Announcement: Main Paper due next class, T, Febr.26, 9:30am. Please hand in

- hard-copy of paper
- paper as email (psfile, word-document or pdf)

In-Class Work: Traffic Flow (Newcomers)

1. Uniform Random Numbers

1a. Copy the program
~kvollmay/sunhome/classes.dir/capstone_s2008.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 29–58 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: To do so set up the following unix-command:
`executable | gawk '{print NR,$1}' | xgraph -m -nl`

2. Initialize Car Positions

2a. Write a program that prints 500 numbers. Each number is with probability `PCAR = 0.3` "1" and otherwise "-1". Run the program and check it with xgraph as in 1b.

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` by assigning a value `1` and otherwise keep the road site empty by assigning the value `-1`.

3. Initialize Road

For the initialization of the road we next pick for each car on the road a random velocity $v \in \{0, 1, \ldots,$ `VMAX` $\}$. To get these random velocities use `int`, for example `int(4.67)=4` and `int(3.05)=3` and `int(v)` makes an integer out of a double `v`. Test your program by printing out the road.

In-Class Work: Traffic Flow (Advanced)

1. – 3. Initialize Road Work on in-class work 1.– 3. on the backpage.

4. Distance

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 3. 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.

In-Class Work: Traffic Flow (Newcomers)

5. Update Velocities

Use the program
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic5_sample.cc
and add to it that it determines all new velocities and updates them in the `road` array. See comments in traffic5_sample.cc where you need to add lines. Keep as is
(idummy=-7,ROADLENGTH = 20,VMAX = 5, PCAR = 0.3) and for checking your program compare your output with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic5b_data.cc

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 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 (already in traffic4b.cc) and check if it is what you expected. Compare your result with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic6_data.cc

In-Class Work: Traffic Flow (Advanced)

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.

5b. Use your program of 4b or the program
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic4b.cc
Add to traffic4b.cc 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/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic5b_data.cc

6. Update Positions

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 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 (already in traffic4b.cc) and check if it is what you expected. Compare your result with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic6_data.cc

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 5b. Compare your output with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic7_data.cc

8. Space-Time Diagrams

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). Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. Then use DynamicLattice to make a graph of time over position. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

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

Vary PCAR. Interpret the resulting space-time diagrams.

In-Class Work: Traffic Flow (Newcomers)

6. Update Positions

Use your program from last class for in-class work 5. (velocity update) or
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic5.cc
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 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 (already in traffic4b.cc) and check if it is what you expected. Compare your result with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic6_data.cc

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 (idummy=-7,ROADLENGTH = 20,VMAX = 5, PCAR = 0.3, MAXTSTEPS=100) Compare your output with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic7_data.cc

8. Space-Time Diagrams (if time)

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). Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. Then use DynamicLattice to make a graph of time over position. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

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

Vary PCAR. Interpret the resulting space-time diagrams. Please get me when you get here, so that we can share your results with the class.

In-Class Work: Traffic Flow (Advanced)

7. Finish Program

Use your program from the in-class work 6. or use
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic6.cc
Now you are set to finish the program for our traffic flow model. Add to your program the time loop. Compare your result for the case (idummy=-7,ROADLENGTH = 20,VMAX = 5, PCAR = 0.3, MAXTSTEPS=100) Compare your output with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic7_data

8. Space-Time Diagrams

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). Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. Then use DynamicLattice to make a graph of time over position. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

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

Vary PCAR. Interpret the resulting space-time diagrams. Please get me when you get here, so that we can share your results with the class.

9. Nagel-Schreckenberg Model

9a. Before you work on this task, get me. I will discuss with you our next model: the Nagel-Schreckenberg model.
Add to your program of 7. the randomization of the velocity, so complete the Nagel-Schreckenberg Model. Use `VMAX = 5, PCAR = 0.2, PDEC = 0.25, ROADLENGTH = 20, MAXTSTEPS=100` and `idummy = -7` and compare your result with ~ kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic9_data

9b. Now increase your `ROADLENGTH` to 200 and have a look at the space-time diagram with DynamicLattice.

9c. Keep all parameters as in 8b 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 $\langle v\rangle(t)$ (if time)

Use your program from 10. with parameters (`idummy=-7, VMAX=5, PCAR=0.2, PDEC=0.25, ROADLENGTH=1000, and TSTEPS=200`) and instead of printing out the road print on the screen for each time `t` one line with two numbers: `t` and $\langle v\rangle$ where $\langle v\rangle$ is the mean velocity:
$\langle v\rangle = \frac{\sum_{i=0}^{N-1} v_i}{N}$ where N is the number of cars. Compare your result with ~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic10_data
Look at $\langle v\rangle(t)$ with:

```
executable | xgraph -m
```

Let's have together a look at your result.

In-Class Work: Traffic Flow (Advanced)

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.

5b. Use your program of 4b or the program
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic4b.cc
Add to traffic4b.cc 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/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic5b_data.cc

6. Update Positions

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 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 (already in traffic4b.cc) and check if it is what you expected. Compare your result with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic6_data.cc

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 5b. Compare your output with
~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic7_data.cc

8. Space-Time Diagrams

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). Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. Then use DynamicLattice to make a graph of time over position. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

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

Vary PCAR. Interpret the resulting space-time diagrams.

In-Class Work: Traffic Flow (Newcomers)

8. Space-Time Diagrams

Use your program (if you had the same result as `traffic7_data`) or the solution program ~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic7.cc
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). Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. Then use DynamicLattice to make a graph of time over position. In case you run your program for 100 timesteps, you get 101 lines, so your DynamicLattice command is for example:

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

Vary PCAR. Interpret the resulting space-time diagrams. Please get me when you get here, so that we can share your results with the class.

9. Nagel-Schreckenberg Model

9a. Before you work on this task, get me. I will discuss with you our next model: the Nagel-Schreckenberg model.
Add to your program of 7. the randomization of the velocity, so complete the Nagel-Schreckenberg Model. Use `VMAX = 5, PCAR = 0.2, PDEC = 0.25, ROADLENGTH = 20, MAXTSTEPS=100` and `idummy = -7` and compare your result with ~ kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic9_data

9b. Now increase your `ROADLENGTH` to 200 and have a look at the space-time diagram with DynamicLattice.

9c. Keep all parameters as in 8b 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.

In-Class Work: Traffic Flow (Advanced)

9. Nagel-Schreckenberg Model

See back side of this page.

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

Use your program from 10. with parameters (`idummy=-7, VMAX=5, PCAR=0.2, PDEC=0.25, ROADLENGTH=1000, and TSTEPS=200`) and instead of printing out the road print on the screen for each time `t` one line with two numbers: `t` and $v_{\rm av}$ where $v_{\rm av}$ is the mean velocity:
$v_{\rm av} = \frac{1}{N}\sum_{i=0}^{N-1} v_i$ where N is the number of cars and v_i is the velocity of car i. Compare your result with ~kvollmay/sunhome/classes.dir/capstone_s2008.dir/traffic.dir/traffic10_data
Look at $v_{\rm av}(t)$ with:
`executable | xgraph -m`
Get me so that we can discuss your result. Our interpretation is necessary for 11.

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

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 $v_{\rm av}$ equilibrates after some time to some value $v_{\rm eq}$ around which $v_{\rm av}$ fluctuates. We now want to see how $v_{\rm eq}$ depends on the concentration of cars c =`nocars/double(ROADLENGTH)`.

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

Get me before you work on this. Next add to your program the flux $j = c \star v_{\rm eq}$ so that it prints out $c, v_{\rm eq}$, 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.