BasicProblems

October 25, 2016

Problem 1 Use Julia's array and control flow syntax in order to define the NxN Strang matrix:

$$\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{bmatrix}$$

```
In [ ]: #### Prepare Data

        X = rand(1000, 3)                # feature matrix
        a0 = rand(3)                     # ground truths
        y = X * a0 + 0.1 * randn(1000);  # generate response
```

Problem 2 Given an Nx3 array of data (`randn(N,3)`) and a Nx1 array of outcomes, produce the data matrix `X` which appends a column of 1's to the data matrix, and solve for the 4x1 array β via β`X = b` using `qrfact` or `\`. (Note: This is linear regression)

Problem 3 Compare your results to that of using `llsq` from `MultivariateStats.jl` (note: you need to go find the documentation to find out how to use this!)

Problem 4 Compare your results to that of using ordinary least squares regression from `GLM.jl`

Problem 5 The logistic difference equation is defined by the recursion

$$b_{n+1} = r * b_n(1 - b_n)$$

where b_n is the number of bunnies at time n. Starting with $b_0 = .25$, by around 400 iterations this will reach a steady state. This steady state (or steady periodic state) is dependent on r. Write a function which solves for the steady state(s) for each given r, and plot "every state" in the steady attractor for each r (x-axis is r, y=value seen in the attractor) using Plots.jl. Take $r \in (2.9, 4)$

Problem 6 Make a function `person_info(x)` where, if `x` is a any type of person, print their name. However, if `x` is a student, print their name and their grade. Make a new type which is a graduate student, and have it print their name and grade as well. If `x` is anything else, throw an error. Do not using branching (`if`), use multiple dispatch to solve the problem!

Note that in order to do this you will need to re-structure the type hierarchy. Make an AbstractPerson and AbstractStudent type, define the subclassing structure, and write dispatches on these abstract types. Note that you cannot define subclasses of concrete types!

(Not only is the multiple-dispatch way more Julian, we will see later that it also has a lot of performance enhancements due to how it interacts with the compiler).

Distribution Quantile Problem (From Josh Day) To find the quantile of a number `q` in a distribution, one can use a Newton method

$$\theta_{n+1} = \theta_n - \frac{cdf(\theta) - q}{pdf(\theta)}$$

to have $\theta_n \to$ the value of for the `q`th quantile. Use multiple dispatch to write a generic algorithm for which calculates the `q`th quantile of any `UnivariateDistribution` in Distributions.jl, and test your result against the `quantile(d::UnivariateDistribution,q::Number` function.

Hint: Use $\theta_0 =$ mean of the distribution

Operator Problem In mathematics, a matrix is known to be a linear operator. In many cases, this can have huge performance advantages because, if you know a function which "acts like a matrix" but does not form the matrix itself, you can save the time that it takes to allocate the matrix (sometimes the matrix may not fit in memory!)

Therefore, instead of solving regressions on matrices, let's be brave and generalize our regression algorithm to work on any operator, and make it solve the problem for the Laplacian operator. Here are the steps that are required to do this:

- Make an abstract type `AbstractOperator`
- Re-define AbstractMatrix as a subtype
- Define a concrete type `LaplacianOperator` which holds a function. This function `f(x)` should calculate the discrete Laplacian of `x` (i.e. multiply it on the left by the Strang matrix, but without forming the matrix!)
- Write dispatches for `*(::LaplacianOperator,::Vector)`, `eltype(::LaplacianOperator)`, `size(::LaplacianOperator,d::Integer)`.
- Write `least_square(::Operator,::Vector)` to solve the least-square approximation problem `Ax=b` for any operator. Note that since you do not have a matrix, you cannot use `\` or factorizations like `qrfact`. Instead, take a look at `cg` in IterativeSolvers.jl
- Test your least_square function vs `llsq` and `lm` (where you use the data matrix as a Strang matrix). Do you get the same result?

Problem 8 If you know `start`, `step`, and `stop`, how do you calculate the `i`th value? Can you create a function MyRange which where for `a` being a `MyRange`, and `a[i]` is the correct value? Use the Julia array interface in order to define the function for the `a[i]` syntax on your type.

Problem 9 Do ?linspace. Make your own LinSpace object using the array interface.

http://ucidatascienceinitiative.github.io/IntroToJulia/Html/ArrayIteratorInterfaces

Do your implementations obay dimensional analysis? Try using the package `Unitful` to build arrays of numbers with units (i.e. an array of numbers who have values of Newtons), and see if you can make your LinSpace not give errors.

Problem 10 Check your implementation vs the source code of Ranges.jl. Tim Holy is the master of Julia arrays, learn from him!

Problem 11 Check out the call overloading notebook:

http://ucidatascienceinitiative.github.io/IntroToJulia/Html/CallOverloading

Overload the call on the UnitStepRange to give an interpolated value at intermediate points, i.e. if `a=1:2:10`, then `a(1.5)=2`.