university of groningen / faculty of mathematics and natural sciences / computing science

Practicals Introduction to Scientific Computing

Jos B.T.M. Roerdink

February 2017

1 General information

1.1 Auxiliary files

Auxiliary files for the practicals of the course Introduction to Scientific Computing can be found in Nestor as zip archives under "Practicals".

1.2 Setting the Matlab path

You can set the path via an environment variable on the command line (by '$' we indicate the command prompt of the Unix shell, and by '>>' the Matlab prompt). For example,

```
$ export MATLABPATH=<your-directory>
```

You can also add the path on the Matlab command prompt:

```
>> path('<your-directory>', path)
```

To get an overview of the directories in your Matlab-pad, type:

```
>> path
```

1.3 Images in Matlab

Image display

The function `imshow` can be used to display an image in a figure window. For example,

```
>> imshow('moon.tif')
```

will show image `moon.tif` in the current figure window or will open a new window if necessary.
The function `image` can be used to display a matrix `C` as an image:

```
>> image(C);
```

Usually, it is more convenient to use the function `imagesc`, which scales the values in the matrix in such a way that the full gray scale range is used. Also, a colour table can be specified. For example,

```
>> imagesc(C);
>> colourmap(gray(256));
```

Image I/O

Images can be read in Matlab via the function `imread`. For example,

```
>> x = imread('moon.tif');
```

reads the image `'moon.tif'` from the current directory or a directory in the Matlab search path.
The function `imwrite` can be used to write images. For example,

```
>> imwrite(x, 'newmoon.tif');
```

writes `x` in TIFF format to the current directory with filename `'newmoon.tif'`.
Both `imread` and `imwrite` have a number of optional parameters, see Matlab's `help` for more details.

1.4 Reporting

- For your reports, you need to use the LaTeX templates that are provided in Nestor. An example assignment with report (both the LaTeX code and the PDF) is also included.
- Whenever your are required to generate some output (including images), include it in your report.
- Use appropriate names for variables, e.g., i, j for loop variables, 'count' for a counter, etc.
- Include all requested and created code files in appendices of your report. Insert brief code snippets at the appropriate places in your report, or refer to the lines numbers of the appendices that contain your code.
- List your code neatly using `lstlistings`. You can tweak the settings in such a way that the font is no smaller than `\small`, line breaks are added automatically, and line numbers are added as well. See the LaTeX templates for examples. For more information, see `http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings`.

1.5 Practicals

In an assignment, a number between brackets indicates the maximal number of points you can score for that assignment.

Practical 0: Introduction to Matlab

See Nestor.

Practical 1: Sequence alignment

In this practical you will implement the algorithm for global sequence alignment in Matlab. We limit ourselves to sequences of DNA nucleotides: A, C, T, G.

Assignment 1 Basic Needleman-Wunsch algorithm (35)

a. Implement the basic Needleman-Wunsch algorithm (Algorithm 2.1 from the syllabus) in Matlab. Give the file with your program the name `nw1.m`.
Your program should read two sequences of length n and m from an input file. This can be done by the following lines of Matlab code (assuming the input file is `input.txt`):

```
in=fopen('input.txt');                % open input file
s=fgetl(in);                          % read line 1 of the input
t=fgetl(in);                          % read line 2 of the input
fclose(in);                           % close file
len_s=length(s);                      % length string s
len_t=length(t);                      % length string t

% Here comes your code
 ................
% The result should be a matrix $D$ containing the costs of optimal alignment
```

The scoring matrix $w(a_i, b_j)$ has the form:

$$w = \begin{pmatrix} p & q & q & q \\ q & p & q & q \\ q & q & p & q \\ q & q & q & p \end{pmatrix}$$

That is, the costs of a match are p and of a mismatch q, e.g., $w(A, A) = p$, $w(A, T) = q$, etc. Furthermore, the gap penalty is g.
The parameters p, q and g should be variables in your algorithm, to be initialized at the start of your Matlab-program. The result of your program is a $(n + 1) \times (m + 1)$-dimensional matrix D containing the costs of optimal alignment.

b. Inspect the file `nw_test1.txt` from the archive Practical1.zip in Nestor. This file contains the two sequences:

```
GGAATGG
ATG
```

Run your program `nw1.m` in Matlab with `nw_test1.txt` as input, and the following values of the parameters: $p = 0$, $q = 4$, $g = 5$.
The matrix D that contains the costs (edit distances) of the alignment should agree with the matrix at the bottom of p. 26 of the syllabus (without the arrows).

