Assignment 9

Corrections

- (1a), assume $\lambda_1 > 0$ the for every part.
- (2) the Runge Kutta formulas have been corrected to add parentheses and factors of Δt where they were missing.
- More ideas are added about how to visualize the point cloud for the Lorenz system and its relation to the attractor.

1. Let A be an $n \times n$ square matrix with possibly complex eigenvalues, and right and left eigenvectors (column and row vectors respectively)

$$Ar_j = \lambda_j r_j \ , \ \ \ell_j A = \lambda_j \ell_j$$

Assume the eigenvalues are distinct and the eigenvectors are normalized so that $\ell_j r_j = 1$ for all j. The *power method* for finding the largest eigenvalue and eigenvector constructs a sequence of vectors $x_t \in \mathbb{R}^n$ using the recurrence

$$x_{t+1} = \frac{Ax_t}{\|Ax_t\|} \ .$$

Any vector norm $\|\cdot\|$ may be used. Assume that λ_1 is real and largest in the sense that $|\lambda_1| > |\lambda_j|$ if $j \neq 1$.

(a) Suppose $\lambda_1 > 0$. Show that if $\ell_1 x_0 \neq 0$ then the following limit exists and is a right eigenvector with eigenvalue λ_1:

$$\widehat{r} = \lim_{t \to \infty} x_t \ , \ \ A\widehat{r} = \lambda_1 \widehat{r} \ .$$

Show that

$$\lambda_1 = \lim_{t \to \infty} \frac{\|x_{t+1}\|}{\|x_t\|} \ . \tag{1}$$

Hint. x_t is a scaling of a power of A applied to x_0, which means that if $y_{t+1} = Ay_t$ with $y_0 = x_0$, then $x_t = m_t y_t$. The m_t are numbers that might be called scaling factors or normalization factors. Express y_t and x_t in terms of eigenvectors: $y_t = \sum_j u_{tj} r_j$, $x_t = \sum_j w_{tj} r_j$. You may use the fact (e.g., from the class *Mathematical Analysis*) that if the numbers w_{tj} have limits for each j as $t \to \infty$, then x_t has a limit. The finite limit of the sums is the sum of the limits.

(b) Write a code to implement this power method for the 3×3 matrix produced by the following code. You must use the seed given to get the correct A.

```
import numpy.random as rn
rng = rn.default_rng(seed=19)  # seed must be 19
n = 3
A = rng.normal(0., 1., [n,n])
```

Iterate up to some time T, which you can find by trial and error to give results at nearly machine precision. Do not take T much larger than that. Take x_0 to be random (independent uniforms or normals or any other continuous random variable family).[1] Show that you get an eigenvector whose eigenvalue is (1).[2] Make a log plot of $\|x_{t+1} - x_t\|$ for $0 \le t \le T$, which should show that the convergence is exponentially fast (i.e., linear on a log plot). For part (d), also find the left eigenvector using the power iteration

$$z_{t+1} = \frac{z_t A}{\|z_t A\|} \quad .$$

Use a similar z_0 and iterate up to the same T.

(c) Show that if $\ell_1 x_0 = 0$ then the equation (1) is not satisfied (in exact arithmetic).

(d) Use the left eigenvector $\widehat{\ell_1}$ computed in part (b) to orthogonalize x_0 against ℓ_1. That means, find a linear algebra formula for c so that $\widetilde{x}_0 = x_0 - cr_1$ satisfies the orthogonality relation $\ell_1 \widetilde{x}_0 = 0$. Make a plot of the numbers

$$\frac{\|x_{t+1}\|}{\|x_t\|} \quad .$$

Explain the observation that these numbers eventually converge to λ_1 even though part (c) says they should not.

(e) (*Explanation, nothing to hand in for this*) The power method is a way to find largest eigenvalues of matrices that are too large for direct linear algebra algorithms. However, you have to be careful because some facts of linear algebra, true facts in exact arithmetic, are spoiled in computational algorithms because of numerical instability and inexact arithmetic. Computed results can be completely different from "theoretical" (exact arithmetic) results, not just off by errors on the order of machine precision.

2. *Horner's rule* is a way to evaluate polynomials. Suppose p is a degree d polynomial $p(x) = a_0 + a_1 x + \cdots + a_d x^d$. Suppose none of the coefficients is equal to zero, which will be true in the application of Exercise 3. Horner's

[1] There is literally zero chance (with true random variables in exact arithmetic) that this fails to satisfy the criterion of part (a).

