This chapter is taken from the book A Primer on Scientific Programming with Python (http://www.springer.com/gp/book/9783662498866) by H. P. Langtangen, 5th edition, Springer, 2016.

Exercises

Exercise 1: Determine the limit of a sequence

a) Write a Python function for computing and returning the sequence

$$a_n = \frac{7+1/(n+1)}{3-1/(n+1)^2}, \quad n = 0, 2, \ldots, N\,.$$

Write out the sequence for $N = 100$. Find the exact limit as $N \rightarrow \infty$ and compare with a_N.

b) Write a Python function for computing and returning the sequence

$$D_n = \frac{\sin(2^{-n})}{2^{-n}}, \quad n = 0, \ldots, N\,.$$

Determine the limit of this sequence for large N.

c) Given the sequence

$$D_n = \frac{f(x+h)-f(x)}{h}, \quad h = 2^{-n}, \tag{47}$$

make a function `D(f, x, N)` that takes a function $f(x)$, a value x, and the number N of terms in the sequence as arguments, and returns the sequence D_n for $n = 0, 1, \ldots, N$. Make a call to the `D` function with $f(x) = \sin x$, $x = 0$, and $N = 80$. Plot the evolution of the computed D_n values, using small circles for the data points.

d) Make another call to `D` where $x = \pi$ and plot this sequence in a separate figure. What would be your expected limit?

e) Explain why the computations for $x = \pi$ go wrong for large N.

☞ **Hint.**

Filename: `sequence_limits`.

Exercise 2: Compute π via sequences

The following sequences all converge to π:

$$
\begin{aligned}
(a_n)_{n=1}^{\infty}, &\quad a_n = 4\sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1},\\
(b_n)_{n=1}^{\infty}, &\quad b_n = \left(6\sum_{k=1}^{n} k^{-2}\right)^{1/2},\\
(c_n)_{n=1}^{\infty}, &\quad c_n = \left(90\sum_{k=1}^{n} k^{-4}\right)^{1/4},\\
(d_n)_{n=1}^{\infty}, &\quad d_n = \frac{6}{\sqrt{3}}\sum_{k=0}^{n} \frac{(-1)^{k}}{3^k(2k+1)},\\
(e_n)_{n=1}^{\infty}, &\quad e_n = 16\sum_{k=0}^{n} \frac{(-1)^{k}}{5^{2k+1}(2k+1)} - 4\sum_{k=0}^{n} \frac{(-1)^{k}}{239^{2k+1}(2k+1)}\,.
\end{aligned}
$$

Make a function for each sequence that returns an array with the elements in the sequence. Plot all the sequences, and find the one that converges fastest toward the limit π. Filename: `pi_sequences`.

Exercise 3: Reduce memory usage of difference equations

Consider the program `growth_years.py` from the section Interest rates (._diffeq-readable001.html#sec:diffeq:irates). Since x_n depends on x_{n-1} only, we do not need to store all the $N+1$ x_n values. We actually only need to store x_n and its previous value x_{n-1}. Modify the program to use two variables and not an array for the entire sequence. Also avoid the `index_set` list and use an integer counter for n and a `while` loop instead. Write the sequence to file such that it can be visualized later. Filename: `growth_years_efficient`.

Exercise 4: Compute the development of a loan

Solve (16) (._diffeq-readable001.html#mjx-eqn-16)-(17) (._diffeq-readable001.html#mjx-eqn-17) in a Python function. Filename: `loan`.

Exercise 5: Solve a system of difference equations

Solve (32) (._diffeq-readable001.html#mjx-eqn-32)-(33) (._diffeq-readable001.html#mjx-eqn-33) in a Python function and plot the x_n sequence. Filename: `fortune_and_inflation1`.

Exercise 6: Modify a model for fortune development

In the model (32) (._diffeq-readable001.html#mjx-eqn-32)-(33) (._diffeq-readable001.html#mjx-eqn-33) the new fortune is the old one, plus the interest, minus the consumption. During year n, x_n is normally also reduced with t percent tax on the earnings $x_{n-1} - x_{n-2}$ in year $n-1$.

a) Extend the model with an appropriate tax term, implement the model, and demonstrate in a plot the effect of tax ($t = 27$) versus no tax ($t = 0$).

b) Suppose you expect to live for N years and can accept that the fortune x_n vanishes after N years. Choose some appropriate values for p, q, I, and t, and experiment with the program to find how large the initial c_0 can be in this case.