Assignment 2 Needleman-Wunsch algorithm with predecessors (35)

Next you will expand the basic algorithm so that not only the costs of an optimal alignment are computed, but also the predecessors. We limit ourselves to the determination of *one* predecessor for each position (i, j).

a. Implement the Needleman-Wunsch algorithm with predecessors in Matlab by expanding your program `nw1.m`. Call your new program `nw2.m`. The input and parameters are the same as in Assignment 1.
The result of your program should be as follows:

   (a) a $(n+1) \times (m+1)$-dimensional matrix D with the the optimal alignment (as in Assignment 1);

   (b) a $(n+1) \times (m+1)$-dimensional matrix P, whose elements indicate the direction of the predecessor. For this purpose we use simple characters that occur on your keyboard, that is, `'|'` for "from the north", `'-'` for "from the west", and `'\'` for "from the north west". More precisely:

$$P(i,j) = \begin{cases} '-' & \text{if } (i, j-1) \text{ is the predecessor} \\ '\backslash' & \text{if } (i-1, j-1) \text{ is the predecessor} \\ '|' & \text{if } (i-1, j) \text{ is the predecessor} \\ '*' & \text{if } (i,j) \text{ has no predecessor} \end{cases}$$

   N.B. The last case occurs at the position on the first row and the first column. In case there is more than one predecessor, you have to choose one, with the following priority order: north west; west; north.

b. Run your program `nw2.m`, again with `nw_test1.txt` as input, and the same values of the parameters: $p = 0$, $q = 4$, $g = 5$.
Let Matlab print the matrices D and P below one another on `stdout`. The result should be as follows (corresponding to the matrix at the bottom of p. 26 of the syllabus):

```
D =

     0     5    10    15
     5     4     9    10
    10     9     8     9
    15    10    13    12
    20    15    14    17
    25    20    15    18
    30    25    20    15
    35    30    25    20

P =

*---
|\\\
|\\\
|\\\
|\\\
||\\
|||\
|||\
```

Assignment 3 Needleman-Wunsch algorithm with optimal alignment (30)

Finally, you will expand the algorithm such that also the optimal alignment is determined. Use your program `nw2.m` as basis. Call the corresponding program `nw3.m`.

a. Compute the alignment of the two input strings `s` and `t`. By determining the trace-back path (repeatedly following the predecessor) you can walk back from the endpoint $(n+1, m+1)$ to the starting point $(1,1)$. (Because we have chosen only a single predecessor in each point there is also only *one* optimal alignment.) The strings after alignment we call `s_al` and `t_al`. So these are calculated "from back to front". Determine during the calculation also a string `l_al` (with the same length as `s_al` and `t_al`) which has a `'|'` symbol on position `k` if a match occurs between the input strings at that position, and a space otherwise.

b. Run your program `nw3.m`, again with `nw_test1.txt` as input, and the same values of the parameters: $p = 0$, $q = 4$, $g = 5$.
Let Matlab print the strings `s_al`, `l_al` and `t_al` orderly formatted below one another on `stdout`. The result should be as follows:

```
GGAATGG
   || |
---AT-G
```

c. Instead of printing the results on `stdout` by the `printf` function you can also print the results to a file by the `fprintf` function, as follows. Include the following lines[1] at the end of your program `nw3.m`, and add the missing code for printing. The code for printing the input sequences has already been inserted.

```
output=fopen('nw3-output.txt', 'w');       % open file
fprintf(output,'Name: <Your name(s)>\n'); % enter your name(s)
fprintf(output,'IBC, Practical 3\n\n');

fprintf(output,'\n\nString s:\n');
for i=1:length(s)
  fprintf(output,'%s',s(i));
end
fprintf(output,'\n\nString t:\n');
for i=1:length(t)
  fprintf(output,'%s',t(i));
end

fprintf(output,'\n\nMatrix D:\n\n');
% Here comes the code for printing matrix D

fprintf(output,'\n\nMatrix P:\n\n');
% Here comes the code for printing matrix P

fprintf(output,'\n\nAlignment:\n\n');
% Here comes the code for printing the alignment
% That is, strings s_al, l_al, t_al below one another (aligned)

fclose(output);                            % close file
```

