end
        end
    end
    g1
end
```