STA242 Assignment 2

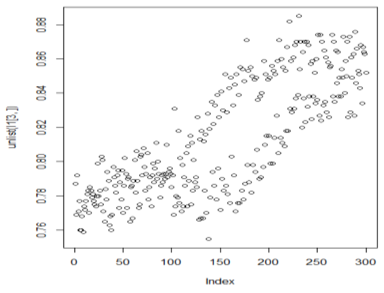
Xu Han

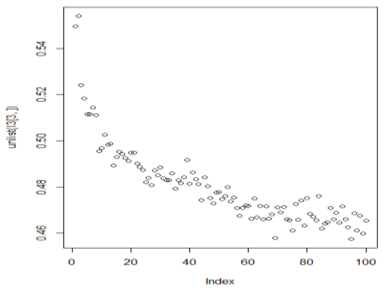
4/30/2015

1. Behavior of the BML model

Suppose we have a grid with m rows and n cols, and there are red cars and blue cars randomly on the grid both with size n. We want to check whether the density 2n/m*n influence the flow. We generate a grids with 100 rows and 100 columns, and assign different density on the grid, and test the average speed over 300 steps.

0.1	0.2	0.3	0.4	0.5
0.92	0.80	0.69	0.448	0.17
0.6	0.7			
0.07	0.02			





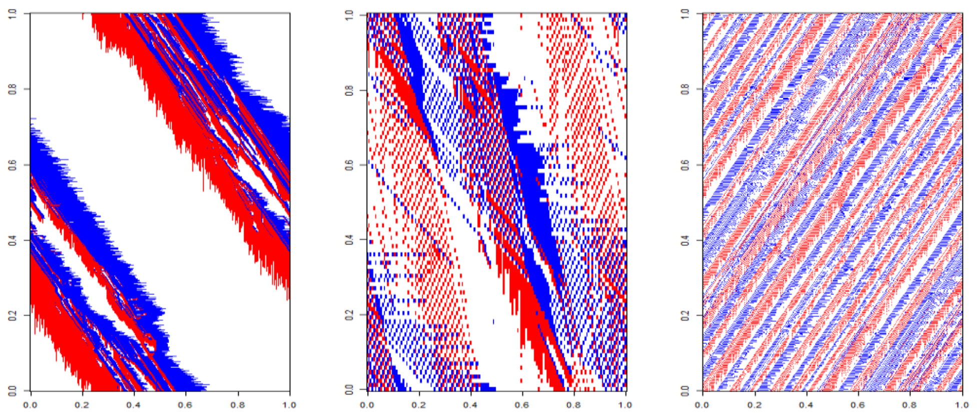
From the table we see that with the density increase, the flow reduce high smoothness to complete jam. From density 0.1 to density 0.5, the smoothness drops more and more fast.
We can see from the scatter plot that the speed is increasing at density 0.2 and is decreasing at density 0.4. This means that at a low density, the flow has self-organize ability and when density is increasing at a certain point, in this case is 0.4, the speed tends to decrease over time which indicates the self-transition to jam phase. So we conclude that if the time is fixed, higher density leads to lower speed; when the density is fixed and is small, the flow has self-organize ability; when the density is fixed and is large, the flow become jammed. We may consider 0.4 as a transition point, and at this density point the velocity of flow became slow.

So we would think if the density does not change, and we change the shape of grid from lattice above to rectangular. Still we make two type cars have the same size. Dose this change affect the flow?
We generate a grid with 150 rows and 90 columns, and assign different density to grids. Still we keep two type cars to have the same size.

0.1	0.2	0.3	0.4	0.5
0.89	0.79	0.66	0.47	0.25
0.6	0.7			
0.09	0.03			

From the table we clearly see that the shape does influence flow. When density is below 0.3, lattice performs more smooth; when the density range from 0.3 to 0.6, the rectangular has higher speed. Especially when the density is between 0.4 and 0.5, the rectangular performs pretty better.

We find that flows may perform some pattern at density around 0.4 over 10000 steps



The left plot is simulated with 512 by 512 grid with 0.4 density, we see the flow is complete jamed. The middle plot is simulated with 150 by 90 grid with 0.4 density, the pattern is disorderd which combining features from both the jammed and free-flowing phases. The right plot is simulated with 512 by 512 grid with 0.3 density and it is free-flowing. We then check summary information for both plot. The middle plot has a 0.5 speed, the left plot has a 0 speed and the right plot has 0.98 velocity.. We call the pattern shown in middle a transition phase. It follows rectangluar has a larger transition density, because from the patten of plot we may conclude the transition density of lattice is between 0.3 and 0.4.

