```
function [num,spl,sgf,opts] = words2num(txt,opts,varargin)
% Convert text of numbers written in English to their numeric values (GB/IN/US).
%
% (c) 2015-2023 Stephen Cobeldick
%
% WORDS2NUM detects in the input string any numbers written in English and
% converts them to numeric values, e.g. 'one hundred and one' -> 101.
%
%%% Syntax:
%  num = words2num(txt)
%  num = words2num(txt,opts)
%  num = words2num(txt,<name-value pairs>)
% [num,spl,sgf,opts] = words2num(...)
%
% The number format is based on: <http://www.blackwasp.co.uk/NumberToWords.aspx>
% and <https://www.bbc.co.uk/learningenglish/course/intermediate/unit-25/session-1>
%
% The number recognition can be controlled by options such as the character
% case, requiring or excluding 'and', '-' or ',' from the numbers, a leading
% sign, the class of the output numeric, etc.: see the section "Options" below.
% If no number is identified in the string then the output <num> will be empty.
%
% Note1: Common spelling variants are accepted, e.g.: Do|Duo, Tre|Tres,
%        Quin|Quinqua, Ses|Sex, Sept|Septem|Septen, Octocto|Octoocto.
% Note2: Fractional digits may be given following the word 'point'.
% Note3: Infinity and Not-a-Number must be spelled out in full.
% Note4: Identifies and converts number strings up to realmax('double').
%
% Copyright (c) 2023, Stephen Cobeldick
% All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided that the following conditions are met:
%
% * Redistributions of source code must retain the above copyright notice, this
%   list of conditions and the following disclaimer.
%
% * Redistributions in binary form must reproduce the above copyright notice,
%   this list of conditions and the following disclaimer in the documentation
%   and/or other materials provided with the distribution
%
% * Neither the name of  nor the names of its
%   contributors may be used to endorse or promote products derived from this
%   software without specific prior written permission.
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
% DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
% FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
% DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
```

```
% SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
% CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
% OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
% OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%
%% Options %%
%
% The options may be supplied either
% 1) in a scalar structure, or
% 2) as a comma-separated list of name-value pairs.
%
% Field names and string values are case-insensitive. The following
% field names and values are permitted as options (**=default value):
%
% Field  | Permitted  |
% Name:  | Values:    | Description (example string or regular expression):
% =======|============|====================================================
% class  |'double'  **| The class of the output <num>. Note that information
%        |'single'    | may be lost during conversion from the text number/s.
%        |'int8'|'int16'|'int32'|'int64'|'uint8'|'uint16'|'uint32'|'uint64'
% -------|------------|----------------------------------------------------
% scale  |'short'   **| Short scale, modern English   (1e9=='one billion')
%        |'long'      | Long scale, B.E. until 1970's (1e9=='one thousand million')
%        |'peletier'  | Most other European languages (1e9=='one milliard')
%        |'rowlett'   | Russ Rowlett's Greek-based    (1e9=='one gillion')
%        |'indian'    | Indian with Lakh, Crore, etc. (1e9=='one arab')
%        |'yllion'    | Donald Knuth's logarithmic    (1e9=='ten myllion')
% -------|------------|----------------------------------------------------
% case   |'ignore'  **| Any case allowed  ('oNe tHouSAnd AnD TWENtY-FoUR')
%        |'lower'     | Lowercase number  ('one thousand and twenty-four')
%        |'upper'     | Uppercase number  ('ONE THOUSAND AND TWENTY-FOUR')
%        |'title'     | Titlecase number  ('One Thousand and Twenty-Four')
% -------|------------|----------------------------------------------------
% mag    |'simple'  **| Magnitude must be simple  ('one trillion')
%        |'compound'  | Magnitude may be compound ('one million million')
% -------|------------|----------------------------------------------------
% and    | []       **| Optional 'and' before tens/ones ('one hundred (and )?one)
%        | true       | Require 'and' before tens/ones  ('one hundred and one')
%        | false      | Exclude 'and' before tens/ones  ('one hundred one')
% -------|------------|----------------------------------------------------
% comma  | []       **| Optional comma separator ('one million,? one thousand')
%        | true       | Require comma separator  ('one million, one thousand')
%        | false      | Exclude comma separator  ('one million one thousand')
% -------|------------|----------------------------------------------------
% hyphen | []       **| Optional hyphen between tens and ones ('twenty(-| )four')
%        | true       | Require hyphen between tens and ones  ('twenty-four')
%        | false      | Exclude hyphen between tens and ones  ('twenty four')
% -------|------------|----------------------------------------------------
% space  | []       **| Optional space character ('one *hundred')
%        | true       | Require space character  ('one hundred')
%        | false      | Exclude space character  ('onehundred')
```

```
% -------|------------|-----------------------------------------------------
% sign   | []       **| Optional sign prefix ('(positive )?twenty-four')
%        | true       | Require sign prefix  ('positive twenty-four')
%        | false      | Exclude sign prefix  ('twenty-four')
% -------|------------|-----------------------------------------------------
% white  | '\s'     **| A vector of whitespace characters, may include
%        |            | regexp meta-characters (e.g. ' ' or '-\t' or '_\s' etc.)
