handed out: May 12, 2020
handing in: May 21, 2020
presentation/discussion: May 22, 2020

1. **My Own Discrete Dynamical System** present ☐

   Construct and explore a discrete dynamical system of your choice.

   (a) Choose a generating function for an interesting discrete dynamical system.
   What properties are required besides $f: \Omega \mapsto \Omega$, where Ω is the domain of generator f? Possible aspects: smooth? symmetric? maximum in the interior of Ω? multiple maxima? Suggested choice for a start: smooth function on $[0, 1]$ with $f(0) = f(1) = 0$, $\max_u f(u) = 1$, and a single, strongly non-symmetric maximum.
   To be very specific (and not very elegant):

   $$f(u) = \frac{\mu}{0.399743}[1-u]^2 \sin(\pi u) \ , \quad u \in [0,1] \ , \quad \mu \in [0,1] \ .$$

   (b) Draw a cobweb diagram. Do not spend much time on coding, may do this by hand to just get an intuition for the system and for its regimes.

   (c) Calculate and draw the bifurcation diagram as done in exercise 2.2.

   (d) Zoom into interesting regions. Compare your findings with those for the logistic map.

   (e) Plot some exemplary higher iterates to understand the windows and transitions in analogy to Figures 3.4, 3.5, and 3.16 for the logistic map. [Check the bifurcation diagram for orientation.]

2. **Spectrum of Logistic Map** present ☐

   Study the power spectrum of the sequence $\mathtt{u}(u_0) = \{u_i, i \in [1, n]\}$ of the logistic map (3.7), i.e., $|\mathcal{F}\mathtt{u}|^2$, where $\mathcal{F}$ is the discrete Fourier transform (DFT). This transform assumes that the sequence is periodic, i.e., that $u_1 = u_n$.

   For the choice of n the following conditions must be considered: (i) the spectral definition gets sharper as n gets larger, (ii) the DFT is most efficient for n a product of powers of small prime numbers like $n = 2^p$, with $p \in \mathbb{N}$, or $n = 2^p \cdot 3^q$.

   As always, before running any simulation, ask yourself what you expect.

   Python users: Go with `rfft` *from* `numpy`. *Look at the manual if you are not familiar with the function.*

   (a) Focus on asymptotic regimes by first spinning-up, i.e., iterate the function m times, say $m = 10^4$, discard the values, and generate $\mathtt{u}$ starting from the last state.

      i. For $\mu \in \{3.3, 3.5, 3.56, 3.57, 3.826, 3.83, 3.8494\}$ identify the corresponding regimes in Figure 3.9, and generate $\mathtt{u}$ with $n = 2^{12} = 4'096$ states. You may want to choose n larger to get sharper peaks, but for the time being stick to powers of 2.

      ii. Calculate the spectra $|\mathcal{F}\mathtt{u}|^2$, plot them (collecting different cases as appropriate), and discuss them. Decide on linear or logarithmic scaling of the axes depending on your focus.)

      iii. For $\mu = 3.83$, generate a larger sequence with $3n$, members, n from above. Calculate the spectrum and compare it to the one obtained for the sequence with n members. Discuss.

   (b) Focus on transient regimes and choose μ and u_0 such that you expect a long transition phase, e.g., $\mu = 3$ (why at this value?). Notice that these sequences by construction will be non-periodic, at least contain an initial non-periodic phase. Whether this is significant, hence the interpretation of the power spectrum is more difficult, depends on the fraction of states in the transient regime.

3. **Transition Plot** present ☐

   For some sufficiently large values of μ, the logistic map (3.7) produces a series of apparently random numbers. It indeed was used as one of the early random-number generators. Produce two sets of such random series with range $[0, 1]$ with, say, 1'000 members each by (i) using the random number generator of your system and (ii) iterating (3.7) with $\mu = 4$, starting from an arbitrary initial value.

   (a) Plot the two series as ordered sequences. Do you notice any differences?

   (b) Do the "transition plots" $u_{i+1}(u_i)$ and $u_{i+2}(u_i)$ for both sequences. Explain.

Comments:

(a) *Transfer plots are a first tool to detect structures in seemingly random noise and to determine the dimension of the generating system.*

(b) *Random number generators provided by the system are typically optimized for speed and may have rather mediocre statistical properties. In Python, the system's PRNG can be accessed via the* `os.urandom` *function; refer to the documentation for more information.*

(c) *Python's* `random` *module is based on the Mersenne Twister algorithm which is comparably slow but has decent statistical properties. The* `numpy.random` *module uses the PCG64 generator, which has better statistical properties. To learn more about PRNGs in Python and how to implement your own, see* `https://realpython.com/python-random/`.

4. **The Last Question** want to discuss ☐

(a) What are the key messages you took home from the lecture?

(b) What are still open questions?

(c) What associated issues did you miss?