2. Performance of code.

I have three versions of BML functions, I use Rprof to check where the bottleneck is to improve the function, the third version is the fastest. Below the three table is the output of Rprof. The grid argument is a 100 by 100 grid with 100 red cars and 100 blue cars, number of steps is 10000.

ancient vestion	self.time	self.pct	total.time	total.pct
moveCars	25.42	41.00	61.98	99.97
!=	4.40	7.10	4.40	7.10
[<-	4.06	6.55	4.06	6.55
[[.data.frame	2.96	4.77	11.02	17.77
$	2.76	4.45	17.74	28.61
match	2.50	4.03	4.12	6.65

From the output, we see the 'moveCars' took lots of time. There are there for loops in the 'MoveCars', so our first goal is to replace loop with high performance method.

second version	self.time	self.pct	total.time	total.pct
getLocations	4.06	29.68	11.50	84.06
!=	4.02	29.39	4.02	29.39
moveCars	0.64	4.68	13.66	99.85
mathc	0.34	2.49	0.82	5.99
deparse	0.28	2.05	0.86	6.29
data.frame	0.24	1.75	3.02	22.08

The second version is 6 time faster than the ancient version. However we see that 'getLocation' took lots of time. If we see our function, we find that 'getLocation' is in a big loop. So, it is reasonable to think if we can remove 'getLocation' out of loop or remove part of it out of loop.

Third version	self.time	self.pct	total.time	total.pct
[<-data.frame	0.54	14.44	2.14	57.22
moveCars	0.30	8.02	0.42	11.23
NextLocs	0.26	6.95	2.54	67.91
Ops.factor	0.24	6.42	0.68	18.18
[.data.frame	0.24	6.42	0.34	9.09
anyDuplicated	0.18	4.81	0.18	4.81

The Third version is much faster. So we choose this function in our Package.

Next, we want to see the performance of our chosen function 'BML' on different sizes of grid and different density over t time steps

First We generate a 100 by 100 grid, change density, make two types car have the same size and set time steps equal to 3000

density = 0.3	self.time	self.pct	total.time	total.pct
0ps.factor	0.88	17.6	1.38	27.6
[<-.data.frame]	0.84	16.8	2.46	49.2
moveCars	0.74	14.8	1.00	20.0
NextMethod	0.40	8.0	0.40	8.0
NextLocs	0.38	7.6	2.86	57.2
[.data.frame	0.38	7.6	0.50	10.0

density = 0.7	self.time	self.pct	total.time	total.pct
0ps.factor	2.02	20.28	3.36	33.73
[<-.data.frame]	1.50	15.06	5.16	51.81
moveCars	1.30	13.05	1.38	13.86
NextMethod	1.26	12.65	1.26	12.65
NextLocs	1.18	11.85	6.42	64.46
[.data.frame	0.86	8.63	1.26	12.65

From table we see that the density is double and the time is double. What's more, with a larger density, the time we use to subset become larger.

Then We generate a 300 by 300 grid, change density, make two type have the same size and set time steps equal to 3000 .

density = 0.3	self.time	self.pct	total.time	total.pct
0ps.factor	8.22	20.09	12.96	31.67
[<-.data.frame]	7.14	17.45	21.14	51.66
moveCars	6.74	16.47	8.24	20.14
NextMethod	4.78	11.68	4.86	11.88
NextLocs	4.34	10.61	4.34	10.61
[.data.frame	2.50	6.11	24.12	58.94

density = 0.7	self.time	self.pct	total.time	total.pct
0ps.factor	20.74	21.47	33.14	34.31
[<-.data.frame]	18.78	19.45	54.36	56.28
moveCars	11.22	11.62	11.28	11.68
NextMethod	11.20	11.60	11.20	11.60
NextLocs	8.24	8.53	63.70	65.96
[.data.frame	8.06	8.35	11.84	12.26

From table we see that the density is double and the time is double. What's more, with a larger gird which is 9 time bigger than the fist one, the time we use to subset become much longer than others. If we fix density, for example, at density equal to 0.3, 300 by 300 grid is 9 times larger than 100 by 100 grid, and it’s time is close to 9 times longer than 100 by 100. If we fixed density to 0.7, there also is such relationship. So, since the increase in dimension will cause quadratic increase in space of grid, the relation between time and dimension of gird should be a quadratic curve. Furthermore, since we take most of our time to subsetting, it is inevitable to take longer time to simulate larger grid unless we optimizing our subsetting method.

