Take-Home Final

ELEMENTS OF SCIENTIFIC COMPUTING

Instructions: Please create a folder in your GitHub repository called "thfinal" and for each problem below push a *different* file (as required per the problem) to this folder.

Problem 1. (8 pts) (*Numerical Integration*) In class, we have discussed various methods of numerical integration to solve for the integral of functions of a single variable, however we can easily generalize these techniques to solve for the integrals of multivariate functions.

That is, for a surface in 3-dimensional space we can approximate the volume under the surface, and above a particular region on the 2D plane below it, by adding up the volumes of many thin parallelepipeds. For example, if we want to use the midpoint rule with a square base for each of the rectangular parallelepiped subdivisions, then we get:

$$\int_c^d \int_a^b f(x,y)\,dxdy \approx \Delta x \Delta y \cdot [f(x_1^*, y_1^*) + f(x_1^*, y_2^*) + ... + f(x_1^*, y_n^*) + f(x_2^*, y_1^*) + ... + f(x_n^*, y_n^*)]$$

where $\Delta x = \frac{b-a}{n}$ and $\Delta y = \frac{d-c}{n}$ when subdividing the region under the surface into a n^2 grid. Also, x_i^* here is the midpoint of the interval $[x_{i-1}, x_i]$, while y_j^* is the midpoint of the interval $[y_{j-1}, y_j]$. Note: this definition is 0-indexed.

Write a program in Julia to evaluate the double integral

$$\int_0^1 \int_0^1 e^{-xy}\,dxdy$$

using the midpoint rule as defined above. Subdivide the $[0,1] \times [0,1]$ region into 10,000 squares.

You should push to GitHub your Julia code for this problem in a .ipynb or .jl file.

Problem 2. (8 pts) (*Time-Dependent Heat Equation*) Consider the time dependent heat equation:

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0$$

for $0 \leq x \leq 1$ and $t \geq 0$, with boundary conditions:

$$u(t,0) = 0,\ u(t,1) = 0,$$

and initial condition:

$$u(0,x) = \begin{cases} 2x & \text{if } 0 \leq x \leq 0.5 \\ 2 - 2x & \text{if } 0.5 \leq x \leq 1 \end{cases}$$

Discretize the given heat equation with $\Delta x = 0.05$ and $\Delta t = 0.0012$ (see lecture 3 notes), and solve this problem in Julia in the time interval $[0, 0.06]$. Plot the solution for every $10\Delta t$ starting with the plot for $t = 0$ and ending with the plot for $t = 0.06$ using PyPlot. For inspiration on this problem, here is an example on solving the time dependent heat equation using Matlab.

You should push to GitHub your Julia code for this problem in a .ipynb or .jl file.

Problem 3. (6 pts) (*Logistic Regression and Optimization*) For this problem, consider the `LogReg.ipynb` notebook from lecture 8, in particular the snippet of code implementing the gradient descent algorithm for logistic regression:

```
function gradientDescent(X, y, theta, alpha, iter)
    m = length(y)
    Jhist = zeros(iter,1)
    for k=1:iter
        delta = zeros(size(theta))
        for i=1:m
            delta += (1/m)*(sigmoid(X[i,:]*theta) - y[i])[1]*(X[i,:])'
        end
        theta = theta - (alpha/m)*delta
        Jhist[k] = cost(X, y, theta)[1]
    end
    return theta, Jhist
end
```

Play around with the initial `theta`, `alpha` and `iter` inputs and observe `Jhist`. For *each combination* of these inputs (you should try at least 5 combinations!) write two or three sentences describing what is going on with the output. Finally, write a paragraph with a hypothesis as to why the gradient descent algorithm does not perform very well on this logistic regression problem. Note: you do not need to find a combination that causes it to converge.

You should push to GitHub your .tex and .pdf files for this problem.

Problem 4. (8 pts) (*Student Presentations*) Answer the following questions regarding the topics that your fellow students have presented this quarter:

a) (2 pts) Say you did something wrong on this take-home and you would like to replace your local changes with the last content pushed to your repository, how can you do that using `git`?

b) (2 pts) Consider the following snippet of code:

```
nprocs()
addprocs(4)
a = dones((4000,1), workers()[1:4], [4,1])
b = {@spawnat p sum(localpart(a)) for p=procs(a)}
c = map(fetch, b)
sum(c)
```

At first without running the code, what do you think the output will be? Paste it into IJulia, run it and verify your answer. Add in-line comments to each line describing what the code on that line is doing.

c) (2 pts) Multiply two 4×4 matrices of your choice (non-zero, non-identity matrices only!) using the *Strassen Algorithm*. You may do it by hand and typeset your work in LaTeX or you may implement this using Julia.

d) (2 pts) Write a function in Julia that takes as input a comma separated file and outputs a tab separated file with the same data contained in the input.

You should push to GitHub your .tex and .pdf files, as well as your Julia code for all parts of this problem in a .ipynb or .jl file.