

433-385 Modeling, Analysis and Visualisation

Project 2, 2004

Due date: Monday 1st November, 5pm

This project covers the second half of the course: Bioinformatics and Medical Imaging. Please make sure to read the notes on submission and assessment at the end of this document.

All programs should read input data from the specified input files and formats.

A. Bioinformatics

1. Edit Distance Matrix [3 marks]

Write a program *dpa.pro* which computes and displays the edit distance matrix between two strings according to the dynamic programming algorithm (DPA):

$$D(i, j) = \min[D(i, j-1) + 1, D(i-1, j) + 1, D(i-1, j-1) + t(i, j)]$$

where $t(i, j) = \begin{cases} 0, & \text{if } S_1(i) = S_2(j) \text{ (match)} \\ 1, & \text{if } S_1(i) \neq S_2(j) \text{ (mismatch)} \end{cases}$ $D(i, 0) = i, \quad D(0, j) = j$

Your output (in a window called “edit distance matrix”) should look like the following:

		A	A	B	A
	0	1	2	3	4
B	1	1	2	2	3
A	2	1	1	2	2
B	3	2	2	1	2

By varying window and character sizes (using *charsize*) make sure your program can cope with arbitrary strings of length up to 20 characters (i.e. before readability and window size becomes an issue). The program should accept the two string inputs (on separate lines) from a file called *dpa_input.dat*. Read in data from the file as follows:

```
string1='' & string2='' ;define vars as empty strings
openr, lun, 'dpa_input.dat',/get_lun
readf, lun, string1
readf, lun, string2
close,lun & free_lun, lun
```

Input file name: *dpa_input.dat*

Input file format:

```
string1
string2
```

2. Needleman-Wunsch Sequence Alignment [5 marks]

Write a program *nw.pro* (using *dpa.pro* as a starting point) which computes an optimal alignment of two sequences based on the Needleman-Wunsch algorithm for a user-determined edit weight set (d,r,m) as outlined in the lectures. The program should proceed in four stages:

- Compute the edit distance matrix according to the given edit weight set (d,r,m)
- Assign pointers to all edit operations using the notation $(\backslash,|,-)$ as in the lecture example
- Carry out the trace-back and thus compute an optimal alignment
- Write the alignment to the screen display using the notation given in lecture examples

After computing the traceback the cells along an optimal path should be coloured red (or some other colour to distinguish the path). There may be more than one optimal alignment – in this case your program should calculate and output one alignment only. The output (in a window called “Needleman-Wunsch”) should look like the following:

		w	r	i	t	e	r	s
	0	-1	-2	-3	-4	-5	-6	-7
v	1	\ 1	\ -2	\ -3	\ -4	\ -5	\ -6	\ -7
i	2	\ 2	\ 2	\ 2	- 3	- 4	- 5	- 6
n	3	\ 3	\ 3	\ 3	\ 3	\ - 4	\ - 5	\ - 6
t	4	\ 4	\ 4	\ 4	\ 3	\ - 4	\ - 5	\ - 6
n	5	\ 5	\ 5	\ 5	\ 4	\ - 4	\ - 5	\ - 6
e	6	\ 6	\ 6	\ 6	5	\ 4	\ - 5	\ - 6
r	7	\ 7	\ 6	\ - 7	6	5	\ 4	- 5

w	r	i	-	t	-	e	r	s
v	-	i	n	t	n	e	r	-

By varying window and character sizes (using *charsize*) make sure your program can cope with arbitrary strings of length up to 20 characters (i.e. before readability and window size becomes an issue). The program should accept the two string inputs and the weight set parameters d, m, r (on separate lines) from a file called *nw_input.dat*.

Question A1: Investigate changing the weight set (d,m,r) to positive and negative weights and describe the effect of these changes on the optimal alignments.

Question A2: Are the edit weights (d,m,r) really independent?

