Kolmogorov Backward Equations

Ashutosh Bharambe

July 1, 2020

```
using Flux, StochasticDiffEq
using NeuralNetDiffEq
using Plots
using CuArrays
using CUDAnative
```

0.1 Introduction on Backward Kolmogorov Equations

The backward Kolmogorov Equation deals with a terminal condtion. The one dimensional backward kolmogorov equation that we are going to deal with is of the form :

$$\frac{\partial p}{\partial t} = -\mu(x)\frac{\partial p}{\partial x} - \frac{1}{2}\sigma^2(x)\frac{\partial^2 p}{\partial x^2}, \quad p(T,x) = \varphi(x)$$

for all $ t \in[[0 , T]] $ and for all $ x \in R^d $

The Black Scholes Model The Black-Scholes Model governs the price evolution of the European put or call option. In the below equation V is the price of some derivative , S is the Stock Price , r is the risk free interest rate and σ the volatility of the stock returns. The payoff at a time T is known to us. And this makes it a terminal PDE. In case of an European put option the PDE is:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} - rV = 0, \quad V(T,S) = max\{\mathcal{K} - S, 0\}$$

for all $ t \in[[0 , T]] $ and for all $ S \in R^d $

In order to make the above equation in the form of the Backward - Kolmogorov PDE we should substitute

$$V(S,t) = e^{r(t-T)}p(S,t)$$

and thus we get

$$e^{r(t-T)}\frac{\partial p}{\partial t} + re^{r(t-T)}p(S,t) = -\mu(x)\frac{\partial p}{\partial x}e^{r(t-T)} - \frac{1}{2}\sigma^2(x)\frac{\partial^2 p}{\partial x^2}e^{r(t-T)} + re^{r(t-T)}p(S,t)$$

And the terminal condition

$$p(S,T) = max\{\mathcal{K} - x, 0\}$$

We will train our model and the model itself will be the solution of the equation

0.2 Defining the problem and the solver

We should start defining the terminal condition for our equation:

```
function phi(xi)
    y = Float64[]
    K = 100
    for x in eachcol(xi)
        val = max(K - maximum(x) , 0.00)
        y = push!(y , val)
    end
    y = reshape(y , 1 , size(y)[1] )
    return y
end
```

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

Now we shall define the problem : We will define the σ and μ by comparing it to the orignal equation. The xspan is the span of initial stock prices.

```
d = 1
r = 0.04
sigma = 0.2
xspan = (80.00 , 115.0)
tspan = (0.0 , 1.0)
σ(du , u , p , t) = du .= sigma.*u
μ(du , u , p , t) = du .= r.*u
prob = KolmogorovPDEProblem(μ , σ , phi , xspan , tspan, d)
```

```
KolmogorovPDEProblem
timespan: (0.0, 1.0)xspan: (80.0, 115.0)μ
Main.##WeaveSandBox#437.μSigma
Main.##WeaveSandBox#437.σ
```

Now once we have defined our problem it is necessary to define the parameters for the solver.

```
sdealg = EM()
ensemblealg = EnsembleThreads()
dt = 0.01
dx = 0.01
trajectories = 100000
```

```
100000
```

Now lets define our model m and the optimiser

```
m = Chain(Dense(d, 64, elu),Dense(64, 128, elu),Dense(128 , 16 , elu) , Dense(16 , 1))
use_gpu = false
if CUDAnative.functional() == true
  m = fmap(CuArrays.cu , m)
  use_gpu = true
end
opt = Flux.ADAM(0.0005)
```

```
Flux.Optimise.ADAM(0.0005, (0.9, 0.999), IdDict{Any,Any}())
```

And then finally call the solver

```
@time sol = solve(prob, NeuralNetDiffEq.NNKolmogorov(m, opt, sdealg, ensemblealg),
verbose = true, dt = dt,
            dx = dx , trajectories = trajectories , abstol=1e-6, maxiters = 1000 ,
use_gpu = use_gpu)
```

```
Error: Out of GPU memory trying to allocate 12.207 MiB
Effective GPU memory usage: 8.46% (674.875 MiB/7.787 GiB)
CuArrays allocator usage: 497.040 MiB (capped at 512.000 MiB)
BinnedPool usage: 497.040 MiB (497.040 MiB allocated, 0 bytes cached)
```

0.3 Analyzing the solution

Now let us find a Monte-Carlo Solution and plot the both:

```
monte_carlo_sol = []
x_out = collect(85:2.00:110.00)
for x in x_out
  u_0= [x]
  g_val(du , u , p , t) = du .= 0.2.*u
  f_val(du , u , p , t) = du .= 0.04.*u
  dt = 0.01
  tspan = (0.0,1.0)
  prob = SDEProblem(f_val,g_val,u_0,tspan)
  output_func(sol,i) = (sol[end],false)
  ensembleprob_val = EnsembleProblem(prob , output_func = output_func )
  sim_val = solve(ensembleprob_val, EM(), EnsembleThreads() , dt=0.01,
trajectories=100000,adaptive=false)
  s = reduce(hcat , sim_val.u)
  mean_phi = sum(phi(s))/length(phi(s))
  global monte_carlo_sol = push!(monte_carlo_sol , mean_phi)
end
```

##Plotting the Solutions We should reshape the inputs and outputs to make it compatible with our model. This is the most important part. The algorithm gives a distributed function over all initial prices in the xspan.

```
x_model = reshape(x_out, 1 , size(x_out)[1])
if use_gpu == true
  m = fmap(cpu , m)
end
y_out = m(x_model)
y_out = reshape(y_out , 13 , 1)
```

```
13×1 Array{Float32,2}:
 -6.858921
 -7.0025887
 -7.146932
 -7.2919345
 -7.4375534
 -7.5837555
 -7.7305126
 -7.8778043
 -8.025596
 -8.173863
```

```
-8.322588
-8.471742
-8.621298
```

And now finally we can plot the solutions

```
plot(x_out , y_out , lw = 3 ,  xaxis="Initial Stock Price", yaxis="Payoff" , label =
"NNKolmogorov")
plot!(x_out , monte_carlo_sol , lw = 3 ,  xaxis="Initial Stock Price", yaxis="Payoff"
,label = "Monte Carlo Solutions")
```