naturalValuation.pas

April 8, 2018

Contents

1 naturalvaluation **3**

2 Purpose **3**

3 Comand line interface **3**
3.1 iotable 3

4 rf **5**

5 valuationN **5**

6 csvfilereader **6**

7 Introduction **6**
7.1 Background 6
7.2 CSV File Format 6
7.2.1 CSV File Structure 7

8 CSV File Rules **8**
8.1 File Size 9
8.2 CSV Records 9

9 CSV Record Rules **9**
9.1 CSV Field Column Rules 9
9.2 Header Record Rules 9

10 getdatamatrix **11**

11 recursedown **11**

12 recurse **11**

13 getcolheaders **12**

14 recurse **12**

15 getrowheaders **13**

16 recurse **13**

17 colcount **13**

18 getcell **14**

19 removetrailingnull **14**

20 onlynulls **14**

21 rowcount **15**

22 isint **15**

23 printcsv **15**

24 parsecsvfile **16**

25 thetoken **16**

26 peek **17**

27 isoneof **17**

28 nextsymbol **17**

29 have **17**

30 haveoneof **18**

31 initialise **18**

32 resolvealpha **18**

33 resolvedigits **19**

34 resolvetoken **19**

35 markbegin **19**

36 markend **20**

37 setalpha **20**

38 emptyfield **20**

39 parsebarefield **20**

40 parsedelimitedfield **21**

41 appendcurrentchar **21**

42 parsefield **21**

43 parserecord **22**

44 parseheader **22**

45 parsewholefile **22**

46 vectorlib **24**

47 l2norm **24**

48 l1norm **24**

49 normaliseVec **24**

1 naturalvaluation

program *naturalValuation* ;
uses *csvfilereader* ,*vectorlib* ;

2 Purpose

The programme computes labour values and all alternative value bases from input output tables.

It outputs the valuations using all bases other than labour then the vector of labour values and then the vector of output at market value. Output is in one vector per line space separated.

3 Comand line interface

Usage : `naturalValuation` *`iotable.csv`*

The files should be laid out with a first line and first column made up of text fields labeling the rows and columns. All other cells should be numeric. Discussion of matrix sizes in what follows refers exclusively to the rectangular subarray of numeric values.

3.1 iotable

The first file, the iotable one, should contain an N by M io table in standard column form, with a column corresponding to an industry so that cell at row i col j contains the amount of output from the ith industry used by the jth industry.

The last line must contain the outputs of each industry.

```
type
    pmat =^ matrix ;
    channel = record
        p : pcsv;
        r;
        m : ↑ matrix ;
    end ;
procedure rf ( var ch :channel ;param :integer ); (see Section 4 )
var
    Let iot ∈ channel;
    Let L, V, O, O1, baseN ∈ ^vector;
    Let square ∈ ^matrix;
    Let ip, normip, normo, normn ∈ real;
    Let i, n, j ∈ integer;
function valuationN ( n :integer ):pvec ; (see Section 5 )

begin
    rf (iot, 1);

    new ( L ,iot .m ^ .cols );
    new ( V ,iot .m ^ .cols );
    new ( O ,iot .m ^ .cols );
    new ( O1 ,iot .m ^ .cols );
    L↑← 1;
    V↑← iot.m↑[iot.m↑.rows - 1];
    O↑← iot.m↑[iot.m↑.rows];
    normo← √(O↑ .O↑);
    new ( square ,iot .m ^ .cols ,iot .m ^ .cols );
    square↑← iot.m T↑;
    for n← 1 to square.rows do
    begin
        baseN← valuationN (n);
        normn← √(baseN↑ .baseN↑);
```

```
        ip← (O↑.baseN↑);
        write(ip/(normo×normo):12:3);

        write(ip/(normo×normn):12:3);
        write(baseN↑ : 12 : 3 );

    end ;
    for  i← 1  to  20  do
    begin
        o1↑← V↑ + ∑ (L↑ × square↑ ) ;
        L↑← O1↑ / O↑ ;
    end ;
    basen↑← o1↑;
    normn← √(baseN↑ .basen↑) ;
    ip← (O↑.baseN↑);
    { scaled inner product}
    write(ip/(normo×normo):12:3);
    {normalised inner product}
    write(ip/(normo×normn):12:3);


    writeln(O1↑ : 12 : 3 , 1.0 : 12 : 3, 1.0 : 12 : 3, O↑ : 12 : 3 );

end .
```