APPENDIX

```r
createBMLGrid =
  function(r=100,c=99, ncars = c(100,100))

  {
    if(length(c) == 0 & length(r)==0) stop('row or col is not given')

    if (r*c < sum(ncars)) stop('number of cars out of bound')

    Grids = matrix('',ncol = c, nrow = r)

    prod = sample(1:(r*c), sum(ncars))#prod is a index vector

    Grids[prod] = sample(rep( c('red', 'blue'), ncars))#from sample we randomly get a voctor
    #of 'red' and 'blue'. Then we give the the vector to Grids by index prod.
    #We consider Grids as a big vector concatenated by colums.

    Grids = structure(Grids,class = 'BMLGrid')#give S3 class

    Grids
  }

getLocations =
  function(g)
  {
    i = row(g)[g != ""] # g is a grid created from 'createBMLGrid'

    j = col(g)[g != ""]

    pos = cbind(i, j)

    colors = g[pos]

    p = data.frame(i=i,j=j,colors=colors)

    p
  }


BlueCarLocs = function(p,g)
{
  l = p$colors %in% 'blue' # p is output of 'getLocation' i.e a dataframe.

  rows = p[l, 1]
  cols = p[l, 2]

  nextRows = rows - 1L

  nextRows[ nextRows == 0 ] = nrow(g)

  nextCols = cols

  nextLocs_list = list(nextRow = nextRows, nextCol = nextCols,

                       rows = rows, cols = cols)

  nextLocs_list
}



RedCarLocs =
  function(p, g)
  {
    l = p$colors %in% 'red' # p is output from getLocation. It is a dataframe.

    rows = p[l, 1]# l is a logical index vector.

    cols = p[l, 2]

    nextRows = rows

    nextCols = cols + 1L # red car move rightward on step.

    nextCols[ nextCols > ncol(g) ]  = 1L#if a red car is on the right edge of grid

    # move it to the left edge.

    nextLocs_list = list(nextRow = nextRows, nextCol = nextCols,

                         rows = rows, cols = cols)

    nextLocs_list
  }



moveCars =
  function(l, g, color = 'blue')
  {
    nextLocs = cbind(l$nextRow, l$nextCol)# If color is blue, l is output of 'BlueCarLoc'

    # if color is red, l is output of 'RedCarLocs'
    w = g[nextLocs]  == "" #g is grid. w is logical vector indicate if next location is
    # a blank.

    g[ nextLocs[w, , drop = FALSE] ] = color#If next location is blank, give color to it

    # use drop to make nextLocs as a matrix.

    g[ cbind(l$rows, l$cols)[w, , drop = FALSE] ] = "" #We need to delete the color of

    #current space after we give color to next step.

    list(Grids = g, w = w)
  }


NextLocs =
  function(l, p, s, color = 'blue') # if color is blue, l is output of 'BlueCarLocs'

    #if color is red, l is output of 'RedCarLocs'. l is a list of current location and
    #next locations.
  {
    w = s$w #s is output of 'moveCars' w is logical vector where True mean empty space.

    g = s$Grids

    if(color == 'blue')
    {
      l$nextRow[!w] = l$nextRow[!w]+1L# In 'BlueCarLocs' we minus 1L to all cols. #Howeversome are bl
ocked,so we need to add 1L back.

      l$nextRow[l$nextRow > nrow(g)] = 1L

    } else {

      l$nextCol[!w] = l$nextCol[!w] -1L

      l$nextCol[l$nextCol == 0] = ncol(g)
    }

    p[p$colors == color, 'i'] = l$nextRow

    p[p$colors == color, 'j'] = l$nextCol

    p
  }
```

```
BML =
  function(g, numSteps)
  {
    cars = getLocations(g)

    for(i in 1:numSteps)
    {
      if(i%%2 == 0)
      {
        locs_list =  RedCarLocs(p=cars, g)

        g = moveCars(l=locs_list, g, color = 'red')

        cars = NextLocs(l=locs_list, p=cars,color = 'red',s = g)

      } else {

        locs_list = BlueCarLocs(p = cars, g)

        g = moveCars(l = locs_list,g, color = 'blue')

        cars = NextLocs(l = locs_list,p=cars, color = 'blue', s = g)

      }

      g = g$Grids

    }
    g
  }





plot.BMLGrid =
  function(x)
  {
    z = matrix(match(x, c('','red', 'blue')), nrow(x), ncol(x))#replace characters with

    # intergers

    image(z, col = c('white','red','blue'))

    box()
  }




summary.BMLGrid =
  function(x){

    nrows = dim(x)[1]#x is a grid with class GMLGrids

    ncols = dim(x)[2]

    nredcars = cumsum( x %in% 'red')[nrows*ncols]# X %in% 'red' return a vector.