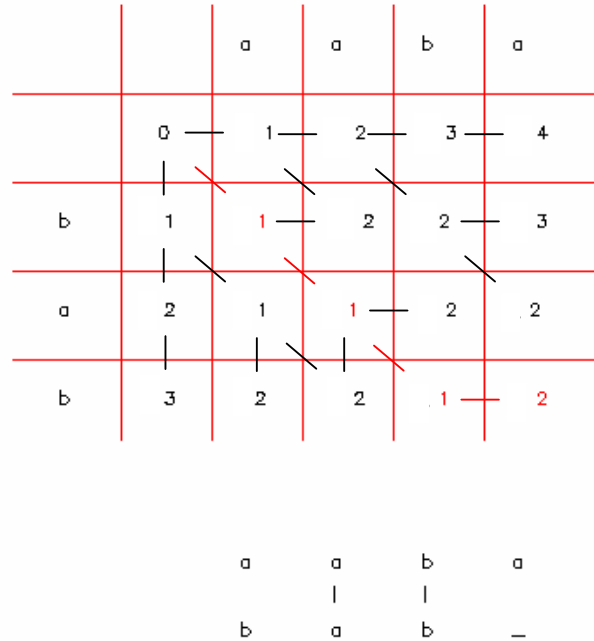
Input file name: *nw_input.dat*

Input file format:

```
string1
string2
d
m
r
```

3. Traceback with lines [2 marks]

Write a program *lines.pro* which is identical to *nw.pro* in function except that for the pointers draw lines (using the line drawing command) instead of the notation (\,|,-). Your output should look something like the following:



The pointers for the traceback along the optimal alignments should be red.

The program should accept the two string inputs (on separate lines) from a file called *lines_input.dat*.

Input file name: *lines_input.dat*

Input file format:

```
string1
string2
d
m
r
```

B Medical Imaging and Visualisation

This section requires you to complete a number of different visualisations of a brain imaged using the technique of Magnetic Resonance Imaging (MRI) using a number of the built-in IDL visualisation routines. You will be required to display images of this brain/head as slices of the raw data, in ray projection, as isosurfaces, as cut-away isosurfaces and as semi-transparent surfaces showing detail of structures beneath (e.g. we have placed an artificial tumour in the dataset).

1. Reading in and displaying the data [2 marks]

Firstly, you will need to read in the data file *mridata.img* to an array in IDL. The dimensions of the volume are 181 pixels by 217 pixels (1 pixel = 1mm real space) by 36 slices (*x*, *y*, *z*). Use the `read_binary` function to read in this file with the `data_dims` keyword to specify the dimensions of your data set, e.g.:

```
data = read_binary('mridata.img', data_dims=[181,217,36])
```

Write a program called *brain1.pro* which will read in the brain data, display given data slices in axial and sagittal view and animate these views (as specified below).

First, using `tv` display an *axial* slice (top view) of the brain (i.e. `data[*,* ,axial_slice_number]` where `axial_slice_number` is between 0 and 35) in a window entitled 'Axial'. Use the `congrid` function to stretch the dataset from 36 to 181 in the *z* direction (corresponding to data slice thickness = 5mm).

```
volumedata = congrid(data,181,217,181)
```

Now, using `tv` display an *axial* slice (now `axial_slice_number` is between 0 and 180) of the brain in a window entitled 'Axial', a *sagittal* slice (side view) in a window entitled 'Sagittal' (`sagittal_slice_number` is between 0 and 216) and a *coronal* slice (front view) in a window entitled 'Coronal' (`coronal_slice_number` is between 0 and 180). Using `xyouts` write the slice number to the display for each view. As far as you are able, remove the "chopsticks" artefact at the top of the head (and for all that follows) from the dataset.



Use the `XINTERANIMATE` command (omit `/showload`, and set frame rate to `frame_rate` as given in the input file) to animate all images of the dataset sequentially in one animation. The animation should run in the following order: axial, sagittal and coronal. (You may need to use the `reform` command – ask the tutors).