Filename: `fortune_and_inflation2`.

Exercise 7: Change index in a difference equation

A mathematically equivalent equation to (5) (._diffeq-readable001.html#mjx-eqn-5) is

$$x_{i+1} = x_i + \frac{p}{100}x_i, \tag{48}$$

since the name of the index can be chosen arbitrarily. Suppose someone has made the following program for solving (48):

```
from scitools.std import *
x0 = 100                # initial amount
p = 5                   # interest rate
N = 4                   # number of years
index_set = range(N+1)
x = zeros(len(index_set))

# Compute solution
x[0] = x0
for i in index_set[1:]:
    x[i+1] = x[i] + (p/100.0)*x[i]
print x
plot(index_set, x, 'ro', xlabel='years', ylabel='amount')
```

This program does not work. Make a correct version, but keep the difference equations in its present form with the indices `i+1` and `i`. Filename: `growth1_index_ip1`.

Exercise 8: Construct time points from dates

A certain quantity p (which may be an interest rate) is piecewise constant and undergoes changes at some specific dates, e.g.,

$$p \text{ changes to } \begin{cases} 4.5 & \text{on Jan 4, 2019} \\ 4.75 & \text{on March 21, 2019} \\ 6.0 & \text{on April 1, 2019} \\ 5.0 & \text{on June 30, 2019} \\ 4.5 & \text{on Nov 1, 2019} \\ 2.0 & \text{on April 1, 2020} \end{cases} \tag{49}$$

Given a start date d_1 and an end date d_2, fill an array `p` with the right p values, where the array index counts days. Use the `datetime` module to compute the number of days between dates. Filename: `dates2days`.

Exercise 9: Visualize the convergence of Newton's method

Let $x_0, x_1, \ldots, x_N$ be the sequence of roots generated by Newton's method applied to a nonlinear algebraic equation $f(x) = 0$ (see the section Newton's method (._diffeq-readable001.html#sec:diffeq:Newtonsmethod:sec)). In this exercise, the purpose is to plot the sequences $(x_n)_{n=0}^N$ and $(|f(x_n)|)_{n=0}^N$ such that we can understand how Newton's method converges or diverges.

a) Make a general function

```
Newton_plot(f, x, dfdx, xmin, xmax, epsilon=1E-7)
```

for this purpose. The arguments `f` and `dfdx` are Python functions representing the $f(x)$ function in the equation and its derivative $f'(x)$, respectively. Newton's method is run until $|f(x_N)| \leq \epsilon$, and the ϵ value is available as the `epsilon` argument. The `Newton_plot` function should make three separate plots of $f(x)$, $(x_n)_{n=0}^N$, and $(|f(x_n)|)_{n=0}^N$ on the screen and also save these plots to PNG files. The relevant x interval for plotting of $f(x)$ is given by the arguments `xmin` and `xmax`. Because of the potentially wide scale of values that $|f(x_n)|$ may exhibit, it may be wise to use a logarithmic scale on the y axis.

☞ **Hint.**

b) Demonstrate the function on the equation $x^6 \sin \pi x = 0$, with $\epsilon = 10^{-13}$. Try different starting values for Newton's method: $x_0 = -2.6, -1.2, 1.5, 1.7, 0.6$. Compare the results with the exact solutions $x = \ldots, -2 - 1, 0, 1, 2, \ldots$.

c) Use the `Newton_plot` function to explore the impact of the starting point x_0 when solving the following nonlinear algebraic equations:

$$\begin{aligned} \sin x &= 0, && (50)\\ x &= \sin x, && (51)\\ x^5 &= \sin x, && (52)\\ x^4 \sin x &= 0, && (53)\\ x^4 &= 16, && (54)\\ x^{10} &= 1, && (55)\\ \tanh x &= 0\,.\tanh x \qquad = x^{10}\,. && (56) \end{aligned}$$

☞ **Hint.**

Filename: `Newton2`.