% -------|------------|-----------------------------------------------------
% prefix | ''       **| A regular expression that precedes each number substring.
% -------|------------|-----------------------------------------------------
% suffix | ''       **| A regular expression that follows each number substring.
% -------|------------|-----------------------------------------------------
%
% Note5: Text names and values may be character vector or string scalar.
% Note6: The <scale> 'yllion' ignores the options <comma>, <and>, and <mag>.
%
%% Examples %%
%
% >> words2num('zero') % or "zero"
% ans = 0
%
% >> words2num('One Thousand and TWENTY-four')
% ans = 1024
% >> words2num('One_Thousand_and_TWENTY-four', 'white','_')
% ans = 1024
% >> words2num('One Thousand and TWENTY-four', 'case','lower')
% ans = 4
% >> words2num('One Thousand and TWENTY-four', 'case','upper')
% ans = 20
% >> words2num('One Thousand and TWENTY-four', 'case','title')
% ans = 1000
% >> words2num('One Thousand and TWENTY-four', 'hyphen',false)
% ans = [1020,4]
% >> words2num('One Thousand and TWENTY-four', 'and',false)
% ans = [1000,24]
% >> words2num('One Thousand and TWENTY-four', 'suffix','-')
% ans = 1020
%
% >> [num,spl] = words2num('Negative thirty-two squared is one thousand and twenty-↙
four.')
% num = [-32,1024]
% spl = {'',' squared is ','.'}
%
% >> [num,spl] = words2num('one hundred and twenty-three pounds and forty-five pence')
% num = [123,45]
% spl = {'',' pounds and ',' pence'}
%
% >> [num,spl] = words2num('pi=threepointonefouronefiveninetwosixfivethreefiveeight')
% num = 3.14159265358
% spl = {'pi=',''}
%
```

```
% >> [num,spl] = words2num('One Hundred and One Dalmatians')
% num = 101
% spl = {'',' Dalmatians'}
%
% >> words2num('one hundred and seventy-nine uncentillion')
% ans = 1.79e+308
% >> words2num('one hundred and eighty uncentillion') % >realmax
% ans = Inf
% >> words2num('one hundred and eighty uncentillion', 'class','int64')
% ans = +92233720368547758O7
% >> words2num(num2words(intmin('int64')),'class','int64')
% ans = -9223372036854775808
%
% >> words2num('one point zero zero two zero zero three trillion')
% ans = 1002003000000
% >> words2num('one trillion, two billion, three million')
% ans = 1002003000000
% >> words2num('one trillion, two billion three million', 'comma',true, 'and',true)
% ans = [1002000000000,3000000]
% >> words2num('one trillion, two billion three million', 'comma',false)
% ans = [1000000000000,2003000000]
% >> words2num('one million million', 'mag','compound')
% ans =  1000000000000
%
% >> words2num('four billion', 'scale','short')
% ans = 4000000000
% >> words2num('four thousand million', 'scale','long')
% ans = 4000000000
% >> words2num('four arab', 'scale','indian')
% ans = 4000000000
% >> words2num('four milliard', 'scale','peletier')
% ans = 4000000000
% >> words2num('four gillion', 'scale','rowlett')
% ans = 4000000000
% >> words2num('FORTY MYLLION', 'scale','yllion')
% ans = 4000000000
%
% >> words2num('Negative Infinity')
% ans = -Inf
% >> words2num('Not-a-Number')
% ans = NaN
%
%% Input and Output Arguments %%
%
%%% Inputs:
% txt  = CharacterVector or StringScalar, containing zero or more numbers
%        written in english words, the numbers to be converted to numeric.
% opts = StructureScalar, optional fields and values as per 'Options' above.
% OR
% <name-value pairs> = a comma-separated list of names and corresponding values.
