March 8, 2011 CAPS 491-11 Spring 2011

In-Class Work: Traffic Flow

1. Uniform Random Numbers

1a. Copy the program
~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/float_rand0-1.cc
into your working directory. Compile the program and let it run. Scan the header and main of the program to learn how to get random numbers. Do not try to understand the function randomd itself. Random function generators are an art for itself, and we just use this random number generator because it is a very well working one. Whenever you want to write a program in which you use random numbers you need to do the following step:
1. Include in the header of your program the two lines:

```
double randomd(long *);
   long idummy = -7;
```

2. Include at the end of your program the definition of the function randomd, i.e. lines 28–58 of float_rand0-1.cc.
3. whenever you want another random number use `randomd(&idummy)`

1b. Have a look at which random numbers you get:
`float_rand0-1.out | gawk '{print NR,$1}'`
and to see the numbers graphically
`float_rand0-1.out | gawk '{print NR,$1}' | xgraph -m -nl`

2. Initialize Car Positions

2a. Use the program float_rand0-1.cc to write a program that prints (instead of random numbers) 500 numbers where each number is "1" with probability `PCAR = 0.3` and otherwise "-1". Run the program and check it with xgraph as in 1b. And here is a quick way to check how often 1 was chosen: `traffic2a.out | gawk 'BEGIN{n=0}{if($1==1) n++}END{print n}'`

2b. Use your program from 2a. and use it to initialize a `road` array of `ROADLENGTH=20`. Put on each site a car with probability `PCAR=0.3` by assigning a value 1 and otherwise keep the road site empty by assigning the value −1. Check the program by printing the road.

3. Initialize Road

3a. For the initialization of the road we also will have to initialize the velocity of each car. As preparation for this task modify your program of 2a such that it prints 500 random numbers $v \in \{0, 1, \ldots, \texttt{VMAX}\}$ (use `VMAX=5`) by using the random number generator. Hint: You may use that `int` converts double to integer, for example `int(4.67)=4` and `int(3.05)=3`.
3b. Now you are ready to initialize the road. Modify your program of 2b to put on each site of the road a car with probability `PCAR` and to give each car on the road a random velocity $v \in \{0, 1, \ldots, \texttt{VMAX}\}$ (use `VMAX=5`).

4. Distance (if time)

Please get me before you start working on the following so that I can explain the next steps.

4a. Next we work on finding the distance between a car and the car in front of it. To get this distance we will avoid in the following to have to check each site in front of a car if it is empty or not. Instead we define an additional array: `carpos` which stores for each car on the road its position on the road. So if the first car (index 0) on the road is on site 3 then `carpos[0]=3`, if the second car is on site 5 then `carpos[1]=5`, etc.. Add to your program of 3a. the array carpos and check your program by printing out both the complete road and the car positions.

4b. Now add to your program that for each car the distance to the car in front of it is determined and printed out. Take into accout the periodic boundary conditions both to determine the car in front and to determine the distance.
Comment: This task is more difficult than it might seem at first glance. Please get me after having thought about this task.

In-Class Work: Traffic Flow

4. Distance

4a. Use your running program for 3b or copy the solution program `~kvollmay/classes.dir/capstone_s2011.dir/traffic3b.cc` Next we work on finding the distance between a car and the car in front of it. To get this distance we will avoid in the following to have to check each site in front of a car if it is empty or not. Instead we define an additional array: `carpos` which stores for each car on the road its position on the road. So if the first car (index 0) on the road is on site 3 then `carpos[0]`=3, if the second car is on site 5 then `carpos[1]`=5, etc.. Add to your program of 3a. the array carpos and check your program by printing out both the complete road and the car positions.

4b. Now add to your program that for each car the distance to the car in front of it is determined and printed out. Take into accout the periodic boundary conditions both to determine the car in front and to determine the distance.
Comment: This task is more difficult than it might seem at first glance. Please get me after having thought about this task.

5. Update Velocities

5a. For the update of the velocities we will need a function which determines the smallest number of three integers. Write a program which reads in three integers, determines via a function which of these three integers the smallest number is, and prints out the result. Use a function, so that we can use the same function in our traffic flow program. Test your program by printing the minimum of each of the three sets: `(3,2,10),(1,13,10),(16,14,10)`.