Input file name: *brain1_input.dat*

Input file format:

```
axial_slice_number
sagittal_slice_number
coronal_slice_number
frame_rate
```

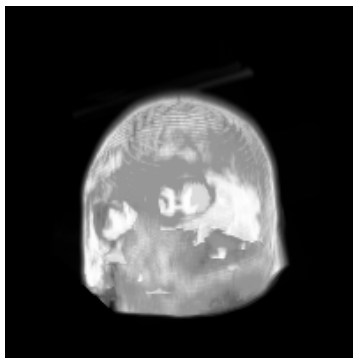
Question B1 *What is the range of (x,y,z) pixel values containing the tumour data?*

2. Ray casting through the brain [2 marks]

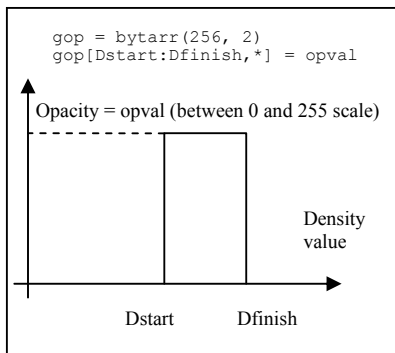
Write the program *brain2.pro* which carries out simple ray casting of the head. You'll need to set up the 3D coordinate space first by using the `scale3` procedure.

```
sz = size(data)
scale3, xrange=[0, sz[1]], yrange=[0, sz[2]], zrange=[0, sz[3]]
```

The library function `voxel_proj` should be used to perform the ray casting. Open two (~200x200 pixel) windows and show the brain visualised using the maximum intensity projection method, and using an opacity array `gop` (both visualisations computed using `voxel_proj`).



Ray casting: maximum intensity projection



Single window opacity profile



Ray casting: opacity array set to highlight the tumour

In finding suitable opacity arrays, the `profiles` routine is a useful mouse-command for determining the pixel values along a row in a given image (i.e. a particular slice as displayed in *brain1.pro*). You can rotate the direction of projection using the `t3d` procedure. eg `t3d, rotate=[30, 0, 0]` rotates the direction of view 30 degrees about the x axis. You may also need to translate the projection to keep it in the field of view using the `translate` option in ,e.g. `t3d, translate=[0.5, 0, 0]` translates the projected image half a unit in the direction of positive x. If you translate or rotate into oblivion use `t3d, /reset` to reset the view direction and location. Experiment with different colour tables to produce an image with good contrast.

Input file name: *brain2_input.dat*

Input file format:

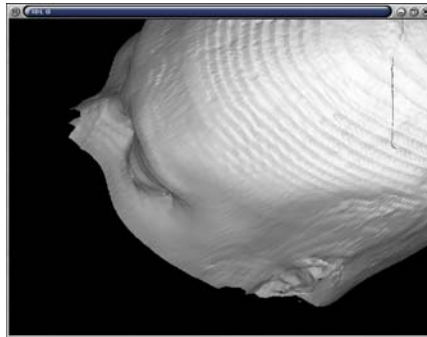
```
xrotation, yrotation, zrotation
xtranslation, ytranslation, ztranslation
Dstart, Dfinish, opval
```

3. Isosurface of the Skull and Brain [3 marks]

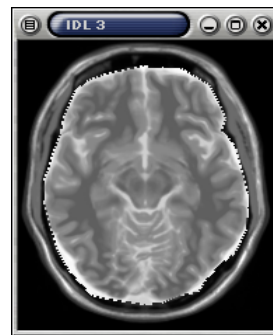
Write a program called *brain3.pro* which will render an isosurface of the skull and brain using the `isosurface` function and the `polyshade` function. Again, use the `t3d` procedure to orient the skull in a manner that shows a good selection of the surface information (e.g. facial features).

Example use of `isosurface` and `polyshade` functions:

```
isosurface, data, brain_isosurface, v, p
; where brain_isosurface is a good value for surface (you determine it)
; v & p are outputs for vertices ;and polygons returned respectively
tvsc1, polyshade(v, p, /t3d)
```



Isosurface of the skull



Using `defroi` to 'peel off' skull data

To see only the brain within the skull we need to perform a segmentation of the brain. Use the `defroi` function to coarsely cut out the brain from the skull in each image plane. Then remove any extra 'mess' outside the brain with the `search3d` function. Here are some hints to help you on this task:

brain=bytarr(181,217,36) → Define the array for the brain only data

→ loop over slices, i = 0, 35

`tvsc1, data[*,* , i]` → *Display the first slice then click round the brain...*

```
elements_brain_0_slice = defroi(181,217)
temp = data[*,* , i]
```

→ Manually select the elements in the brain for each slice

```
brain_temp = bytarr(181,217)
brain_temp[elements_brain_0_slice] = temp[elements_brain_0_slice]
brain[*,* , i] = brain_temp
```

Repeat for each slice (save when necessary to avoid losing tedious work).

