Assignment 4.

February 15.
Objective: See it with your own eyes.

1. Write a procedure that simulates a random walk with parameters $x = X(0)$, L, a, b, and c. One call to this procedure should produce one path and report τ and $X(\tau)$, which can be 0 or L. If you have a uniform random number generator[1] `U = rand()`, you can simulate a random walk by: $x \to x - 1$ if $U \leq c$, $x \to x$ if $c \leq U \leq c + b$, and $x \to x + 1$ otherwise. You can estimate the probability that $X(\tau) = 0$ by doing N simulations, letting M be the number of simulations with $X(\tau) = 0$ and using M/N as the estimate of the probability. In the same way, you can estimate $E[\tau]$ by averaging the τ values from N simulations.

   (a) Let $w(x) = P(X(\tau) = 0 \mid X(0) = x)$. With $L = 20$, $a = b = c = \frac{1}{3}$, estimate the numbers $w(x)$ for each $x \in [1, L-1]$. Plot the estimates and the theoretical value on the same plot. Use an N value that is big enough to give good agreement.

   (b) Repeat part (a) with $L = 20$, $a = \frac{1}{2}$, and $b = c = \frac{1}{4}$. You will have to complete the theoretical calculation of $w(x)$ in the notes. Comment on the difference in the shapes of the curves here and in part (a).

   (c) Estimate $f(x) = E[\tau \mid X(0) = x]$ with $L = 50$, and $a = b = c = \frac{1}{3}$. Plot the estimates and the theoretical values on the same plot.

   (d) Repeat part (c) with $a = \frac{1}{2}$, and $b = c = \frac{1}{4}$. Can you explain the rough shape of the result using the idea of drift as a constant speed? Compare the size of f here and in part (c). Which is larger and why?

2. Write a program to solve the forward equation for a random walk. Use absorbing boundary conditions at $k = 0$ and $k = L$. Take $L = 200$. Take $a = b = c = \frac{1}{3}$. Start with initial conditions $u_0(k)$ and solve the forward equation up to time $T = 1000$. Take $u_0(k)$ corresponding to $X(0) = 100$.

   (a) Compute and plot $H(t) = 1 - \sum_{k=1}^{L-1} u(k,t)$. This is the probability of hitting the boundary before time t. How likely is this with the present parameters? Note that the random walk takes about $\frac{2}{3}T = 667$ jumps (two thirds of the time steps involve $X \to X + 1$ or $X \to X - 1$), and it takes 100 jumps to get from the starting point to the boundary.

   (b) Make a plot of $u(k,T)$ as a function of k. How likely is it for $X(T)$ to be close to the boundary?

[1] This is how it looks in C/C++. It is slightly different in Matlab or VBA or R.

(c) With the same initial condition as part (a), compute and plot

$$S(t) = \sum_{k=1}^{L} \left(u(k,t) - u(k-1,t)\right)^2 \ .$$

If you feel ambitious, look for a *power law* $S(t) \approx C \cdot t^{-p}$ (for large t), possibly by plotting $\log S$.

(d) Compute $R(t) = \sum_{t'<t} S(t')$ and plot this to see that $\lim_{t\to\infty} R(t) = \sum_{t=0}^{\infty} S(t)$ exists and is finite. If you estimated a power law in part (c), this will be consistent.

You may program in any language you want. Below is what some of the code might look like in C/C++. If you program in Matlab, remember that arrays start with index 1, not index 0. What is called `u[k]` in C/C++ would be called `u(k+1)` in Matlab. The trick for absorbing boundary conditions ($u(0,t) = u(L,t) = 0$) is to set $u(0) = u(L) = 0$ at the beginning and never change these values. The loops run from $k = 1$ to $k = L-1$ ($k = 2$ to $k = L$ in Matlab) to avoid changing $u(0)$ or $u(L)$. Also, we use two one dimsional arrays `u` and `uNew` instead of a two dimensional array u. This uses much less computer memory. In C/C++, the main loop could be

```
double u[L+1], uNew[L+1];

 // create initial conditions

u[0] = 0; uNew[0] = 0; u[L] = 0; uNew[L] = 0; // Boundary values never change
for ( t = 0; t < T; t++ ) {  //   The time step loop
   for ( k = 1; k < L-1; k++ )   // Compute the new u values.
      uNew[k] = a*u[k-1] + b*u[k] + c*u[k+1];
   for ( k = 1; k < L-1; k++ )   // Copy them back to the old array.
      u[k] = uNew[k];
 }
```