Run your program `nw3.m` again and check that the file `nw3-output.txt` contains the correct information.

[1] These can also be found in the file `print_to_file.m` from the archive Practical1.zip.

Hand in:

- a concise report in PDF format (generated by LaTeX), in which:

  a. you describe, for each part of every assignment, how you arrived at the solution,
  b. you answer all the questions posed in the assignments.

  Write at the top of the first page of your report: "**Introduction to Scientific Computing, Practical 1**", followed by your name(s), student number(s), and the date when you hand in the report.

- your Matlab programs `nw1.m`,..., `nw3.m`.

Since you work in pairs, you need to include information in your report about your individual contributions. For each assignment, indicate the contribution (mention percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report).

All files have to be handed in as **a single archive** (called `YourName.zip`), where "YourName" is the concatenation of your last names (or your last name, if you work individually). See Nestor for the address to which you have to send the archive.

Practical 2: Cellular Automata

In this practical you will implement a few algorithms for Cellular Automata in Matlab.

Assignment 1 Cellular Automata with "majority voting" (30)

a. Inspect the Matlab M-file `ca.m` from the archive Practical2.zip in Nestor. The appendix on page 13 gives a listing of this M-file. As initial pattern a random distribution of living and dead cells is created by the Matlab call `A=rand(n,n)<p`. Here `A` is the matrix of cells, `n` the number of cells in the horizontal and vertical direction, and `p` the probability that a cell is alive (in the code we have set `n`=64, `p`=0.7). The elements of `A` are 0 (cell is dead) or 1 (cell is alive). Around the matrix `A` we add boundaries with a thickness of one cell with value 0, resulting in a `(n+2)` by `(n+2)` matrix `A1`.

   Study the code and the comments. If necessary, consult the Matlab tutorial of the first practical, or the `help` function of Matlab.

b. Run the program `ca.m` in Matlab. The initial fraction of living cells is computed by the Matlab function `sum` and printed on the screen (strictly spoken it is not necessary to compute `abs(A)` because the elements of `A` are non-negative in our case, but this form will be convenient later in this assignment).

   Also, through the Matlab command

   ```
   imHandle = imagesc(A,[0 1]);
   ```

   an image is displayed on your screen with the initial pattern. Here `imHandle` is a so-called "function handle" by which the function `imagesc` can later be called indirectly; the second argument of `imagesc` is the value range of the matrix `A`, in this case `[0 1]`, because `A(i,j)` only takes the values 0 or 1. Also, the initial pattern is saved as a PNG image that is stored in your current directory. Check that the images on the screen and in the PNG image agree. In the file name of the PNG image the values of `n` and `p` are included, as well as the generation number `gen` (in the initial situation `gen=1`). This is convenient when you carry out several experiments with different values of these parameters.

c. Load the file `ca.m` in an editor, and add the missing code. Call your new program `ca1.m`. This should compute the successive generations via majority voting (see Section 3.2 of the syllabus).

   The maximal number of generations is 100. The algorithm should terminate earlier (that is, before 100 generations have passed) when no more change occurs. As measure for change you can take the number of cells of which the value is different in two successive generations.

   Display the pattern in each successive generation, that is, the corresponding matrix, say `A_new`, on the screen by including the following Matlab commands in your program `ca1.m`:

   ```
   set(imHandle, 'CData', A_new);
   drawnow;
   ```

   This has the effect that the image with "function handle" `imHandle` (see above) is updated, without calling `imagesc` again (which would be very inefficient). Instead of `drawnow` you may also use the command `pause(t)` to let Matlab pause during `t` seconds before displaying the next pattern (`t` is a non-negative real number).