5b. Use your program of 4b and add to it that it determines all new velocities (and updates them in the `road` array), and prints out the road with the new velocities. Compare your result for the case
(idummy=-7,ROADLENGTH = 20,VMAX = 5, PCAR = 0.3) with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic5b.data`

6. Update Positions (if time)

Now you are ready to add to your program the update of the positions. To do so use a new loop over all cars and update both the carpos array and the road array. For the road make sure to first copy the new velocity into a variable (e.g. `vnew`), then empty the old site and then put the car in road on its new site `xnew = xold + vnew` with the new velocity. After the update of all positions print the complete road //(already in traffic4b.cc) and check if it is what you expected. Using the same parameters as given in 5b compare your result with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic6.data`
Test your program further by using PCAR=0.5 and compare again with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic6.data`

7. Finish Program (if time)

Now you are set to finish the program for our traffic flow model. Add to your program from 6. the time loop. Compare your result for the case of 100 timesteps and otherwise the same parameters as in 5b. Compare your output with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic7.data`

March 29, 8am: Running Program
Homework for March 22: Continue working on your program. (Writing the program will take several days, so work on the programming continuously.)

In-Class Work: Traffic Flow

6. Update Positions

Copy the solution program for the velocity update
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic5.cc`
or if you had finished already 5. of last class, you may use your working program from last class. Next add to the program the update of the positions. To do so use a new loop over all cars and update both the `carpos` array and the `road` array. For the road make sure to first save the new velocity, then empty the old site and then put the car in `road` on its new site `xnew = xold + vnew` with the new velocity. After the update of all positions print the complete road and check if it is what you expected. Use the same parameters as in `traffic5.cc`, namely `idummy=-7,ROADLENGTH = 20,VMAX = 5, PCAR = 0.3` and compare with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic6.data`
Test your program further by using PCAR=0.5 and compare again with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic6.data`

7. Finish Program

Now you are set to finish the program for our traffic flow model. Add to your program from 6. the time loop. Compare your result for the case of 100 timesteps and otherwise the same parameters as in 6 with `PCAR=0.3`. Compare your output with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic7.data`

8. Space-Time Diagrams

8a. Now we are ready to have a look at the flow of the cars. To be able to see the main patterns (and to get nice pictures) use a larger ROADLENGTH (for example = 200). We use DynamicLattice to make a graph of time over position. Print the road for every time step, as it was done to get `traffic7.data`. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

```
traffic8.out | DynamicLattice -nx 200 -ny 101 -matrix
```

8b. To distinguish stopped cars easily from driving cars, let's indicate every empty site instead of with -1 now with -VMAX. This results in DynamicLattice in empty sites being blue, stopped cars in white and driving cars in pinkish/red. Look at the space-time diagram.

8c. Vary PCAR. Interpret the resulting space-time diagrams. Once you have several space-time diagrams for different PCAR, get me so that we can discuss as a class your results.

9. Nagel-Schreckenberg Model

9a. Get me when you get to this part. We are now ready to finish the programming of the Nagel-Schreckenberg model. Add to your program of 8. the randomization of the velocity, so complete the Nagel-Schreckenberg Model. Use `VMAX = 5, PCAR = 0.2, PDEC = 0.25, ROADLENGTH = 20, MAXTSTEPS=8` and `idummy = -7` and compare your result with
`~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic9a.data`

Comment: There are variations in how you might use the random numbers. Therefore, please get me in case of discrepancies between your and my results.

9b. Now set `ROADLENGTH=200` and `MAXSTEPS=100` and have a look at the resulting space-time diagram with DynamicLattice.

9c. Keep all parameters as in 9b but vary
(i) `PCAR` between 0.05 and 0.35
(ii) `PDEC` between 0.0 and 0.5
(iii) `VMAX` between 1 and 10.
What do you observe in each case? Please get me to discuss your observations and to share your results with the class.

10. Mean Velocity $v_{\mathbf{av}}(t)$ (if time)

Use your program from 9. with parameters (`idummy=-7, VMAX=5, PCAR=0.2, PDEC=0.25, ROADLENGTH=1000, and MAXTSTEPS=200`) and instead of printing out the road print on the screen for each time `t` one line with two numbers: `t` and v_{av} where v_{av} is the mean velocity:
$v_{\text{av}} = \frac{1}{N} \sum\limits_{i=0}^{N-1} v_i$
is the number of cars and v_i is the velocity of car i. Compare your result with `~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic10.data` Look at $v_{\text{av}}(t)$ with:
```
executable | xgraph -m
```
Get me so that we can discuss your result. Our interpretation is necessary for 11.

11. $\mathbf{v_{eq}}(c)$ (if time)

For simplicity let us go back to the model without randomized velocities, so use PDEC = 0.0. Set also VMAX = 4 and ROADLENGTH = 1000. You saw in 10. that v_{av} equilibrates after some time to some value v_{eq} around which v_{av} fluctuates. We now want to see how v_{eq} depends on the concentration of cars $c =$`nocars/double(ROADLENGTH)`.