    nbluecars = cumsum( x %in% 'blue')[nrows*ncols]

    RedBlocked = NumCarsBlocked(x, colors = 'red')#NumCarBlocked is a function

    BlueBlocked = NumCarsBlocked(x, colors = 'blue')

    list(dimention = list(nrows=nrows,ncols=ncols), numRedCars = nredcars,

         numBlueCars = nbluecars, numBlockedReds = RedBlocked,

         numBlockedBlues = BlueBlocked)

  }


NumCarsBlocked = function(x, colors = 'red'){

  cars = getLocations(x) #x is a grid, cars is a dataframe contain

  if(colors == 'red'){

    l = RedCarLocs(p = cars, g=x)

  } else {

    l = BlueCarLocs(p = cars, g=x)
  }

  nextLocs = cbind(l$nextRow, l$nextCol)

  w = x[nextLocs]  != ""
```

```r
  n = cumsum(w)[nrow(nextLocs)]

  n
}


TotalCars = function(x, color = 'red'){

  nrows = nrow(x)

  ncols = ncol(x)

  if(color =='red'){

    TotalCars = cumsum( x %in% color)[nrows*ncols]

  } else {

    TotalCars = cumsum( x %in% color)[nrows*ncols]

  }

  TotalCars

}




BML_Process = function(x,Steps, color = 'red' )
{
  g = BML(g = x, numSteps = Steps)  # x is a grid

  nb = NumCarsBlocked(g, color)

  tot = TotalCars(x = g, color)

  nm = tot - nb

  v = nm/tot

  list(NumMove = nm, NumBlocked = nb, velocity =  v)

}
```

```
velocity =
  function(x, numSteps)
  {
    sapply(1:numSteps,function(x)
    {
      if(x%%2 == 0){
        BML_Process(x = g,Steps = x, color = 'red' )

      } else {
        BML_Process(x = g,Steps = x, color = 'blue' )
      }
    })


  }
```

```r
#The Second version of function BML
BML = function(g, numSteps = 3){


  for(t in 1:numSteps){

    cars = getLocations(g)

    if(t%%2 == 0){

      w = cars$colors %in% c('red')

      rows = cars[w,1]

      cols = cars[w,2]

      nextRows = rows

      nextCols = cols + 1L

      nextCols[nextCols > ncol(g)] = 1L


    } else {

      w = cars$colors %in% c('blue')

      rows = cars[w,1]

      cols = cars[w,2]

      nextRows = rows - 1L
      nextRows[ nextRows == 0 ] = nrow(g)
      nextCols = cols

    }

    nextLocs = cbind(nextRows, nextCols)
    l = g[ nextLocs ]  == ""
    g[ nextLocs[l, , drop = FALSE] ] = as.character(cars$colors[w][1])
    g[ cbind(rows, cols)[l,, drop = FALSE] ] = ""

  }

  return(g)
}
```

```
# The first version of BML
BML =
  function(g, numSteps = 3)
    {
      i = row(g)[g != ""]

      j = col(g)[g != ""]

      pos = cbind(i, j)

      colors = g[pos]

      cars = data.frame(i=i,j=j,colors=colors)

      for(t in 1:numSteps)
        {
          if(!t%%2 == 0)
            {
              w = which(cars$colors == 'red')
                for(idx in w)
                  {
                    curPos = c(i = cars$i[ idx ], j = cars$j[idx])

                    nextPos = c(curPos[1], if(curPos[2] == ncol(g))
                                                        1L
                                                  else curPos[2] + 1L)

                   if(g[ nextPos[1], nextPos[2] ] == "")

                     {
                       g[nextPos[1], nextPos[2]] = 'red'

                       g[curPos[1], curPos[2]] = ""
                   }
                    g = g
                }

              } else {
                  w = which(cars$colors == 'blue')

                  for(idx in w)
                    {
                      curPos = c(i = cars$i[ idx ], j = cars$j[idx])

                      nextPos = c(if(curPos[1] == 1)
                                    ncol(g)
                                  else curPos[1] - 1L,curPos[2])
                      if(g[ nextPos[1], nextPos[2] ] == "")
                        {
                          g[nextPos[1], nextPos[2]] = 'blue'

                          g[curPos[1], curPos[2]] = ""
                      }
                  }

                  g = g
              }
            i = row(g)[g != ""]
          j = col(g)[g != ""]

          pos = cbind(i, j)

         colors = g[pos]

         cars = data.frame(i=i,j=j,colors=colors)

       g=g
      }
      return(g)
  }
```