4 rf

```
procedure rf ( var ch :channel ;param :integer );
```

Read in one of the file parameters and extract the data from it

```
begin
    with  ch  do
    begin
        p← parsecsvfile (paramstr (param));
        if  p = nil  then
        begin
            writeln( 'error opening or parsing file '  , paramstr (param));
            halt (2);
        end
        else  ;
        r← getrowheaders (p);
        c← getcolheaders (p);
        m← getdatamatrix (p);
```

```
    end ;
end ;
```

5 valuationN

```
function valuationN ( n :integer ):pvec ;
var
    Let i ∈ integer;
```

compute the normalised value vector for the nth value base whilst doing this labour input assumed zero

```
begin
    L↑← 1;
    for i← 1 to 20 do
    begin
        L↑[n]← 1;
        o1↑← ∑ (L↑ × square↑ ) ;
        L↑← O1↑ / O↑ ;
    end ;
    valuationN← (o1);
end ;
```

6 csvfilereader

```
unit csvfilereader ;
```

This parses csv files meeting the official UK standard for such files The following text is imported from that definition at https://www.ofgem.gov.uk/sites/default/files/docs/2013/01/csvfilefor

7 Introduction

7.1 Background

The comma separated values (CSV) format is a widely used text file format often used to exchange data between applications. It contains multiple records (one per line), and each field is delimited by a comma.

7.2 CSV File Format

The primary function of CSV file is to separate each field values by comma separated and transport text - based data to one or more target application. A source application is one which creates or appends to a CSV file and a target application is one which reads a CSV file

7.2.1 CSV File Structure

The CSV file structure use following two notations

FS (Field Separator) i.e. comma separated

FD (Field Delimiter) i.e. Always use a double - quote.

Each line feed in CSV file represents one record and each line is terminated by any valid NL (New line i.e. Carriage Return (CR) ASCII (13) and Line Feed (LF) ASCII (10)) feed. Each record contains one or more fields and the fields are separated by the FS character (i.e. Comma) A field is a string of text characters which will be delimited by the FD character (i.e. double - quote (")) Any field may be quoted (with double quotes).

Fields containing a line - break, double - quote, and/or commas should be quoted. (If they are not, the file will likely be impossible to process correctly).

The FS ch aracter (i.e. comma) may appear in a FD delimited field and in this case it is not treated as the field separator. If a field's value contains one or more commas, double - quotes, CR or LF characters, then it MUST be delimited by a pair of double - quotes (AS CII 0x22).

DO NOT apply double - quote protection where it is not required as applying double quotes on every field or on empty field would takes more file space If a field requires Excel protection, its value MUST be prefixed with a single tilde character .

See example below:

FS =,

FD =”

Data Record:

```
Test1,Test2,,"Test3,Test4","Test5 ""Test6"" Test7","Test8,""",",Test9"
```

Indicat es the following four fields

`Test1`	5 characters
`Test2`	5 characters
	0 characters
`Test3,Test4`	11 characters
`Test5 "Test6" Test7`	20 characters
`Test8,"`	8 characters
`,Test9`	6 characters

8 CSV File Rules

- The file type extension MUST be set to .CSV
- The character set used by data contained in the file MUST be an 8 - bit (UTF - 8).
- No binary data should be transported in CSV file.
- A CSV file MUST contain at least one record.
- No limit to the number of data records
- The End of Record m ust be set to CR +LF (i.e. Carriage Return and Line Feed)
- Do not use whitespaces in the file name
- The EOR marker MUST NOT be taken as being part of the CSV record
- EOF (End of File) character indicates a logical EOF (SUB - ASCII 0x1A) and not the physical en d .
- A logical EOF marker cannot be double - quote protected.
- Any record appears after the EOF will be ignored

8.1 File Size

Maximum csv file size should be 30 MB.

8.2 CSV Records

A CSV record consists of two elements, a data record followed by an end - of - record marker (EOR). The EOR is a data record delivery marker and does not form part of the data delivered by the record

9 CSV Record Rules

Pls. note this rule applies to every CSV record including the last record in the file.