Compute the fraction of living cells in the final state (the values of initial and final fractions have to be included in your report), and again save the final pattern of living and dead cells as a PNG file, where now `gen` is the generation number after termination of the algorithm.

d. Repeat the experiment with the value $p = 0.2$. Compare the result with that for $p = 0.7$ and give an explanation for the observed difference.

Assignment 2 Cellular Automata with "majority voting" and three states (40)

As an extension of the previous assignment we now consider the case that a cell has not two, but three states: living, dead, and sleeping. We encode this as follows:

$$A(i,j) = \begin{cases} 2 & \text{if cell}(i,j) \text{ is living} \\ 1 & \text{if cell}(i,j) \text{ is sleeping} \\ 0 & \text{if cell}(i,j) \text{ is dead} \end{cases}$$

The transition rule is now as follows:

- if one of the states S (living, sleeping, or dead) of the 9 cells in the 3×3 neighbourhood[2] of cell (i,j) in generation n has the *absolute majority*, then cell (i,j) will be in that state S in generation $n+1$. (We say that a state S has the absolute majority when at least 5 of the cells are in state S.)
- if none of the states of the 9 cells in the 3×3 neighbourhood of cell (i,j) has the absolute majority in generation n, then the state of cell (i,j) remains the same.

As boundary condition we again assume that cells outside the domain are dead.

a. Copy your program `ca1.m` to the file `ca2.m` and adapt this program so that successive generations are computed via majority voting for cells with three states, as indicated above. The initial state is now described by two parameters p and q, where p is the fraction of living cells and q the fraction of sleeping cells (the fraction of dead cells is then $1-p-q$).

   Replace `imname='random'` by `imname='random3'`.

   Matlab tips:

   - The initialisation of the matrix A can be carried out in Matlab by the following lines:

```
R=rand(n,n);              % matrix with random values between 0 and 1
A=(R>1-p-q)+(R>1-p);      % matrix A with random values 0,1,2
```

   - The initial fractions of cells with value k ($k = 0, 1, 2$) in the $n \times n$ matrix `A` can be computed by the Matlab command: `sum(sum(abs(A==k)))/n^2`.
   - the Matlab command `nbh=A(i-1:i+1,j-1:j+1)` produces a 3×3-matrix `nbh` with the 9 values of `A` in the neighbourhood of `(i,j)`.
   - In the call of `imagesc` you have to adapt the range.
   - In the call of `imwrite`, `A` must first be linearly rescaled so that the values are in the range between 0 and 1.

[2] Cell(i,j) included.

b. Run your program with the following two parameter combinations:
(i) $p = 0.33, q = 0.33$; (ii) $p = 0.2, q = 0.7$.
Compare the results and give an explanation of the differences you observe. Also, save the initial and final configurations in both cases as PNG images, where the values of `n`, `p`, `q`, and `gen` are included in the file name of each image.

Assignment 3 Cellular Automata with "Game of Life" rule (30)

a. In the Matlab file `gosper_n=64.mat` from the archive Practical2.zip a matrix of zeroes and ones has been saved in binary format. This matrix contains the pattern of the "Gosper glider gun" (Figure 3.6 of the syllabus), extended by zeroes to a 64×64 matrix.

Copy the file `ca.m` to `ca3.m`.

In `ca3.m`, replace the lines

```
imname='random';
A=rand(n,n)<p;
```

by

```
imname='gosper';
inputfile = [imname,'_n=',int2str(n),'.mat'];
load (inputfile,'A');
```

After these lines have been executed the matrix `A` will contain the initial pattern of zeroes and ones of the "Gosper glider gun".

Change the code in the file `ca3.m` so that the successive generations are computed via the *Game of Life* rule (see Section 3.3 of the syllabus). Display the pattern in each generation on the screen. (N.B.: in the draw and write routines of `ca.m` living cells are represented by white pixels and dead cells by black pixels; in the syllabus it is the other way around.)