Then do a 3d search around the outside of the brain for all elements less than a certain threshold.

```
elements2 = search3d(brain, 0, 0, 10, 0, 63)
```

Where (0, 0, 10) is a position in the volume array outside the brain that is also between the search window voxel values (in this case 0 and 63).

```
brain[elements2] = 0
```

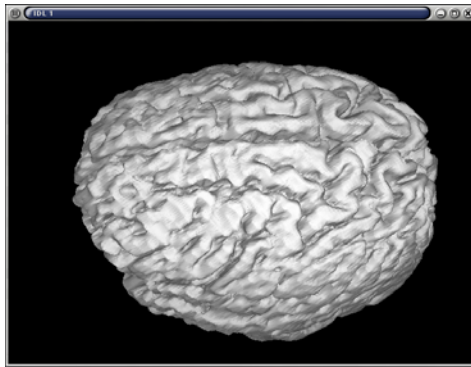
→ Sets these lower valued elements outside the brain to zero for improved contrast/noise reduction.

```
openw, lun, 'brain.img', /get_lun  
writeu, lun, brain  
close, lun & free_lun, lun
```

→ open file for output and write binary brain-only data

Then display the brain in the same manner as the skull...

After you have completed the `defroi` brain extraction and saved the data set to file `brain.img`, your final `brain3.pro` code should read in the `mridata.img` and `brain.img` files, render an isosurface of the skull and brain in separate windows and a third window with some part cut away showing the brain inside, e.g. by setting the appropriate array elements to zero and adding and subtracting arrays, cut way a quarter of the skull at the front of the head and display (as shown below).



Isosurface of the isolated
brain



Cutaway view of the
isolated brain

Input file name: `brain3_input.dat`

Input file format:

```
brain_isosurface, skull_isosurface  
xrotation, yrotation, zrotation  
xtranslation, ytranslation, ztranslation
```

4 Experiment with different lighting directions [1 mark]

The program `brain4.pro` is a copy of `brain3.pro`, but with input parameters defining the lighting model and parameters to be used for viewing. Use the `set_shading` procedure and the `light` keyword to change the lighting direction on your isosurfaces. Experiment with different lighting to create a well illuminated image. Compare the case of using Gouraud shading to the case where Gouraud shading is not used.

```
set_shading, light=[-0.5, 0.5, 0.5] → 3-element vector giving position of light source.
```

```
set_shading, gouraud=0 → Turns Gouraud shading off - default is on.
```

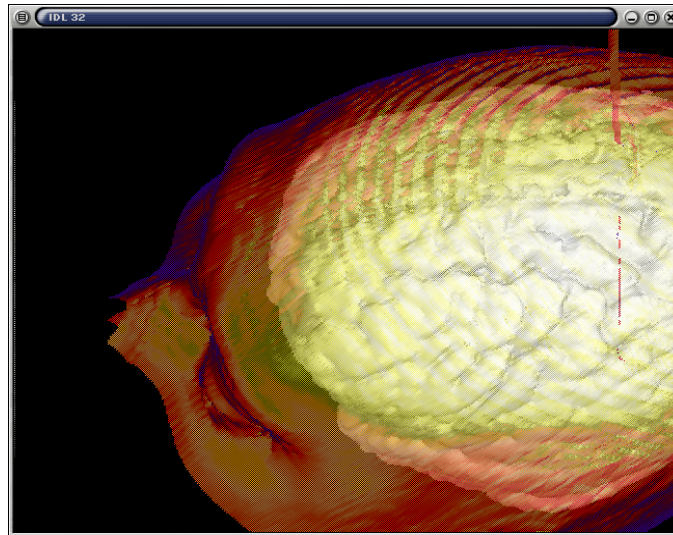
Input file name: `brain4_input.dat`

Input file format:

```
brain_isosurface, skull_isosurface
xrotation, yrotation, zrotation
xtranslation, ytranslation, ztranslation
gouraud, lightx, lighty, lightz
```