9.1 CSV Field Column Rules

- Each recor d within the same CSV file MUST contain the same number of field columns . The header record describes how many fields the application should expect to process.
- Field columns MUST be separated from each other by a single separation character
- A field column MUST NOT have leading or trailing whitespace

9.2 Header Record Rules

A header record allows the Ofgem IT systems to guard against the potential issues such as missing column or additional column that are not in scope

- The header record MUST be the first recor d in the file.
- A CSV file MUST contain one header record only .
- Header labels MUST NOT be blank.
- Use single word only
- Do not use spaces (Use _ if words needs to be separated)

```
interface
  const
    textlen =80;
  type
    pcsv =^ csvcell ;
    celltype =( linestart ,numeric ,alpha );
    textfield =textline ;
    csvcell = record
      right : pcsv;
      case tag : celltype of
        linestart : (down : pcsv);
        numeric : (number : real);
        α : (textual : pstring);
      end ;

      headervec ( max :integer ) =array [1..max ]of pcsv ;
      pheadervec = ↑ headervec ;
    procedure printcsv ( var f :text ;p :pcsv ); (see Section ?? )
    function parsecsvfile ( name :textline ):pcsv ; (see Section ?? )
    function rowcount ( p :pcsv ):integer ; (see Section ?? )
    function getdatamatrix ( p :pcsv ):^ matrix ; (see Section 10 )
    function getcell ( p :pcsv ;row ,col :integer ):pcsv ; (see Section ?? )
    function getrowheaders ( p :pcsv ):^ headervec ; (see Section ?? )
    function getcolheaders ( p :pcsv ):^ headervec ; (see Section ?? )
    function colcount ( p :pcsv ):integer ; (see Section ?? )
```

returns nil for file that can not be opened, otherwise returns pointer to tree of csvcells.

```
    implementation

field delimitor
    const
      FD = 34;
field separator
      FS = 44;
record separator
      RS = 10;
```

```
        EOI = $1a;
        CR = 13;
    type
        token = (FDsym);
        tokenset = set of token ;
    var
        categorisor:  array  [byte]  of  token;

    function getdatamatrix ( p :pcsv ):^ matrix ; (see Section 10 )
```

10 getdatamatrix

```
function getdatamatrix ( p :pcsv ):^ matrix ;
```

extract the column headers as a vector of strings

```
var
    m : ↑ matrix ;
procedure recursedown ( j :integer ;q :pcsv ); (see Section 11 )
```

11 recursedown

```
procedure recursedown ( j :integer ;q :pcsv );
procedure recurse ( i :integer ;q :pcsv ); (see Section 12 )
```

12 recurse

```
procedure recurse ( i :integer ;q :pcsv );
begin

    if  q ≠ nil  then
    begin
        if  i ≥ 1  then
        begin
            if  q↑.tag = numeric  then
                m↑[j, i]← q↑.number
            else  m↑[j, i]← 0.0
        end ;
        recurse (i + 1, q↑.right);
    end
    ;
end ;

begin
```

```
    if  q ≠ nil  then
    begin
        recurse (0, q↑.right);
        recursedown (j + 1, q↑.down);
    end
end ;
begin

    if  p = nil  then  getdatamatrix← nil
    else
    begin
        new ( m ,rowcount ( p )-1,colcount ( p )-1);
        recursedown (1, p↑.down);
        getdatamatrix← m;
    end ;
end ;
function getcolheaders ( p :pcsv ):^ headervec ; (see Section 13 )
```

13 getcolheaders

```
function getcolheaders ( p :pcsv ):^ headervec ;
```

extract the column headers

```
var
    M;
    h : ↑ headervec ;
procedure recurse ( i :integer ;q :pcsv ); (see Section 14 )
```

14 recurse

```
procedure recurse ( i :integer ;q :pcsv );
begin
    if  q ≠ nil  then
    begin

        if  i ≥ 1  then  h↑[i]← q;
        recurse (i + 1, q↑.right);
    end
end ;
begin

    if  p = nil  then  getcolheaders← nil
    else
    begin
```

```
        new ( h ,colcount ( p )-1);
        recurse (0, p↑.right);
        getcolheaders← h;
    end ;
end ;
function getrowheaders ( p :pcsv ):^ headervec ; (see Section 15 )
```