Write the patterns in the generations 1, 16, 46 and 76 to PNG files as in Assignment 1 (so the first image will get the file name `gosper_n=64_gen=1.png`, etc.).

b. Run your code and study the patterns that you observe on the screen. Include your observations in your report. Also investigate the four PNG images and describe whether the patterns in these images agree with the theory from Example 3.8 of the syllabus.

Hand in:

- a concise report in PDF format (generated by LaTeX), in which:

  a. you describe, for each part of every assignment, how you arrived at the solution;
  b. you include all the 12 PNG images that you produced in the experiments (four each for the Assignments 1, 2, 3);
  c. you answer all the questions posed in the assignments.

  Write at the top of the first page of your report: "**Introduction to Scientific Computing, Practical 2**", followed by your name(s), student number(s), and the date when you hand in the report.

- the files `ca1.m`, `ca2.m` and `ca3.m` with your implementations.

Since you work in pairs, you need to include information in your report about your individual contributions. For each assignment, indicate the contribution (mention percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report).

All files have to be handed in as **a single archive** (called `YourName.zip`), where "YourName" is the concatenation of your last names (or your last name, if you work individually). See Nestor for the address to which you have to send the archive.

Appendix 1: the skeleton program `ca.m`

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Introduction to Scientific Computing - WBCS14003                              %
%                                                                               %
%  Simulate spatial pattern formation in Matlab                                 %
%  via cellular automata                                                        %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;                                    % clear the command window
close all                               % close open figure windows
clear all;                              % remove items from the workspace

n=64;                                   % number of cells horizontally/vertically
p=0.7;                                  % probability that a cell is alive
max_gen=100;                            % maximal number of generations

% Initialize matrix A
imname='random';                        % name of the pattern
A=rand(n,n)<p;                          % n x n matrix A with random zeroes/ones
                                        % expected number of living cells is p (for n large)

% Print the initial fraction of living cells on the screen
fprintf('initial fraction of living cells=%f\n',sum(sum(abs(A)))/n^2);

% Display the initial pattern of living and dead cells as an image
% The living cells are white, the dead cells black.
figure;                                 % open a figure window
imHandle = imagesc(A,[0 1]);            % display the matrix A as an image.
                                        % The value range of A is [0 1].
colormap(gray);                         % set a gray scale color table

% Write the image to a PNG file
gen=1;                                  % current generation number
imfile = [imname,'_n=',int2str(n),'_p=',num2str(p),'_gen=',int2str(gen),'.png'];
imwrite(A, imfile);                     % write A

% Expand matrix A to matrix A1 because of the extra borders needed
A1=zeros(n+2,n+2);                      % initialise (n+2)x(n+2) matrix with zeroes
A1(2:n+1,2:n+1)=A;                      % Insert matrix A in matrix A1

% Now compute the successive generations via the majority rule.
% The algorithm should terminate as soon as no more differences
% occur between successive generations.