Exercise 10: Implement the secant method

Newton's method (34) (._diffeq-readable001.html#mjx-eqn-34) for solving $f(x) = 0$ requires the derivative of the function $f(x)$. Sometimes this is difficult or inconvenient. The derivative can be approximated using the last two approximations to the root, x_{n-2} and x_{n-1}:

$$f'(x_{n-1}) \approx \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}\,.$$

Using this approximation in (34) (._diffeq-readable001.html#mjx-eqn-34) leads to the Secant method:

$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-1} - x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}, \quad x_0, x_1 \text{ given}\,. \tag{57}$$

Here $n = 2, 3, \ldots$. Make a program that applies the Secant method to solve $x^5 = \sin x$. Filename: `Secant`.

Exercise 11: Test different methods for root finding

Make a program for solving $f(x) = 0$ by Newton's method, the Bisection method, and the Secant method. For each method, the sequence of root approximations should be written out (nicely formatted) on the screen. Read $f(x)$, $f'(x)$, a, b, x_0, and x_1 from the command line. Newton's method starts with x_0, the Bisection method starts with the interval $[a, b]$, whereas the Secant method starts with x_0 and x_1.

Run the program for each of the equations listed in Exercise 9: Visualize the convergence of Newton's methodd. You should first plot the $f(x)$ functions so you know how to choose x_0, x_1, a, and b in each case. Filename: `root_finder_examples`.

Exercise 12: Make difference equations for the Midpoint rule

Use the ideas of the section The integral as a difference equation (._diffeq-readable001.html#sec:diffeq:integral) to make a similar system of difference equations and corresponding implementation for the Midpoint integration rule:

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f(a - \frac{1}{2}h + ih),$$

where $h = (b-a)/n$ and n counts the number of function evaluations (i.e., rectangles that approximate the area under the curve). Filename: `diffeq_midpoint`.

Exercise 13: Compute the arc length of a curve

Sometimes one wants to measure the length of a curve $y = f(x)$ for $x \in [a, b]$. The arc length from $f(a)$ to some point $f(x)$ is denoted by $s(x)$ and defined through an integral

$$s(x) = \int_a^x \sqrt{1 + [f'(\xi)]^2} d\xi \,. \tag{58}$$

We can compute $s(x)$ via difference equations as explained in the section The integral as a difference equation (._diffeq-readable001.html#sec:diffeq:integral).

a) Make a Python function `arclength(f, a, b, n)` that returns an array `s` with $s(x)$ values for n uniformly spaced coordinates x in $[a, b]$. Here `f(x)` is the Python implementation of the function that defines the curve we want to compute the arc length of.

b) How can you verify that the `arclength` function works correctly? Construct test case(s) and write corresponding test functions for automating the tests.

☞ **Hint.**

c) Apply the function to

$$f(x) = \int_{-2}^{x} = \frac{1}{\sqrt{2\pi}} e^{-4t^2} dt, \quad x \in [-2, 2] \,.$$

Compute $s(x)$ and plot it together with $f(x)$.

Filename: `arclength`.

Exercise 14: Find difference equations for computing $\sin x$

The purpose of this exercise is to derive and implement difference equations for computing a Taylor polynomial approximation to $\sin x$:

$$\sin x \approx S(x; n) = \sum_{j=0}^{n} (-1)^j \frac{x^{2j+1}}{(2j+1)!} \,. \tag{59}$$

To compute $S(x; n)$ efficiently, write the sum as $S(x; n) = \sum_{j=0}^{n} a_j$, and derive a relation between two consecutive terms in the series:

$$a_j = -\frac{x^2}{(2j+1)2j} a_{j-1} \,. \tag{60}$$

Introduce $s_j = S(x; j-1)$ and a_j as the two sequences to compute. We have $s_0 = 0$ and $a_0 = x$.

a) Formulate the two difference equations for s_j and a_j.

