Input Output

Garbage In,
Garbage Out

Outline

- Announcements:
  - HWII solutions on web soon
  - Homework III: due Wednesday
- Advanced ASCII
- Binary Basics
- Cell-arrays
- Structs

Advanced ASCII

- Read tables of ASCII data with load
- Other functions like textread will read simple files
- Sometimes, you've just got to do it yourself
  - Complicated files

Opening Files

- To read a file manually, open with fopen
  - fid=fopen('fname', 'rt'); %rt= "read text"
  - fid will be <1 if open fails
  - You should check and return an informative message
- File I/O functions accept fid
- Close the file when you're done with fclose(fid)

Reading files

- A=fscanf(fid,cstring,{N})
  - like C's fscanf, cstring is a C format string:
    - '%d\t%f'--integer (%d),tab(\t),double (%f)
  - fscanf is "vectorized" and Matlab will keep trying to match cstring. Limit with N
- lin=fgetl(fid)
  - Reads a single line from the file as text (char array)
  - Process lin with str2num, findstr, sscanf
- Test for end of file with feof(fid);

Example: Reading .s2r file

- .s2r files are a simple text format for storing data defined on a 2D horizontal mesh
- Specification:
  - line 1: mesh name
  - line 2: comment
  - line 3+: I R
    - I=node number (integer)
    - R=value at node (double)
- I have lots of these files to read (output from FORTRAN model)

```
function s2r=read_s2r(fname);
fid=fopen(fname,'rt');
if(fid<0);error(['Unable to open ',fname]);end
```

read_s2r I

- Read line-by-line using fgetl

```
lin=fgetl(fid);%Read meshname (ignore)
lin=fgetl(fid);%Read comment (ignore)
j=1;
while(~feof(fid));
    lin=fgetl(fid);
    s2r(j,:)=str2num(lin);
    j=j+1;
end
```

read_s2r II

- Pre-allocate s2r: <open and skip as before>

```
buf=100;%size of buffer
s2r=zeros(buf,2);len=buf;
j=1;
while(~feof(fid));
    if(j>len);
        s2r=[s2r; zeros(buf,2)];%add memory to s2r
        len=len+buf;
    end
    lin=fgetl(fid);
    s2r(j,:)= str2num(lin);
    j=j+1;
end
s2r=s2r(1:j-1,:);%remove trailing 0's
```

read_s2r III

- Use fscanf

```
<skip headers as before>
s2r=fscanf(fid,'%d%f'); %s2r is 1-by-(2*NN)
            %s2r=[I(1),R(1),I(2),R(2)…]
NN=length(s2r);
if(mod(NN,2)~=0);error([fname,' contains …]);end
s2r=reshape(s2r, 2, NN/2);
            %s2r=[ I(1) I(2) I(3) . . .
            %       R(1) R(2) R(3) . . .]
s2r=s2r';%Transpose;
```

Writing Files

- Save matrices using save fname varname -ascii
- Doing it yourself:
  - fid=fopen('fname','wt'); %wt= "write text"
  - fprintf(fid,cstring, variables)
  - Example:
    - A=[(1:10)', sin(2*pi*0.1*(1:10)')];%[integers, doubles]
    - fid=fopen('example.txt','wt');
    - fprintf(fid,'%d %f\n',A');%ensures first column is an integer
    - fclose(fid);

Binary Basics

- All computer files are "binary", that is composed of 0's and1's
- When the computer reads ASCII files, it takes chunks of 8 bits (1 byte) and looks up the character
- To save pi to 16 digits takes 18 bytes in ASCII
- If you save the 1's and 0's that correspond to the double precision value of pi, that takes only 8 bytes

Problem with Binary Files

- You can't just look at them
- You must know exactly how they were created
  - integers vs. floating point
  - single precision vs. double precision
  - signed vs. unsigned

Reading Binary files

- fid=fopen(fname,'r');%'r' = read binary
- A=fread(fid,N,precision)
  - N=number of data points, use Inf to read everything
  - precision is how the file was created
    - "uint64" is an unsigned integer saved in 64 bits
    - "double" is a double

(Not so) New Data Types

- Most Matlab variables are arrays (of double or char)
- Matlab has two additional data types
  - cell-arrays--arrays of arbitrary data
  - structs--variable with multiple named fields

Cell-arrays

- In a standard array, each element must be the same type
  - double, char, int, struct
- In a cell array, each element (or cell) can hold anything
  - create with "{ }"
    - cellar{1}=1:10;%array of 10 numbers
    - cellar{2}='an array of 10 numbers';%a string
    - cellar{1}*2
  - can create a blank cell array with "cell"
    - cellar=cell(2,1);%cell array with 2 rows & 1column

Applications of Cell-arrays

- Strings
  - charcell={'Greetings'; 'People of Earth'}
  - disp(charcell) will output each cell on a new line
    - 'Greetings'
    - 'People of Earth'
  - Search lines using strmatch
    - strmatch('Greetings',charcell)
- Encapsulating data
  - FFTcell={a, b, dt}
    - myfft could return a cell array in this form
    - Would keep the related data together
    - But, user would have to know what is in each cell

Structs

- Rather than storing data in elements or cells, structs store data in fields
- Better encapsulation than cell
  - FFTstrct.a=a;
  - FFTstrct.b=b;
  - FFTstrct.dt=dt;
- Can have arrays of structs
  - for j=1:n
    - strctarr(j)=StrctFunc(inputs);
  - end
  - each element must have same fields

Working with Structs

- Lots of commands for working with structs
  - fieldnames(S)--returns a cell array of strings containing names of fields in S
  - isfield(S,'fname')--checks whether fname is a field of S
  - rmfield(S,'fname')--removes fname