% Here comes your code ....
```

Practical 3: The three-body problem

In this practical you will implement a simulation of the three-body problem involving the Sun, the Earth, and the Moon in Matlab. As preparation you will first implement a two-body simulation. Note that the following units will be used:

- astronomical units (au) for distances/positions,
- days for time, and
- 10^{24} kg for mass.

Assignment 1 Implementing a two-body simulation

Tip: to plot the trajectory of a body, the Matlab function `squeeze` can come in handy!

a. Extract Practical3.zip to a separate folder and set this folder as the current folder in Matlab. Now open `simulate2.m` and `computeAccelerations2.m`.

b. In the function `computeAccelerations2.m`, implement the scheme for computing the accelerations $\vec{a}_i$ as described in Section 4.5 of the reader, for just two bodies, using the masses and positions given as arguments to the function.

c. In the function `simulate2` finish computing the derivatives (the right-hand-sides of the equations (4.12) and (4.13) of the reader), updating (Eqs. (4.15) and (4.16) of the reader), and setting the step size and number of steps. You will have to experiment with the step size to see what works (the step size should be small enough so that going even smaller does not really change the results much, while still being large enough to ensure reasonable simulation times).

d. Create a script `nbody_1.m` that plots the result of running `simulate2` with a `duration` of 10 *years* using just the first two (X/Y) coordinates, and include the plot in your report. The "Sun" should be plotted in red, and the "Earth" in blue.

   Take as their masses `[1.988544e6 5.97219]`. Initialize the positions of the two bodies to the origin and `[0.983236 0 0]'`, respectively (considering the Sun to be in the center of the coordinate system). The velocities can be initialized to a zero vector and `[0 0.0174939 0]'`, respectively. Pay attention to rows vs. columns (each body should correspond to one column).

e. In the same script, use `detectOrbitalPeriods.m` to detect the orbital period of the Earth relative to the Sun over the 10 year period you simulated (and report the result). (The orbital period is the period between successive closest alignments to the initial position.) Explain how long you expect the orbital period to be, and what is unrealistic about the result.

Assignment 2 Implementing symplectic integration

In the previous exercise, we saw that the simple integration scheme used led to unrealistic behaviour. This is because it can be shown that the scheme does not respect (for example) the physical principle of conservation of energy. However, it is possible to make some small changes that at least improve the situation.

a. Save `simulate2.m` as `simulate2sym.m` and modify `simulate2sym.m` so that in the loop the velocities are updated first, and then the positions are updated using the new velocities. Note that you should also modify the name of the function to match the file name at the top!

b. Now save `nbody_1.m` as `nbody_2.m`, and make it use `simulate2sym` rather than `simulate2`.

c. Show a plot and report on the orbital period in the same way as before, but now discuss how the orbital periods have changed compared to the previous exercise.

Assignment 3 A more realistic scenario

In this exercise, we will add a third body (the Moon), and use a more realistic initialization.

a. Based on `simulate2sym.m` and `computeAccelerations2.m`, create a `simulate3sym.m` and `computeAccelerations3.m` to simulate a three-body problem. Do not forget to change the size of the position and velocity arrays in `simulate3sym`!

b. Save `nbody_2.m` as `nbody_3.m`, and make it use `simulate3sym` rather than `simulate2sym`.

   For the initialization, we will use the HORIZONS web tool. Go to `http://ssd.jpl.nasa.gov/horizons.cgi` and change the current settings as follows:

Ephemeris Type:	VECTORS
Target Body:	Sun [Sol] [10]
Coordinate Origin:	Solar System Barycenter (SSB) [500@0]
Time Span:	as desired
Table Settings:	defaults
Display/Output:	default (formatted HTML)

   Now, click on "Generate Ephemeris". As described in the output, you now have positions and velocities for the selected target body **Sun [Sol] [10]** with respect to the solar system barycenter (in the units used in this exercise). Use the tool two more times to generate data for **Earth [Geocenter] [399]** and **Moon [Luna] [301]**. Pick a time stamp, and use the corresponding positions and velocities as initialization for your simulation. Report the time stamp you have chosen in your report!

c. The HORIZONS tool used in the previous step will also give you the correct mass for the Moon (mind the units!) in the header before the actual trajectory data; add it to the list of masses.

d. Change the plot in `nbody_3.m` of the X/Y positions of the bodies so that it includes the moon (make its trajectory black). Also include a plot of the trajectory of the Moon relative to Earth.