15 getrowheaders

```
function getrowheaders ( p :pcsv ):^ headervec ;
```

extract the rows headers

```
var
    M;
    h : ↑ headervec ;
procedure recurse ( i :integer ;q :pcsv ); (see Section 16 )
```

16 recurse

```
procedure recurse ( i :integer ;q :pcsv );
begin
    if q ≠ nil then
    begin
        h↑[i]← q↑.right;
        recurse (i + 1, q↑.down);
    end
end ;
begin
    if p = nil then getrowheaders← nil
    else
    begin
        new ( h ,rowcount ( p )-1);
        recurse (1, p↑.down);
        getrowheaders← h;
    end ;
end ;
function colcount ( p :pcsv ):integer ; (see Section 17 )
```

17 colcount

```
function colcount ( p :pcsv ):integer ;
```

return the number of columns in the spreadsheet

```
begin
    if  p = nil  then  colcount← 0
    else
        case  p↑.tag  of
            linestart :  colcount← colcount (p↑.right);
        end
end ;
function getcell ( p :pcsv ;row ,col :integer ):pcsv ; (see Section 18 )
```

18 getcell

```
function getcell ( p :pcsv ;row ,col :integer ):pcsv ;
```

return the cell at position row,col in the spredsheet

```
begin
    if  p = nil  then  getcell← nil
    else  if  row = 1  then
    begin
        else  if  col = 1  then  getcell← p
    end
end ;

procedure removetrailingnull ( var p :pcsv ); (see Section 19 )
```

19 removetrailingnull

```
procedure removetrailingnull ( var p :pcsv );
function onlynulls ( q :pcsv ):boolean ; (see Section 20 )
```

20 onlynulls

```
function onlynulls ( q :pcsv ):boolean ;
begin
    if  q = nil  then  onlynulls← false  false
    else
        if  q↑.tag = α  then
        begin
        end
        else  onlynulls← false  false
end ;
```

```
begin
    if  p ≠ nil  then
        case  p↑.tag  of
            linestart :
            or ( ( p ^ .down =nil )and onlynulls ( p ^ .right )) then p :=nil
            else  removetrailingnull (p↑.down);
        end
end ;
function rowcount ( p :pcsv ):integer ; (see Section 21 )
```

21 rowcount

```
function rowcount ( p :pcsv ):integer ;
begin
    if  p = nil  then  rowcount← 0
    else
        case  p↑.tag  of
            linestart : rowcount← 1 + rowcount (p↑.down);
            numeric← 1
        end
end ;
function isint ( r :real ):boolean ; (see Section 22 )
```

22 isint

```
function isint ( r :real ):boolean ;
var
    i : integer;
begin
    i← round(r);
    isint← (i × 1.0) = r
end ;
procedure printcsv ( var f :text ;p :pcsv ); (see Section 23 )
```

23 printcsv

```
procedure printcsv ( var f :text ;p :pcsv );
begin
    if  p ≠ nil  then
        with  p↑  do
        begin
            if  tag = linestart  then
            begin
                printcsv (f, right);
                if  down ≠ nil  then
```

```
            begin
                writeln(f);
                printcsv (f, down);
            end ;
        end
        else
            if tag = numeric then
            begin
                else write(f, number : 1 : 6);
                if right ≠ nil then
                begin
                    write ( f , ',' );
                end

            end
            else
                if tag = α then
                begin

                    if textual ≠ nil then write(f, '"' , textual↑, '"' ) else write(f, 'nil' );
                    if right ≠ nil then
                    begin

                        write ( f , ',' );
                    end

                end
    end
end ;
function parsecsvfile ( name :textfield ):pcsv ; (see Section 24 )
```

24 parsecsvfile

```
function parsecsvfile ( name :textfield ):pcsv ;
const
    megabyte = 1024 × 1024;
    maxbuf = 30 × megabyte;
type
    bytebuf = array [1..maxbuf ] of byte ;
var
    f : fileptr;
    bp : ↑ bytebuf ;
    fs;
    tokstart;
    firstfield;
function thetoken :token ; (see Section 25 )
```