☞ **Hint.**

b) Implement the system of difference equations in a function `sin_Taylor(x, n)`, which returns s_{n+1} and $|a_{n+1}|$. The latter is the first neglected term in the sum (since $s_{n+1} = \sum_{j=0}^{n} a_j$) and may act as a rough measure of the size of the error in the Taylor polynomial approximation.

c) Verify the implementation by computing the difference equations for $n = 2$ by hand (or in a separate program) and comparing with the output from the `sin_Taylor` function. Automate this comparison in a test function.

d) Make a table or plot of s_n for various x and n values to illustrate that the accuracy of a Taylor polynomial (around $x = 0$) improves as n increases and x decreases.

☞ **Hint.**

Filename: `sin_Taylor_series_diffeq`.

Exercise 15: Find difference equations for computing cos x

Solve Exercise 14: Find difference equations for computing sin x for the Taylor polynomial approximation to cos x. (The relevant expression for the Taylor series is easily found in a mathematics textbook or by searching on the Internet.) Filename: `cos_Taylor_series_diffeq`.

Exercise 16: Make a guitar-like sound

Given start values $x_0, x_1, \ldots, x_p$, the following difference equation is known to create guitar-like sound:

$$x_n = \frac{1}{2}(x_{n-p} + x_{n-p-1}), \quad n = p+1, \ldots, N\,. \tag{61}$$

With a sampling rate r, the frequency of this sound is given by r/p. Make a program with a function `solve(x, p)` which returns the solution array `x` of (61). To initialize the array `x[0:p+1]` we look at two methods, which can be implemented in two alternative functions:

- $x_0 = 1, x_1 = x_2 = \cdots = x_p = 0$
- $x_0, \ldots, x_p$ are uniformly distributed random numbers in $[-1, 1]$

Import `max_amplitude`, `write`, and `play` from the `scitools.sound` module. Choose a sampling rate r and set $p = r/440$ to create a 440 Hz tone (A). Create an array `x1`

of zeros with length $3r$ such that the tone will last for 3 seconds. Initialize `x1` according to method 1 above and solve (61). Multiply the `x1` array by `max_amplitude`. Repeat this process for an array `x2` of length $2r$, but use method 2 for the initial values and choose p such that the tone is 392 Hz (G). Concatenate `x1` and `x2`, call `write` and then `play` to play the sound. As you will experience, this sound is amazingly similar to the sound of a guitar string, first playing A for 3 seconds and then playing G for 2 seconds.
The method (61) is called the Karplus-Strong algorithm and was discovered in 1979 by a researcher, Kevin Karplus, and his student Alexander Strong, at Stanford University. Filename: `guitar_sound`.

Exercise 17: Damp the bass in a sound file

Given a sequence $x_0, \ldots, x_{N-1}$, the following *filter* transforms the sequence to a new sequence $y_0, \ldots, y_{N-1}$:

$$y_n = \begin{cases} x_n, & n = 0 \\ -\frac{1}{4}(x_{n-1} - 2x_n + x_{n+1}), & 1 \leq n \leq N-2 \\ x_n, & n = N-1 \end{cases} \tag{62}$$

If x_n represents sound, y_n is the same sound but with the bass damped. Load some sound file, e.g.,

```
x = scitools.sound.Nothing_Else_Matters()
# or
x = scitools.sound.Ja_vi_elsker()
```

to get a sound sequence. Apply the filter (62) and play the resulting sound. Plot the first 300 values in the x_n and y_n signals to see graphically what the filter does with the signal. Filename: `damp_bass`.

Exercise 18: Damp the treble in a sound file

Solve Exercise 17: Damp the bass in a sound file to get some experience with coding a filter and trying it out on a sound. The purpose of this exercise is to explore some other filters that reduce the treble instead of the bass. Smoothing the sound signal will in general damp the treble, and smoothing is typically obtained by letting the values in the new filtered sound sequence be an average of the neighboring values in the original sequence.