11 a. Take out of your program from 10. the printing of v_{av}. Add to the program that you measure v_{av} only if the time t is larger than EQUILSTEPS = 100. Average over these measured v_{av} which gives you v_{eq}. Print out the concentration c and v_{eq}. With your program of 10. (adjust PDEC & VMAX) check if your result of v_{eq} seems plausible.

11 b. Now change in your program to use `PCAR` no longer as constant but instead add a loop over pcar (0.1 - 1.0) and print out for each pcar the concentration c and v_{eq} as you did in 11a. Have a look at your result with xgraph
```
executable | xgraph -m -nl
```
Get me, so that we can discuss the result.

11 c. Set PDEC=0.0 and run your program first for VMAX=4 and then for VMAX=2. In each case redirect the output into a file:
`executable > filename` Then set PDC=0.25 and run and redirect the output again for VMAX=2 and 4. Look at the data of the four files with xgraph.

Homework for March 29, 8am: Running Main Project Program (in /share.dir/ and readpermission)

This Thursday, March 24 class cancelled
Make-Up Class March 29, 8-9:30am

In-Class Work: Traffic Flow

9. Nagel-Schreckenberg Model

9b. Copy the solution program for the complete Nagel-Schreckenberg Model `~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic9.cc`
or in case you had finished successfully the in-class work 9a. , then you may use your own program. Familiarize yourself with the program and have a look at the resulting space-time diagram with

```
traffic9.out | DynamicLattice -nx 200 -ny 101 -matrix
```

9c. Keep all parameters as in 9b but vary
(i) `PCAR` between 0.05 and 0.35
(ii) `PDEC` between 0.0 and 0.5
(iii) `VMAX` between 1 and 10.
What do you observe in each case? Interpret your results. Once you have looked at each case (i)–(iii) please get me to discuss your observations and to share your interpretations with the class. I will then also give you an introduction to the next step 10.

10. Mean Velocity $v_{\mathbf{av}}(t)$

Use your program from 9. with parameters (`idummy=-7, VMAX=5, PCAR=0.2, PDEC=0.25, ROADLENGTH=1000, and MAXTSTEPS=200`) and instead of printing out the road print on the screen for each time `t` one line with two numbers: `t` and v_{av} where v_{av} is the mean velocity:
$v_{\text{av}} = \frac{1}{N} \sum_{i=0}^{N-1} v_i$
is the number of cars and v_i is the velocity of car i. Compare your result with `~kvollmay/classes.dir/capstone_s2011.dir/traffic.dir/traffic10.data` Look at $v_{\text{av}}(t)$ with:

```
executable | xgraph -m
```

Get me so that we can discuss your result. Our interpretation is necessary for 11.

11. $\mathbf{v}_{\mathbf{eq}}(c)$

For simplicity let us go back to the model without randomized velocities, so use PDEC = 0.0. Set also VMAX = 4 and ROADLENGTH = 1000. You saw in 10. that v_{av} equilibrates after some time to some value v_{eq} around which v_{av} fluctuates. We now want to see how v_{eq} depends on the concentration of cars $c =$`nocars/double(ROADLENGTH)`.

11 a. Take out of your program from 10. the printing of v_{av}. Add to the program that you measure v_{av} only if the time t is larger than EQUILSTEPS = 100. Average over these measured v_{av} which gives you v_{eq}. Print out the concentration c and v_{eq}. With your program of 10. (adjust PDEC & VMAX) check if your result of v_{eq} seems plausible.

11 b. Now change in your program to use `PCAR` no longer as constant but instead add a loop over pcar (0.1 - 1.0) and print out for each pcar the concentration c and v_{eq} as you did in 11a. Have a look at your result with xgraph

```
executable | xgraph -m -nl
```

Get me, so that we can discuss the result.

11 c. Set PDEC=0.0 and run your program first for VMAX=4 and then for VMAX=2. In each case redirect the output into a file:
`executable > filename` Then set PDC=0.25 and run and redirect the output again for VMAX=2 and 4. Look at the data of the four files with xgraph.

12. Flux (IF TIME) Get me before you work on this. Next add to your program the flux $j = c \star v_{\text{eq}}$ so that it prints out c, v_{eq}, and the flux j. Redirect your output into a file and have a look at your result for $j(c)$ with `gawk '{print $1,$3}' filename | xgraph -m -nl`. Use the same parameters as in 11 c. Let's look together at your result.