Help & Documentation*

Codes of this library is written by of Hydro Group members.

1 Dynamic Model

The Dynamic Model excel file is provided here and its use is explained here.

1. `Fahrenheit_to_Celsius(temp_cel)` converts Fahrenheit temp. to Celsius. Celsius is the proper form of temp. used in the model.

   Input:

   - `temp_cel` A real valued temp. or a column of a data frame containing the temp in the Celsius format.

   output

   - A real valued temp. in Fahrenheit or a column of data frame.

2. `initiate_data_frame(col_names, init_temp_c, const)` Creates a new data frame of size 2×13 and fills in the cells which are to be used by dynamic model. It corresponds to rows 11 and 12 of the model given in the excel file.

   Input:

   - `col_names` The names of columns of the data frame to be used in the model.

     `col_names` = [date, time, temp_c, temp_k, ftmprt, sr, xi, xs, ak1, Inter-S, Inter-E, delt, Portions]
   - `init_temp_c`. Initial temps. corresponding to cells C11 and C12 of the excel file. `init_temp_c` = $(15, 12)$.
   - `const` An object of the class `constants` containing constants of the model. They are given below and in the D1 through D8 cells of the excel file.

*Name of the programmer of functions is written in the code for debugging purposes. If you see any flaw, please contact the programmer.

Table 1: `const` object

e0	e1	a0	a1	slp	tetmlt	aa	ee
4.15E+03	1.29E+04	1.40E+05	2.57E+18	1.6	277	a0 / a1	e1 - e0

output: A data frame of the following form. The numbers in the `sr` column in the table 2 are long numbers I did not put them in there.

Table 2: initial data frame to construct the model with.

date	time	temp_c	temp_k	ftmprt	sr	xi	xs	ak1	Inter-S	Inter-E	delt	Portions
None	None	15	288	16.93	-	1	.81	.09	0.00	.07	0.00	0
None	None	12	285	12.44	-	1	1.11	.06	.07	.13	0.00	0

3. `fill_in_the_table(given_table, const)` This function takes the `const` object and `given_table` as input and runs the model to fill in the proper information that we need to compute the Chill Portions which is the ultimate goal of the model.

   input:

   - `given_table` Is the data frame that contains the first two rows, like the one given by 2 and the first three columns, from row 3 to the end, are provided by datalogger and are read off the disk. Anything from the column **temp_k** onward is computed and filled by this function.
   - `const`: The object containing constants of the model mentioned before.

   output: A complete table that has the Chill Portions for the data of out orchard.

4. `dynamic_model(path_to_data, col_names, init_temp_c, const)`

   This function takes the path of the file we wish to compute the Chilling portions for, along with other inputs that we have already mentioned before, and produces the Chilling Portions.

   input:

   - `path_to_data`: the path to the data location on the disk.

- `col_names`: Name of the columns of the data frame, like mentioned before. These names has to be exact, because they are used for computations in the model.

  NOTE: These data should have temp. in Celsius. And it is assumed the first thee columns are **`date`**, **`time`** and **`temp`** respectively.

- `init_temp_c` Initial temp. as mentioned before.
- `const`: An object containing the constants of the model.

`output:` A data frame containing all information we need. (Shall I change this so that it just gives the `Portions`?)

2 Codling Moth Generations

In this document the terms layer, depth and slice location are used interchangeably.

2.1 Get Image Matrices Out of Profile Data

1. `extract_sliceLocation_names(all_profile, no_slices=10)`

   `input`:

   - `all_profile` List of all profiles which is output of `load_dcm_profiled(.)`
   - `no_slices` Number of different spacial images of a given liver.

   `output`: Ordered list of ID's of slices.

   Each profile read off the disk has different information in it like patient name, date of images taken, etc. One of the items stored in a profile is ID of an slice, e.g. -140, -165, etc. The variable corresponding to slice ID is called "SliceLocation" and is reachable as follows: "profile.SliceLocation" for a given "profile".

   Number of slices in our data is 10. Hence, the function uses no_slices=10 as a default if it is not provided by the user.

2. `matrix_of_all_times(all_profiles, depth, no_time_steps, image_dimension)`

   It goes through all data and returns a 3D-matrix of images of the same depth and different times. The size of the matrix is (no_time_steps, image_dimension, image_dimension)

   input:

   - `all_profiles` which is the output of `load_images(.)`
   - `depth`: The target layer (currently we have 10 layers.)
   - `no_time_steps`: Total number of time steps the images are taken. (currently 59).
   - `image_dimension` is the dimension of images we have. (currently 512 x 512). maybe we can skip this, inside the function we can look at one image, extract its size, and use it.

   output: `image_matrix`. 3D matrix of images of a certain layer taken at different times.

2.2 Extracting Submatrices

1. `extract_submatrix_center(image_matrix, center_coor, margin_size)`

   This function takes a 3D matrix of size (times, image_size, image_size) and returns sub-matrix of size (times, sub_image_size, sub_image_size).

   input:

   - `image_matrix` is the matrix of images of the same depth and all times.
   - `center_coor` is index or coordinate of the center pixel of the target sub-matrix. So, if the center entry of the sub-matrix we want is A_{ij}, then its coordinate is [i, j].
   - `margin_size` is the size of the margin at each side of the center pixel. So, for example, if **`margin_size`** = 1, then we will extract submatrix of size 3x3.

   output: 3D sub-matrix of size **`(all_time, m, m)`** where $m = 1+2\times margin\_size$

2. `extract_submatrix(image_matrices, upper_left, sub_size)`:

   input:

   - `image_matrices` is a 3D matrix of images taken at different times of the same depth.
   - `upper_left` is the coordinate of the upper left pixel of the sub-image we want to look at. The user has to think about the indexing like Python.
   - `sub_size` is the size of the sub-matrix we want to look at. It is assumed the sub-matrix is square.

   output: a matrix of size (time_steps, sub_size, sub_size).

2.3 Aggregation

1. `aggregate_2D(matrix, sub_matrix_dim)` is a function that produces a matrix by non-local means

   input:

   - `matrix` is a square matrix of an image to be reduced.
   - `sub_matrix_dim` is number of rows (= number of columns) of the blocks in the matrix. For example if matrix is of size 36 x 36, and we want to look at tiles of size 2 x 2, there will be 9 blocks of size 2 x 2

   output: a matrix of size **`m x m`**, where **`m = sqrt(no_blocks)`**.

2. `aggregate_3D(matrix, sub_matrix_dim)` is a function that takes in the 3D matrix of images of the same depth (depth $\in \{1, 2, \ldots, 10\}$) taken at different times (time $\in \{1, 2, \ldots 59\}$)

3 Other functions

1. `rgb2gray(image)` takes an RGB image as an input and returns its gray scale image using MATLAB's RGB coefficients (0.2989, 0.2989, 0.114).