e. Add code to investigate the orbital period of the Moon relative to the Earth (you can again use `detectOrbitalPeriods.m`). Note that for both types of orbital periods, it is important to consider the trajectories of the Moon/Earth *relative to* the Earth/Sun!

f. Show the plots and orbital periods in your report, and comment on any changes compared to the previous exercise (did the orbital period of the Earth become more or less

regular? more or less accurate?[3]). Also comment on the shape, regularity, and accuracy of the orbital period of the Moon around the Earth.

Hand in:

- a concise report in PDF format (generated by LaTeX), in which:

  a. you describe, for each part of every assignment, how you arrived at the solution;
  b. you include all the images that you produced in the experiments;
  c. you answer all the questions posed in the assignments.

  Write at the top of the first page of your report: "**Introduction to Scientific Computing, Practical 3**", followed by your name(s), student number(s), and the date when you hand in the report;

- your Matlab programs `simulate2.m`, `simulate2sym.m`, `simulate3sym.m`, `computeAccelerations2.m`, `computeAccelerations3.m`, `nbody_1.m`, `nbody_2.m`, `nbody_3.m` (running these last two should generate *all* of the results shown in your report).

Since you work in pairs, you need to include information in your report about your individual contributions. For each assignment, indicate the contribution (mention percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report).

All files have to be handed in as **a single archive** (called `YourName.zip`), where "YourName" is the concatenation of your last names (or your last name, if you work individually). See Nestor for the address to which you have to send the archive.

[3]For a ground truth, you can take the "Sidereal year" reported by the HORIZONS tool in the header preceding the data you generated.

Practical 4: Simulation of reaction-diffusion systems

In this practical you will implement a slight generalization of one of the reaction-diffusion systems shown in the reader and examine the effect of certain parameters. More precisely, you will be implementing this system:

$$\frac{\partial f(x,y,t)}{\partial t} = D_f \left(\frac{\partial^2 f(x,y,t)}{\partial x^2} + \frac{\partial^2 f(x,y,t)}{\partial y^2} \right) + \phi_f(f(x,y,t), g(x,y,t)), \tag{1}$$

$$\frac{\partial g(x,y,t)}{\partial t} = D_g \left(\frac{\partial^2 g(x,y,t)}{\partial x^2} + \frac{\partial^2 g(x,y,t)}{\partial y^2} \right) + \phi_g(f(x,y,t), g(x,y,t), \beta(x,y)), \tag{2}$$

$$D_f = 1/4, \qquad D_g = 1/16,$$

$$\phi_f(f,g) = \frac{s}{16}(16 - f \cdot g), \qquad \phi_g(f,g,\beta) = \begin{cases} \frac{s}{16}(f \cdot g - g - \beta) & \text{if } g > 0 \\ 0 & \text{otherwise.} \end{cases}$$

First you will implement 1D diffusion, and then the full 2D reaction-diffusion system.

Assignment 1 Implementing 1D diffusion (20)

a. Extract Practical4.zip to a separate folder and set this folder as the current folder in Matlab. Now open `simulate1D.m`.

b. In the function `simulate1D` finish setting the step size and number of steps (corresponding to the variables Δt and N of the reader), assuming unit spacing in the x direction. Note that `num_steps*Delta_t` should equal duration and that you should take the stability criterion Eq. (5.9) from the reader into account in choosing the spacing `Delta_t`. To ensure smooth results, make it about half as large as the stability criterion allows.

c. Next, implement the time derivative by setting it equal to D times the second-order spatial derivative, that is, the right-hand side of Eq. (5.7) of the reader (do not loop over the array elements, implement the derivative at all grid points in a single line using array operations). At the first position in the array the f_{i-1} term is clearly undefined, as $i-1$ would point to a position *before* the first element in the array. This is a ubiquitous problem with implementing differential equations in a computer and is dealt with using so-called "boundary conditions" that determine what happens when you step outside the boundaries of the data. There are many variations, but here we will use so-called *periodic boundary conditions*. The idea is that when you step out of the data on the left you re-enter on the right, so that the element to the left of the first element is equal to the last element. An analogous observation holds for the f_{i+1} term.

d. Finish `simulate1D` by making sure `F` gets updated properly.

e. Create a script `rd_1.m` that plots the results of running `simulate1D` with a `duration` of 10 and `D` set to 4 and 8 (so you get two curves in *one* plot). Make sure curve corresponding to $D = 4$ is blue, and the one corresponding to $D = 8$ is red. Put the plot in your report (make sure the tick labels are visible, so do not crop away the axes for example).

Assignment 2 Implementing 2D reaction and diffusion (40)

Important: when saving images you should make sure that the values are between zero and one. For the examples in the reader, the values were scaled by 1/16. You are welcome to use another scaling factor, but be sure to use a fixed factor to ensure that different images are comparable.