[2] The eigenvalue should be about 1.108717043954.

rule is related to re-writing of the polynomial as

$$
\begin{aligned}
& a_0 + a_1 x + \cdots + a_{d-2} x^{d-2} + a_{d-1} x^{d-1} + a_d x^d \\
&= a_0 + a_1 x + \cdots + a_{d-2} x^{d-2} + a_{d-1} x^{d-1} \left(1 + \frac{a_d}{a_{d-1}} x \right) \\
&= a_0 + a_1 x + \cdots + a_{d-2} x^{d-2} \left(1 + \frac{a_{d-1}}{a_{d-2}} x \left(1 + \frac{a_d}{a_{d-1}} x \right) \right) \\
&\vdots \\
&= a_0 \left(1 + \frac{a_1}{a_0} x \left(1 + \frac{a_2}{a_1} x \left(1 + \cdots \left(1 + \frac{a_{d-1}}{a_{d-2}} x \left(1 + \frac{a_d}{a_{d-1}} x \right) \right) \cdots \right) \right) \right) \\
&= a_0 \left(1 + \frac{a_1}{a_0} x \left(1 + \frac{a_2}{a_1} x \left(1 + \cdots \left(1 + \frac{a_{d-1}}{a_{d-2}} x \qquad y_1 \qquad \right) \cdots \right) \right) \right) \\
&= a_0 \left(1 + \frac{a_1}{a_0} x \left(1 + \frac{a_2}{a_1} x \, (1 + \cdots \qquad y_2 \qquad \cdots) \right) \right) \\
&\vdots
\end{aligned}
$$

For example,

$$1 + 2x + 3x^3 + 4x^3 = 1 + 2x(1 + \tfrac{3}{2}x(1 + \tfrac{4}{3}x)\,) \; .$$

The Horner's rule algorithm is

$$
\begin{aligned}
y_1 &= 1 + \frac{a_d}{a_{d-1}} x \\
y_2 &= 1 + \frac{a_{d-1}}{a_{d-2}} y_1 \\
&\vdots \\
p(x) = y_d &= a_0 \left(1 + \frac{a_1}{a_0} y_{d-1} \right)
\end{aligned}
$$

A matrix version of Horner's rule can be used as a time stepper for solving the IVP for ODEs. Consider the linear ODE system

$$\dot{x} = Ax \; .$$

We saw that the solution is given in terms of the matrix exponential, which has a power series expression

$$
\begin{aligned}
x(t + \Delta t) &= e^{\Delta t A} x(t) \\
&= \left(I + \Delta t A + \frac{1}{2} \Delta t A^2 + \cdots \right) x(t)
\end{aligned}
$$

Consider the approximation that neglects terms beyond the fourth order term

$$e^{\Delta t A} \approx I + \cdots + \frac{\Delta t^4}{24} A^4 \; .$$

(a) Show that has local truncation error order Δt^5 and therefore gives a fourth order method for a linear ODE.

(b) Show that this may be implemented using Horner's rule

$$\begin{aligned} y_1 &= \left(I + \frac{\Delta t}{4} A\right) X_k \\ y_2 &= \left(I + \frac{\Delta t}{3} A\right) y_1 \\ &\vdots \\ X_{k+1} = y_4 &= (I + \Delta t A)\, y_3 \end{aligned}$$

(c) Suppose that if the ODE system has a nonlinear $f(x)$. Consider the multi-stage method

$$y_1 = X_k + \frac{\Delta t}{4} f(X_k) \tag{2}$$

$$y_2 = y_1 + \frac{\Delta t}{3} f(y_1) \tag{3}$$

$$\vdots \tag{4}$$

$$X_{k+1} = y_4 = X_k + \Delta t f(y_3) \tag{5}$$

Show that this is second order accurate in the sense that it is locally third order accurate in the sense that the approximate solution operator $X_{k+1} = \widehat{S}(X_k, \Delta t)$ defined by equations (2) through (5) has $\widehat{S}x, \Delta t) = S(x, \Delta t) + O(\Delta t^3)$. *Discussion.* This is the *low storage* four stage Runge Kutta method. You don't need to store X_k once you have y_1, and so on. It has the same "linearized" behavior as the true fourth order four stage method, which is important in some applications (take *Numerical Methods II* to find out why), but it pays for the low storage by going from fourth order (high) accuracy to second order (lower).

3. Write code to implement the following 4 ODE time stepping algorithms. In each case, the function implementing the solver should take as input the starting point x_0, the time step Δt, the number of time steps n, and the function $f(x)$ that defines the ODE system. For this Exercise, you should calculate Δt from T (the final time you want to simulate up to) and n. The routine should take n steps of size Δt and return the state X_n.

