RANDOM NUMBERS

1. Uniform Random Numbers

1a. Copy the program
~ kvollmay/classes.dir/capstone_s2003.dir/game.dir/float_rand0-1.cc
into your working directory. Compile the program and let it run. Have a look at the program to learn how to get random numbers. 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 21–49 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 | awk '{print NR,$0}' | xgraph -m -nl

2. Probability

2a. Write a program that prints 500 numbers. Each number is with probability \( p = 0.3 \) “1” and otherwise “0”. Run the program and check it with xgraph as in 1.

2b. Change your program from 2a. such that it reads in from screen the probability \( p \) and the number of random numbers \( N \). The program then produces \( N \) numbers, where each number is with probability \( p \) “1” and otherwise “0”. The program counts how many of the drawn numbers are “1” \( (n_{\text{alive}}) \) and prints out \( (n_{\text{alive}})/N \). Run your program a few times for different values of \( p \) and \( N \).

3. Integer Random Numbers

Use the random numbers from 1. to print out 500 random numbers which are with equal likelihood \( v = 0, 1, 2, 3, \ldots, 9 \) or 10. (We will use this method for the traffic model to draw velocities for cars on the road.) Check your program with xgraph as you did in 1b.
Once your program is working generalize 10 to the constant \( \text{VMAX}=10 \) which you define at the beginning of your program.
RANDOM NUMBERS AND GAME OF LIFE (ADVANCED)

1. Work on 1a,b. and 2a,b on the backside of this page.

2c Write a program which allows you to make a graph of \((n_{\text{alive}})/N\) as a function of \(N\). Keep \(p = 0.3\) constant. So your program should print out two columns, where the first is \(N\) and the second is \((n_{\text{alive}})/N\). (Vary \(N\) between 0 and 50000.) Look at your result with executable | xgraph -m

Look at the result for different values of \(p\).

3. Work on 3. on the backside of this page.

4. Game of Life: Density As Function of Time

4a. Use your program for the game of life (5a.) Write a function which initializes your lattice as follows: Each cell is alive with a probability \(p_{\text{alive}}\). Initialize your lattice by calling this function for a specific \(p_{\text{alive}}\), e.g \(p_{\text{alive}} = 0.5\). Test your program with a 5x5 lattice by printing out only the initial configuration.

4b. Use your program for 4a. with \(p_{\text{alive}} = 0.4\) and now for 2000 timesteps and a 100x100 lattice. Change your program such that for each time step \(t\) the program prints out \(t\) and \((\text{number of alive cells})/(\text{total number of cells})= n_{\text{alive}}/N\). Plot the resulting \(n_{\text{alive}}/N\) as a function of the time \(t\).

4c. Print out \(t\) and \(n_{\text{alive}}/N\) only every 20 timesteps and run your program for 3000 timesteps. Vary i) the lattice size ii) \(p_{\text{alive}}\) and iii) the neighborhood. Draw the different curves for i) all in one graph. To do so redirect your output into files, for example

    executable > lat100p04

and look at the different files with for example

    xgraph -m lat*p04

Do the same for ii) and iii).

5. Equilibrated \(n_{\text{alive}}/N\) As Function of \(p_{\text{alive}}\)

Use a 100x100 lattice, 1000 timesteps and the von Neumann neighborhood. You saw in 4. that the density \(n_{\text{alive}}/N\) after a certain number of timesteps no longer changes. Add to your program that you loop over \(p_{\text{alive}}\) and that your program prints out \(p_{\text{alive}}\) and the density only for the lattice after the last timestep. You obtain the density as a function of \(p_{\text{alive}}\). Look at your result with xgraph.
Traffic Flow

1. Integer Random Numbers

Copy the program
```
~ kvollmay/classes.dir/capstone_s2003.dir/game_of_life.dir/rand2a.cc
```
into your working directory. This program prints on screen 500 random numbers, which are either “1” (with probability 0.3) or “0”. This program is a slight variation on
```
~ kvollmay/classes.dir/capstone_s2003.dir/game_of_life.dir/float_rand0-1.cc
```
Notice that only the main function is changed but not the function randomd. Similarly copy rand2a.cc into another file and change main such that your program prints out 500 random numbers which are with equal likelihood \( v = 0, 1, 2, 3, \ldots, 9 \) or 10. In 2. we will use this method for the traffic model to draw velocities for cars on the road. Check your program with xgraph:
```
executable | awk '{print NR,$0}' | xgraph -m -nl
```
Once your program is working generalize 10 to the constant \( V_{\text{MAX}}=10 \) which you define at the beginning of your program.