5. Semi-transparent surfaces [2 marks]

Write the program *brain5.pro* which using the technique of image averaging creates a semi transparent surface showing the brain in greyscale, surrounded by the skull (also in grayscale). Here is an example of what a typical image should look like (ignore the colour in this image):



Sample output of averaging images

Rather than attempt to interleave the images (which is quite complex) a simple way to overlay images to “see through” the data is to simply compute the averaged sum: e.g. if arrays A and B contain two isosurfaces, then for p in [0,1] simply compute and display the image given by $C = p*A + (1 - p)*B$.

Your final program should display the isosurfaces of the skull and brain in separate windows and then compute a set of images for animation which show the skull becoming increasing transparent to show the brain surface alone. These images should be animated using `xinteranimate`.

The following input for *brain5.pro* should be read in from a file called *brain5_input.dat* and listed on separate lines: viewing angle parameters, number of frames for animation, lighting model parameters.

Input file name: *brain5_input.dat*

Input file format:

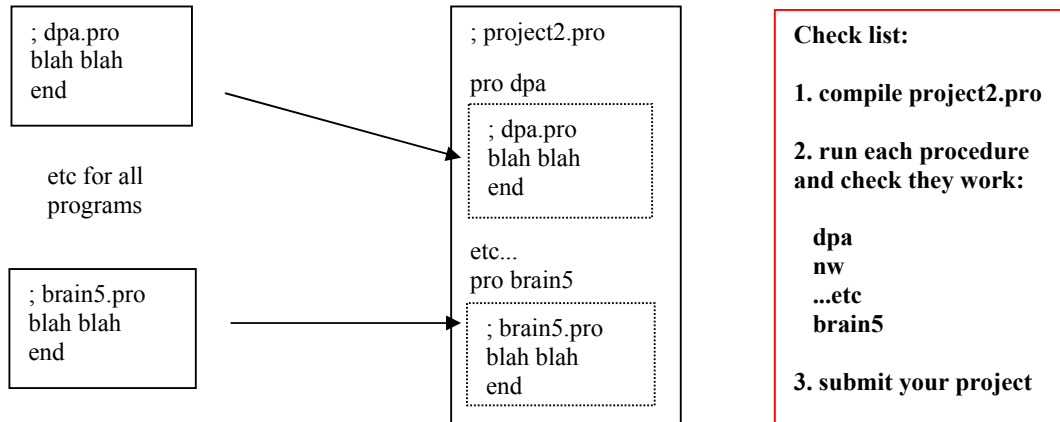
```
brain_isosurface, skull_isosurface
xrotation, yrotation, zrotation
xtranslation, ytranslation, ztranslation
number_frames
gouraud, lightx, lighty, lightz
```

Submission Procedure

The due date is Monday 1st November, 5pm.

You must submit three files: a text file *answers2.txt* with written answers to the questions (where asked), the *brain.img* file you have created and one program file *project2.pro* which contains the following programs (ready to run) as procedures: *dpa.pro*, *nw.pro*, *lines.pro*, *brain1.pro*, *brain2.pro*, *brain4.pro*, *brain5.pro*.

e.g.



Your program *project2.pro* and procedures contained therein must compile and run on the Computer Science student machines in the lab room (UG 10). Programs with compilation errors receive 0 marks contribution to the final mark for the project.

The files should be submitted using the command **submit 385 2** from one of the student machines in the CS department.

Assessment

The project is out of a maximum total of 20 marks and is worth 20% of your final mark.

Projects will be marked on the basis of completeness, correctness, and readability (including documentation).

All assessment in this subject is to be done on an individual basis. Students who submit for assessment the work of other individuals, or who work together excessively, will be penalised by the Department and risk formal prosecution under the University's Disciplinary Regulations. Punishment is also extended to those who supply projects to other students.

Questions and clarification

You may seek assistance from your lab demonstrator during lab classes or by emailing them (see 385 page for the list of email addresses).

Lloyd C.L. Hollenberg

September, 2004.