(a) Forward Euler: $X_{k+1} = X_k + \Delta t f(X_k)$.

(b) One of the second order two state predictor/corrector methods (midpoint or trapezoid, you choose).

(c) "The" four state fourth order Runge Kutta method (see Wikipedia, the notes, the book of Dalhquist and Björk or another source for the exact formulas). There should be four evaluations of f per time step. This method is often called RK4.

(d) The four stage low storage RK method defined by equations (2) thfough (5).

Write a separate function for each method so that you can experiment with the different methods individually or in whatever combination you want. Use the code validation based on asymptotic error expansions to determine the order of accuracy of each method using a sequence $\Delta t_1 = \frac{T}{n}$, $\Delta t_2 = \frac{1}{2}\Delta t_1 = \frac{T}{2n}$, etc. Use as test problems the linear problem

$$\begin{aligned} \dot{u} &= -v \\ \dot{v} &= \quad u \\ u(0) = 1 \ , &\quad v(0) = 0 \ . \end{aligned}$$

and the non-linear problem

$$\begin{aligned} \dot{u} &= -\left(x^2 + y^2\right) v \\ \dot{v} &= \quad \left(x^2 + y^2\right) u \\ u(0) = 1 \ , &\quad v(0) = 0 \ . \end{aligned}$$

The solution to both problems is $u(t) = \cos(t)$, $v(t) = \sin(t)$.

Use a fixed but small Δt and compute trajectories for long times for the linear problem using Euler and RK4 (the linear and nonlinear versions of RK4 are the same for a linear problem). Make a scatterplot of the trajectories. Note how qualitative behavior of the numerical approximations differs from that of the true solution and from each other. The scatterplot of the trajectory of the true solution is a circle. Comment on the statement that numerical approximations get the qualitative behavior right even if they get the quantitative values wrong. Comment on the time it takes for the two methods to "drift" off the exact solution circle.

Choose one of these to do. Try to explore beyond what's explicitly asked. Make sure your code is up to standard, in terms of coding itself, automation, and output and visualization.

Lorenz system and chaos The Lorenz system is described in Section 8.7 of the notes by Bindel and me (link in the Materials page of the class web site). Apply the fixed time step ODE solver from Exercise 3 to compute the $12 = 3 + 3 \cdot 3$ component trajectories $(x(t), A(t))$, with $A(t)$ being the Jacobian sensitivity matrix

$$A_{ij}(t, x(0)) = \frac{\partial S_i(t, x(0))}{\partial x_j(0)} \ .$$

The notes explain the ODE that A satisfies. Do not write a special ODE solver, only write an f that takes a 12 component argument and returns a 12 component result. If this code is too slow, you may use a simpler code that only computes the 3 component trajectory $x(t)$ for some parts of this Exercise.

1. Verify that things work by making a figure like Figure 8.4 in the notes. The `Matplotlib` documentation has code that does exactly this here. Documentation (clickable link) You may copy the graphics parts but you need to put the solver method and time step in the title and, of course, use your ODE solver instead of theirs.[3] You should also save the plot in a file to submit with your writeup. See whether you can tell the difference between large and small time steps or good (RK4) and bad (Euler) solvers in the picture.
2. (*Movie*) Choose a large-ish number n (maybe 1000?) and a small-ish point-cloud radius r (maybe .001?) and choose n starting points $x_k(0)$ whose components are independent Gaussians mean $\overline{x}(0)$ and standard deviation r. For every time T, this defines a point cloud with n points $x_k(T)$. Each frame of the movie should be a 3D scatterplot of this point cloud (example here (clickable link)). It will be more clear if you put the butterfly picture (a long curve produced by the ODE solver) in the plot. The trajectory is a picture of the *attractor* (the set all trajectories approach). The point cloud will disperse to fill out the attractor. For this to be clear, you have to figure out how to make the curve that is a single long trajectory not dominate the figure. For this, make a thin line (set the linewidth to be smaller) and not as dark (choose a lighter color and change the parameter that controls opacity of the line).