2. Initialize Road

Now you are ready to start with the traffic model. Define an array for the road (begin with road length = 20) and put cars on the road with probability \( p_{\text{car}} = 0.3 \) and with \( v_{\text{max}} = 5 \). Test your program by printing out the road.

3. Distance

3a. 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: \( \text{carpos} \) 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 \( \text{carpos}[0]=1 \), if the second car is on site 5 then \( \text{carpos}[1]=5 \), etc.. Add to your program of 2. the array carpos and check your program by printing out both the complete road and the car positions.

3b 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 account the periodic boundary conditions both to determine the car in front and to determine the distance.
Traffic Flow

3. Distance

The flow chart shows that for your update of the velocities you need to determine for each car the distance to the car in front of it. This is the task we focus on today.

3a. 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.. Use your solution to 2. or the program

~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic2.cc

Add to the program the array carpos and check your program by printing out both the complete road and the car positions.

3b. 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 account the periodic boundary conditions twice:
1. to determine the car in front (the car i at carpos[i] has which car in front of it and at which position?) and
2. to determine the distance (once you know the two positions of the car and the car in front of it, how do you get the distance?).
4. Update Velocities

4a. 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.

4b. Use your program of 3b or the program traffic3b.cc. (I will send the solution as attachment to everybody in class who worked on 3b for the daily assignment. Please let me know in case you need a copy of the solution.) Together with your solution to 4a. you are now ready to write a program which initializes the road, prints out this initial configuration, 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_s2003.dir/traffic.dir/traffic4b_data
```

In case your initial road is not identical to the first line of traffic4b_data, read the initial road in from

```
~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic4binput_data
```

Remember to both initialize the road array and the carpos array.

5. Update Positions

Add to your program to update the positions of the cars. Print the road after both the velocities and positions have been updated. Remember to include the periodic boundary conditions when you determine the new position. Also update both the carpos array and the road. Make sure to both empty the site of the previous car position on the road and to fill the new position on the road. Compare your result with

```
~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic5_data
```

6. Finish Program

Now you are set to finish the program for our traffic flow model. Add to your program from 5. the time loop. Compare your result for the case of 100 timesteps and otherwise the same parameters as in 4b. Compare your output with