The simplest smoothing filter can apply a standard average of three neighboring values:

$$y_n = \begin{cases} x_n, & n = 0 \\ \frac{1}{3}(x_{n-1} + x_n + x_{n+1}), & 1 \le n \le N-2 \\ x_n, & n = N-1 \end{cases} \qquad (63)$$

Two other filters put less emphasis on the surrounding values:

$$y_n = \begin{cases} x_n, & n = 0 \\ \frac{1}{4}(x_{n-1} + 2x_n + x_{n+1}), & 1 \le n \le N-2 \\ x_n, & n = N-1 \end{cases} \qquad (64)$$

$$y_n = \begin{cases} x_n, & n = 0,1 \\ \frac{1}{16}(x_{n-2} + 4x_{n-1} + 6x_n + 4x_{n+1} + x_{n+2}), & 2 \le n \le N-3 \\ x_n, & n = N-2, N-1 \end{cases} \qquad (65)$$

Apply all these three filters to a sound file and listen to the result. Plot the first 300 values in the x_n and y_n signals for each of the three filters to see graphically what the filter does with the signal. Filename: `damp_treble`.

Exercise 19: Demonstrate oscillatory solutions of the logistic equation

a) Write a program to solve the difference equation (13) (._diffeq-readable001.html#mjx-eqn-13):

$$y_n = y_{n-1} + q y_{n-1}\left(1 - y_{n-1}\right), \quad n = 0, \ldots, N\,.$$

Read the input parameters y_0, q, and N from the command line. The variables and the equation are explained in the section Logistic growth (._diffeq-readable001.html#sec:diffeq:logistic).

b) Equation (13) (._diffeq-readable001.html#mjx-eqn-13) has the solution $y_n = 1$ as $n \to \infty$. Demonstrate, by running the program, that this is the case when $y_0 = 0.3$, $q = 1$, and $N = 50$.

c) For larger q values, y_n does not approach a constant limit, but y_n oscillates instead around the limiting value. Such oscillations are sometimes observed in wildlife populations. Demonstrate oscillatory solutions when q is changed to 2 and 3.

d) It could happen that y_n stabilizes at a constant level for larger N. Demonstrate that this is not the case by running the program with $N = 1000$.

Filename: `growth_logistic2`.

Exercise 20: Automate computer experiments

It is tedious to run a program like the one from Exercise 19: Demonstrate oscillatory solutions of the logistic equation repeatedly for a wide range of input parameters. A better approach is to let the computer do the manual work. Modify the program from Exercise 19: Demonstrate oscillatory solutions of the logistic equation such that the computation of y_n and the plot is made in a function. Let the title in the plot contain the parameters y_0 and q (N is easily visible from the x axis). Also let the name of the plot file reflect the values of y_0, q, and N. Then make loops over y_0 and q to perform the following more comprehensive set of experiments:

- $y_0 = 0.01, 0.3$
- $q = 0.1, 1, 1.5, 1.8, 2, 2.5, 3$
- $N = 50$

How does the initial condition (the value y_0) seem to influence the solution?

☞ **Hint.**

Filename: `growth_logistic3`.

Exercise 21: Generate an HTML report

Extend the program made in Exercise 20: Automate computer experiments with a report containing all the plots. The report can be written in HTML and displayed by a web browser. The plots must then be generated in PNG format. The source of the HTML file will typically look as follows:

```
<html>
<body>
<p><img src="tmp_y0_0.01_q_0.1_N_50.png">
<p><img src="tmp_y0_0.01_q_1_N_50.png">
<p><img src="tmp_y0_0.01_q_1.5_N_50.png">
<p><img src="tmp_y0_0.01_q_1.8_N_50.png">
...
<p><img src="tmp_y0_0.01_q_3_N_1000.png">
</html>
</body>
```

Let the program write out the HTML text to a file. You may let the function making the plots return the name of the plot file such that this string can be inserted in the HTML file. Filename: `growth_logistic4`.