25 thetoken

```
function thetoken :token ;
begin
    if currentchar ≤ fs then
        thetoken← categorisor bp↑[currentchar]
    else thetoken← EOFsym
end ;
function peek ( c :token ):boolean ; (see Section 26 )
```

26 peek

```
function peek ( c :token ):boolean ;
```

matches current char against the token c returns true if it matches.

```
begin
    peek← c = thetoken
end ;
function isoneof ( s :tokenset ):boolean ; (see Section 27 )
```

27 isoneof

```
function isoneof ( s :tokenset ):boolean ;
begin
    isoneof← thetoken ∈ s
end ;
procedure nextsymbol ; (see Section 28 )
```

28 nextsymbol

```
procedure nextsymbol ;
begin
    if currentchar ≤ fs then currentchar← currentchar + 1
end ;
function have ( c :token ):boolean ; (see Section 29 )
```

29 have

```
function have ( c :token ):boolean ;
begin
    if peek (c) then
```

```
    begin
        nextsymbol;
        have← true;

    end
    else
        have← false;
end ;
function haveoneof ( c :tokenset ):boolean ; (see Section 30 )
```

30 haveoneof

```
function haveoneof ( c :tokenset ):boolean ;
begin
    if isoneof (c) then
    begin
        nextsymbol;
        haveoneof← true;
    end
    else
        haveoneof← false;
end ;

procedure initialise ; (see Section 31 )
```

31 initialise

```
procedure initialise ;
begin
    firstfield← nil;
    lastfield← nil;
    firstrecord← nil;

end ;
procedure resolvealpha ; (see Section 32 )
```

32 resolvealpha

```
procedure resolvealpha ;
var
    i;
begin
    with lastfield↑ do
    begin
        tag← α;
```

```
        new ( textual );
        textual↑← '' ;
        l← tokend min(tokstart + textlen - 1) ;
        { copy field to string}
        for i← tokstart to l - 1 do
        begin
            textual↑← textual↑ + chr(bp↑[i]) ;
        end ;
    end ;
end ;

procedure resolvedigits ; (see Section 33 )
```

33 resolvedigits

```
procedure resolvedigits ;
var
    i;
    s : string;
begin
    with lastfield↑ do
    begin
        tag← numeric;
        new ( textual );
        s← '' ;
        l← tokend min(tokstart + textlen - 1) ;
        { copy field to a string }
        for i← tokstart to l do
        begin
            s← s + chr(bp↑[i]);
        end ;
convert to binary
        val (s, number, l);
    end ;
end ;
procedure resolvetoken ; (see Section 34 )
```

34 resolvetoken

```
procedure resolvetoken ;
begin
    if chr(bp↑[tokstart]) in [ '0' .. '9' ] then resolvedigits
    else resolvealpha
end ;

procedure markbegin ; (see Section 35 )
```

35 markbegin

mark start of a field

```
procedure markbegin ;
begin
    tokstart← currentchar;
    new ( lastfield ^ .right );
    lastfield← lastfield↑.right;
    lastfield↑.right← nil;
end ;
procedure markend ; (see Section 36 )
```

36 markend

marks the end of a field

```
procedure markend ;
begin
    tokend← currentchar;
    resolvetoken;
end ;
procedure setalpha ( s :textfield ); (see Section 37 )
```

37 setalpha

```
procedure setalpha ( s :textfield );
begin
    lastfield↑.tag← α;
    new ( lastfield ^ .textual );
    lastfield↑.textual↑← s;
end ;
procedure emptyfield ; (see Section 38 )
```

38 emptyfield

```
procedure emptyfield ;
begin

    markbegin;
    setalpha ( '' );

end ;

procedure parsebarefield ; (see Section 39 )
```

39 parsebarefield

```
procedure parsebarefield ;
begin
    if isoneof ([RSsym, EOFsym, FSsym]) then emptyfield
else begin begin
        markbegin;
skip over the field
        while haveoneof ([any, space]) do   ;
        markend;
    end ;
end ;
procedure parsedelimitedfield ; (see Section 40 )
```