```
~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic6_data
```

7. 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.
TRAFFIC FLOW

5. Update Positions

Your solution for 3b. tells you how to find the distance of a car and the car in front.

`~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic4b.cc` (lines 68-71) shows you how to update the velocities of all cars. Copy traffic4b.cc into your working directory. Our focus of today’s class is the update of the positions. To add to traffic4b.cc the update of the positions 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 \( x_{\text{new}} = x_{\text{old}} + v_{\text{new}} \) 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/classes.dir/capstone_s2003.dir/traffic.dir/traffic5_data`

6. Finish Program

Now you are set to finish the program for our traffic flow model. Add to your program from 5. the time loop. To know where to put the time loop use our flow chart. Compare your result for the case of 100 timesteps and otherwise the same parameters as in 4b. Compare your output with `~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic6_data`

Congratulations you wrote your own very advanced traffic flow program!
Traffic Flow (Advanced)

Today we are set to pick the fruits of our work on the traffic model! Have fun!

6. Finish Program

Use your program of 5. or traffic5.cc (I will send you the program as email). Now you are set to finish the program for our traffic flow model. Add to your program from 5. the time loop. (Use our flow chart to know where to put the time loop.) Compare your result for the case of 100 timesteps and otherwise the same parameters as in 4b. Compare your output with ~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic6.data

Congratulations you wrote your own very advanced traffic flow program!

7. Space-Time Diagrams

Now we are ready to have a look at the motion of the cars. To be able to see the main patterns (and to get nice pictures) use a larger ROADLENGTH (for example = 200). Print at each time step t=0,1,... the road in one line. Use for each empty site instead of -1 now -VMAX so that the color scheme of DynamicLattice works better for us. (This way the empty sites are blue, v=0 is white and v=VMAX is red.) 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. Get me, so that we can have a look at your results together.

8. Nagel-Schreckenberg Model

8a. 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, 100 timesteps and idummy = -7 and compare your result with ~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic8.data

8b. Now increase your ROADLENGTH to 200 and have a look at the space-time diagram with DynamicLattice.

8c. Keep all parameters as in 8b but vary
   i) PCAR between 0.05 and 0.3 ii) PDEC between 0.0 and 0.5 iii) VMAX between 1 and 10. What do you observe in each case?

9. Mean Velocity

Use your program from 8b. (so same parameters as in 8a but ROADLENGTH=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/classes.dir/capstone_s2003.dir/traffic.dir/traffic9.data

Then increase your roadlength to 1000 and use 200 timesteps. Look at $\langle v \rangle(t)$ with:

executable | xgraph -m

Let’s have together a look at your result.
Traffic Flow

Today we are set to pick the fruits of our work on the traffic model! Have fun!

7. Space-Time Diagrams

Copy `~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic6.cc` into your working directory. Compile the program and run it to see what it does.

Next let us have a look at the motion of the cars. To be able to see the main patterns (and to get nice pictures) increase `ROADLENGTH` to `ROADLENGTH=200` and the number of timesteps to `TSTEPS=100`. Use for each empty site instead of `-1` now `-VMAX` so that the color scheme of `DynamicLattice` works better for us. (This way the empty sites are blue, \(v=0\) is white and \(v=VMAX\) is red.) Then use `DynamicLattice` to make a graph of time over position ("space-time diagram").

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

Vary `pcar`. Interpret the resulting space-time diagrams. Get me, so that we can have a look at your results together.

8. Nagel-Schreckenberg Model

8a. 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`, `100` timesteps and `idummy = -7` and compare your result with `~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic8.data`

8b. Now increase your `ROADLENGTH` to `200` and have a look at the space-time diagram with `DynamicLattice`.

8c. Keep all parameters as in 8b but vary
   i) `PCAR` between 0.05 and 0.3 ii) `PDEC` between 0.0 and 0.5 iii) `VMAX` between 1 and 10. What do you observe in each case?
9. Mean Velocity

9a. $v_{av}(t)$

Use your program for 8. from last class or traffic8.cc (which I will send you as email). Use $i_{dummy}=-7$, $VMAX=5$, $PCAR=0.2$, $PDEC=0.25$, $ROADLENGTH=1000$, and $TSTEPS=200$. Instead of printing out the road print on the screen for each time $t$ one line with two numbers: $t$ and $hv$ where

$$hv = \frac{\sum_{i=0}^{N-1} v_i}{N}$$

where $N$ is the number of cars. Compare your result with ~ kvollmay/classes.dir/capstone_s2003.dir/traffic.dir/traffic9_data

Then increase your roadlength to 1000 and use 200 timesteps. Look at $hv(t)$ with:

```
executable | xgraph -m
```

Get me so that we can discuss your result. Our interpretation is necessary for 9b.

9b. $v_{eq}(c)$

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 9a. 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 $c$.

9bi. Take out of your program from 9a. 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 and $v_{eq}$. With your program of 9a. check if your result of $v_{eq}$ seems plausible.

9bii. 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 and $v_{eq}$ as you did in 9bi. Have a look at your result with xgraph

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

Get me, so that we can discuss the result.

9biii. 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.

10 Flux

Next add to your program the flux so that it prints out $c,v_{eq}$, and the flux. Use that the flux $j=c*v_{eq}$. Redirect your output into a file and have a look at your result for $j(c)$ with

```
awk '{print $1,$3}' filename | xgraph -m -nl
```

Use the same parameters as in 9biii. Let’s look together at your result.