Exercise 22: Use a class to archive and report experiments

The purpose of this exercise is to make the program from Exercise 21: Generate an HTML report more flexible by creating a Python class that runs and archives all the experiments (provided you know how to program with Python classes). Here is a sketch of the class:

```python
class GrowthLogistic(object):
    def __init__(self, show_plot_on_screen=False):
        self.experiments = []
        self.show_plot_on_screen = show_plot_on_screen
        self.remove_plot_files()

    def run_one(self, y0, q, N):
        """Run one experiment."""
        # Compute y[n] in a loop...
        plotfile = 'tmp_y0_%g_q_%g_N_%d.png' % (y0, q, N)
        self.experiments.append({'y0': y0, 'q': q, 'N': N,
                                 'mean': mean(y[20:]),
                                 'y': y, 'plotfile': plotfile})
        # Make plot...

    def run_many(self, y0_list, q_list, N):
        """Run many experiments."""
        for q in q_list:
            for y0 in y0_list:
                self.run_one(y0, q, N)

    def remove_plot_files(self):
        """Remove plot files with names tmp_y0*.png."""
        import os, glob
        for plotfile in glob.glob('tmp_y0*.png'):
            os.remove(plotfile)

    def report(self, filename='tmp.html'):
        """
        Generate an HTML report with plots of all
        experiments generated so far.
        """
        # Open file and write HTML header...
        for e in self.experiments:
            html.write('<p><img src="%s">\n' % e['plotfile'])
        # Write HTML footer and close file...
```

Each time the `run_one` method is called, data about the current experiment is stored in the `experiments` list. Note that `experiments` contains a list of dictionaries. When desired, we can call the `report` method to collect all the plots made so far in an HTML report. A typical use of the class goes as follows:

```
N = 50
g = GrowthLogistic()
g.run_many(y0_list=[0.01, 0.3],
           q_list=[0.1, 1, 1.5, 1.8] + [2, 2.5, 3], N=N)
g.run_one(y0=0.01, q=3, N=1000)
g.report()
```

Make a complete implementation of class `GrowthLogistic` and test it with the small program above. The program file should be constructed as a module. Filename: `growth_logistic5`.

Exercise 23: Explore logistic growth interactively

Class `GrowthLogistic` from Exercise 22: Use a class to archive and report experiments is very well suited for interactive exploration. Here is a possible sample session for illustration:

```
>>> from growth_logistic5 import GrowthLogistic
>>> g = GrowthLogistic(show_plot_on_screen=True)
>>> q = 3
>>> g.run_one(0.01, q, 100)
>>> y = g.experiments[-1]['y']
>>> max(y)
1.3326056469620293
>>> min(y)
0.0029091569028512065
```

Extend this session with an investigation of the oscillations in the solution y_n. For this purpose, make a function for computing the local maximum values y_n and the corresponding indices where these local maximum values occur. We can say that y_i is a local maximum value if

$$y_{i-1} < y_i > y_{i+1}\,.$$

Plot the sequence of local maximum values in a new plot. If $I_0, I_1, I_2, \ldots$ constitute the set of increasing indices corresponding to the local maximum values, we can define the periods of the oscillations as $I_1 - I_0$, $I_2 - I_1$, and so forth. Plot the length of the periods in a separate plot. Repeat this investigation for $q = 2.5$. Filename: `GrowthLogistic_interactive`.

Exercise 24: Simulate the price of wheat

The demand for wheat in year t is given by

$$D_t = ap_t + b,$$

where $a < 0, b > 0$, and p_t is the price of wheat. Let the supply of wheat be

$$S_t = Ap_{t-1} + B + \ln(1 + p_{t-1}),$$

where A and B are given constants. We assume that the price p_t adjusts such that all the produced wheat is sold. That is, $D_t = S_t$.

a) For $A = 1$, $a=-3,b=5,B=0$, find from numerical computations, a stable price such that the production of wheat from year to year is constant. That is, find p such that $ap + b = Ap + B + \ln(1 + p)$.

b) Assume that in a very dry year the production of wheat is much less than planned. Given that price this year, p_0, is 4.5 and $D_t = S_t$, compute in a program how the prices $p_1, p_2, \ldots, p_N$ develop. This implies solving the difference equation

$$ap_t + b = Ap_{t-1} + B + \ln(1 + p_{t-1}) .$$

From the p_t values, compute S_t and plot the points (p_t, S_t) for $t = 0, 1, 2, \ldots, N$. How do the prices move when $N \to \infty$?

Filename: `wheat`.

References

← Prev (._diffeq-readable002.html)