40 parsedelimitedfield

```
procedure parsedelimitedfield ;
```

parses a field nested between " chars converting escape chars as it goes

```
var
    s : textfield;
    i : integer;
    continue : boolean;
procedure appendcurrentchar ; (see Section 41 )
```

41 appendcurrentchar

```
procedure appendcurrentchar ;
begin
    s← s + chr(bp↑[currentchar]);
    nextsymbol;
end ;
begin
    markbegin;
    s←  '' ;
    continue← true;
    repeat
        while isoneof ([FSsym..any]) do
        begin
            appendcurrentchar;
        end ;
eat what may be closing quotes
        have (FDsym);
        continue← peek (FDsym) ∧ (length (s) < textlen);
        if continue then appendcurrentchar;
    until (not continue );
```

```
    setalpha (s);
end ;
procedure parsefield ; (see Section 42 )
```

42 parsefield

```
procedure parsefield ;
begin
    if have (FDsym) then parsedelimitedfield
    else parsebarefield
end ;
procedure parserecord ; (see Section 43 )
```

43 parserecord

```
procedure parserecord ;
begin
    parsefield;
    while have (FSsym) do parsefield;
end ;
procedure parseheader ; (see Section 44 )
```

44 parseheader

```
procedure parseheader ;
begin
    { claim heap space for start of first line }
    new ( firstrecord );
    lastfield← firstrecord;
    firstfield← firstrecord;
    with firstrecord↑ do
    begin
        tag← linestart;
        down← nil;
        right← nil;
    end ;
    parserecord;
end ;
procedure parsewholefile ; (see Section 45 )
```

45 parsewholefile

```
procedure parsewholefile ;
```

```
begin
    parseheader;

    while have (RSsym) do
    begin
        { claim heap space for the start of the new line }
        new ( firstfield ^ .down );
        firstfield← firstfield↑.down;
        lastfield← firstfield;
        with firstfield↑ do
        begin
            tag← linestart;
            down← nil;
            right← nil;
        end ;
        parserecord;
    end ;
end ;
begin
    initialise;
the default case of failure
    parsecsvfile← nil;

    assign (f, name);
open file for reading
    reset (f);
ioresult =0 if opened ok
    if ioresult = 0 then
    begin
        fs← filesize (f);
        if fs < maxbuf then
        begin
            new ( bp );
            blockread (f, bp↑[1], fs, rc);
            if rc = fs then
            begin
                currentchar← 1;
```

We now have the csv file in memory - parse it

```
                parsewholefile;
                removetrailingnull (firstrecord);
                parsecsvfile← firstrecord;
            end ;
            dispose ( bp );
            close (f);
        end ;
    end ;
end ;
begin
    categorisor← any;
```

```
    categorisor FD← FDsym;
    categorisor FS← FSsym;
    categorisor RS← RSsym;
    categorisor EOI← EOFsym;
    categorisor ord( ' ' )← space;
    categorisor CR← space;
    {writeln('fs=',fs,'fd=',fd,'rs=',rs);
    writeln(categorisor);}
end .
```

46 vectorlib

```
unit vectorlib ;
interface
    type
        pvec =^ vector ;
    function l2norm ( var v :vector ):real ; (see Section 47 )
    function l1norm ( var v :vector ):real ; (see Section 48 )
    function normaliseVec ( var v :vector ):pvec ; (see Section 49 )
    implementation

    function l2norm ( var v :vector ):real ; (see Section 47 )
    function l1norm ( var v :vector ):real ; (see Section 48 )
    function normaliseVec ( var v :vector ):pvec ; (see Section 49 )
    begin
    end .
```

47 l2norm

```
function l2norm ( var v :vector ):real ;
begin
    l2norm← √(v.v);
end ;
```

48 l1norm

```
function l1norm ( var v :vector ):real ;
begin
    l1norm← ∑ absr (v) ;
end ;
```

49 normaliseVec

function *normaliseVec* (**var** *v* :*vector*):*pvec* ;
var
    Let $p \in$ pvec;
    Let *norm* $\in$ real;
begin
    new (*p* ,*v* .*cols*);
    norm$\leftarrow$ *l2norm* (*v*);
    $p\uparrow \leftarrow \frac{v}{norm}$;
    normaliseVec$\leftarrow$ *p*;
end ;