a. Open `reactionDiffusionD.m`, this function will eventually compute (the discrete version of) the right-hand sides of Eqs. (1) and (2), while `simulate.m` will eventually take care of the time evolution.

b. In the function `reactionDiffusionD`, finish the computation of the diffusion terms using Eq. (5.13) from the reader, assuming unit spacings in both the x and y directions, and with periodic boundary conditions (to simplify this, take a look at the Matlab function `circshift`).

c. Next, implement the reaction terms. Think of some solution for implementing the condition in ϕ_g without looping over the array elements (hint: what does `A(B<=0)` do?). Note that initially you can set s equal to 1.

d. Open `simulate.m` which should compute the time evolution and finish it (make use of the function `reactionDiffusionD` you already implemented). Note that here it is not as easy to give the same kind of stability criterion as in Eq. (5.9) in the reader, so you just have to experiment in choosing the spacing `Delta_t` (stay below one). Smaller time steps give more stable and more accurate results, but also lead to longer running times (so try to find a value that strikes a nice compromise). Note that you should make sure that when G becomes negative it is clamped to zero (after all, negative concentrations make no sense, and they can cause issues), again try to use array operations rather than loops for this.

e. Create a script `rd_2.m` that first initializes a 100×100 array `beta` using `randn` (mean 12, standard deviation 0.1), and then runs the simulation. You should use `imwrite` to save `beta` (you only need to include this in your report once).

f. Tune the number of steps in `simulate.m` so that the result is essentially stable. Use `imwrite` to save `F` and `G` for various numbers of steps and include the results in your report. Your report should show at least four results: the result after zero (or very few) steps (this should be almost featureless); an intermediate result that shows some structure starting to form; a mostly converged result; and one more result (with a lot more steps) to confirm that the result had indeed (almost) converged. For each result, show both `F` and `G`, and report the number of steps corresponding to each result.

g. Experiment with the parameter s. What effect does it have if you decrease s?

Assignment 3 The influence of β (40)

For this exercise you may want to use a value for s less than 1.

a. Create a script `rd_3.m` that first creates a 100×100 array `betaOffset` initialized using `randn` (zero mean and unit standard deviation).

b. Let the script run the simulation several times, each time with all parameters exactly the same, except that each time `beta` should be set to `12+sigma*betaOffset`, for varying values of `sigma` (between 0.01 and 10). Save the results. (Since `betaOffset` is initialized just once, all results will be based on a `beta` with the same general pattern, the only difference is the magnitude of the pattern.)

c. Describe the results of part b (showing several clearly different results). Also discuss similarities between images with varying `sigma` (if you kept `betaOffset` the same throughout this experiment, there should be clear similarities).

d. Now repeat the previous experiment, but with `betaOffset` set using `checkerboard.png` (be sure to use `im2double` to convert the image to the proper range and type). Briefly discuss the results (and illustrate with some images of the results).

e. Do the experiment one more time, but now using some other image (photograph, vector drawing, animation frame, etc.). Try to get a nice result and include it in your report (detailing the original image you used and what values you used for the various parameters).

Hand in:

- a concise report in PDF format (generated by LaTeX), in which:

  a. you describe, for each part of every assignment, how you arrived at the solution;
  b. you include all the images that you produced in the experiments;
  c. you answer all the questions posed in the assignments.

  Write at the top of the first page of your report: "**Introduction to Scientific Computing, Practical 4**", followed by your name(s), student number(s), and the date when you hand in the report;

- your Matlab programs `simulate1D.m`, `reactionDiffusionD.m`, `simulate.m`, `rd_1.m`, `rd_2.m`, `rd_3.m` (running these last three should generate *all* of the results shown in your report).

Since you work in pairs, you need to include information in your report about your individual contributions. For each assignment, indicate the contribution (mention percentages) for each of you in terms of tasks performed (program design, program implementation, answering questions posed, writing the report).

All files have to be handed in as **a single archive** (called `YourName.zip`), where "YourName" is the concatenation of your last names (or your last name, if you work individually). See Nestor for the address to which you have to send the archive.