2D ions in a trap simulation We sometimes denote a point in 2D as $r = (x, y)$. The inverse square electrostatic repulsion between ions at $r_1 = (x_1, y_1)$ and $r_2 = (x_2, y_2)$ is

$$F_{12} = \frac{r_1 - r_1}{\|r_1 - r_2\|^3} \ .$$

More precisely, this is the force on the r_1 ion from the r_2 ion. The force on the r_2 ion from the r_1 ion is the negative of this. The force is in the direction between r_1 and r_2 and pushes the ions apart. The magnitude of the force is proportional to the *inverse square* of the distance between them:

$$\|F_{12}\| = \frac{1}{\|r_1 - r_2\|^2} \ .$$

This electrostatic force is a *two body* force because it depends on the location of two ions. There also is the trapping force, which is a one body

[3] Their ODE solver has a great docstring. Feel free to adopt that style for docstrings in your code.

force toward the center of the trap proportional to the distance from the center

$$F_1(r_1) = -r_1 \ .$$

If the ions are all on the y axis, these forces are the same as the ones from Assignment 8. The total force on ion j is the sum of the two body forces from the other ions and the trapping force:

$$F_{\text{tot},j} = \sum_{k \neq j} F_{jk} + F_j \ .$$

Note that $F_{jk} = -F_{kj}$, which means that the ions push each other in the opposite direction.[4]

We consider a linear frictional force, which is a force proportional to the speed of an ion, but in the opposite direction.[5] The *friction coefficient* is γ:

$$F_{\text{fr}} = -\gamma \dot{r} \ .$$

Putting the forces together, the dynamics are

$$\ddot{r}_j = F_{\text{tot},j} + F_{\text{fr}} = F_{\text{tot},j} - \gamma \dot{r}_j \ . \tag{6}$$

The potential energy is the 2D version of the potential energy from Assignment 8

$$U(r_1, \cdots, r_n) = \sum_{j=1}^{n} \tfrac{1}{2} \left\| r_j \right\|^2 + \sum_{j<k} \frac{1}{\left\| r_j - r_k \right\|} \ . \tag{7}$$

As in the 1D version, the force F_{tot} is the negative of the gradient of the potential.

Write a function that implements the $f(x)$ in the first order ODE system formulation of the ODE system (6). The dimension of x is $4n$, where n is the number of ions. The dimension goes from n to $2n$ because each ion has two coordinates, then from $2n$ to $4n$ when you add "dummy" variables $\dot{r}_j$ to the system. Use the RK4 ODE solver and solver code from Exercise 3. Make movies to illustrate the results.

1. Verify that if the r_j satisfy the dynamics (6) then

$$\frac{d}{dt} U(r_1(t), \cdots, r_n(t)) = -\gamma \sum_{j=1}^{n} \left\| \dot{r}_j \right\|^2 \ .$$

[4] In a physics class we would dwell on this point longer. It is one of Newton's laws of motion and is related to conservation of momentum total momentum.

[5] This part is "highly non-physical". Friction would come from ions moving in air, but there is no air in an ion trap. The real "cooling" mechanism in ion traps is laser doppler cooling, which is an idea as subtle and brilliant as the Paul trap itself and also received a Nobel prize. If there were friction with air as tennis ball would feel, the force would depend on the velocity in a "highly nonlinear" way because the air flow around a tennis ball is complex and *highly nonlinear*.

Show that this implies that this implies that $U(r(t))$ is a decreasing function of t and that if $r(t) \to r^*$ as $t \to \infty$, then r^* is a stationary point for U. Here $r = (r_1, \cdots, r_n)$. Keep in mind (or verify if you're not sure) that $F_{\text{tot}} = -\nabla U$. Conclude (if you agree) that solving the ODE (6) is a minimization algorithm that uses the same information as gradient descent.

2. Make movies showing showing the trajectory of the solution of (6). Each frame should be a scatterplot of the positions $r_j(t)$ at some t. Experiment with large and small γ, different n (maybe pick just one n in the end), and initial configurations. Describe the difference between the behavior of solutions for large and small γ.

3. Fix γ at a reasonably high value, but not so high that the ODE solution is inaccurate. Choose a large n (more than 100 and possibly more than 1000) and describe the steady states you get. Pay attention particularly to the half of the ions nearest the center. A *crystal structure* is a regular arrangement of points (atoms or ions). A triangular lattice is a crystal structure made up of regular triangles arranged point to point and edge to edge.

4. Parts 1 to 3 have rotation symmetry. Break this symmetry by changing the trap potential from $\frac{1}{2}\left(x^2 + y^2\right)$ to the more realistic *anisotropic* ("isotropic" means having rotational symmetry) potential $\frac{1}{2}x^2 + 3y^2$. The form is not important for this. If you replace 3 with a much larger number, the ions will "want" to be on the x axis. The ion trap in the link of Assignment 8 is not one dimensional, merely "highly anisotropic". This exercise will not work for highly anisotropic traps, only moderately anisotropic ones. Start with various initial configurations and see whether the eventual steady states (local minimizers) are the same. If you find good parameter combinations, this will illustrate the phrase you hear in AI all the time: multiple local minima.