%
```

```
%%% Outputs:
% num  = NumericVector, the numeric values of detected number text in <str>.
% spl  = CellOfChars, the parts of <str> split by the detected number substrings.
% sgf  = NumericVector, the significant figures of the text. Same size as <num>.
% opts = StructureScalar, the used parameter values as per 'Options' above.
%
% See also NUM2WORDS WORDS2NUM_TEST WORDS2NUM_DEMO SIP2NUM BIP2NUM
% SSCANF STR2DOUBLE DOUBLE TEXTSCAN
%% Input Wrangling %%
%
if isa(txt,'string')
    assert(isscalar(txt),...
        'SC:words2num:txt:NotStringScalar',...
        'First input <str> must be a string scalar or a character row vector.')
    txt = txt{1};
else
    szv = size(txt);
    assert(ischar(txt)&&numel(szv)==2&&szv(1)<2,...
        'SC:words2num:txt:NotCharRowVec',...
        'First input <str> must be a string scalar or a character row vector.')
end
%
% Default option values:
stpo = struct(...
    'and',[], 'comma',[], 'hyphen',[], 'sign',[], 'space',[],...
    'white','\s', 'case','ignore', 'class','double', 'scale','short',...
    'prefix','', 'suffix','', 'mag','simple');
%
% Check any supplied option fields and values:
switch nargin
    case 1 % no user-supplied options
    case 2 % options in a struct
        assert(isstruct(opts)&&isscalar(opts),...
            'SC:words2num:opts:NotScalarStruct',...
            'Second input <opts> structure must be scalar.')
        opts = structfun(@w2n1s2c,opts,'UniformOutput',false);
        stpo = w2nOptions(stpo,opts);
    otherwise % options as <name-value> pairs
        varargin = cellfun(@w2n1s2c,varargin,'UniformOutput',false);
        opts = struct(w2n1s2c(opts),varargin{:});
        assert(isscalar(opts),...
            'SC:words2num:options:ValueCellArray',...
            'Invalid <name-value> pairs: cell array values are not permitted.')
        stpo = w2nOptions(stpo,opts);
end
opts = stpo;
%
num = zeros(0,0,stpo.class);
sgf = zeros(0,0);
%
%% Grammar Parsing %%
```

```
%
sIni = '';
sMag = '';
sCur = '';
xNxt = NaN;
%
% Placeholders {case-insensitive placeholder, replacement regular expression}:
pWht = {'<-0->',stpo.white};
pHyp = {'<-1->','[\x2D\x2010\x2011\xFF0D]'}; % hyphens galore!
pDyn = {'<-2->','(??@fDyn())*'};
pBef = {'<-3->','(?@fBef($&))'};
pIni = {'<-4->','(?@fIni($&))'};
pAft = {'<-5->','(?@fAft($&))'};
pNoP = {'<-6->','isempty(regexpi($&,''POINT''))'};
%
% Space character/s:
if isequal([],stpo.space)
    rSpc = sprintf('%s*',pWht{1});
elseif stpo.space
    rSpc = sprintf('%s+',pWht{1});
else
    rSpc = '';
end
% "And" before tens or ones:
if isequal([],stpo.and)
    rAnd = sprintf('%s(\and%s)?',rSpc,rSpc);
elseif stpo.and
    rAnd = sprintf('%s\and%s',rSpc,rSpc);
else
    rAnd = rSpc;
end
% Comma after thousand/million/etc.:
if isequal([],stpo.comma)
    rCom = ',?';
elseif stpo.comma
    rCom = ',';
else
    rCom = '';
end
% Hyphen between tens and ones:
if isequal([],stpo.hyphen)
    rHyp = sprintf('(%s|%s?)',rSpc,pHyp{1});
elseif stpo.hyphen
    rHyp = pHyp{1};
else
    rHyp = rSpc;
end
% Positive/negative sign:
if isequal([],stpo.sign)
    rSgn = sprintf('((Posi|Nega)tive%s)?',rSpc);
elseif stpo.sign
```

```
    rSgn = sprintf('(Posi|Nega)tive%s',rSpc);
else
    rSgn = '';
end
% Prefix regular expression:
if numel(stpo.prefix)
    rPfx = sprintf('(?:(%s))',stpo.prefix);
else
    rPfx = '';
end
% Suffix regular expression:
if numel(stpo.suffix)
    rSfx = sprintf('(?:(%s))',stpo.suffix);
else
    rSfx = '';
end
%
% Ones, Tweens, Teens, and Tens:
cOne = {'Zero','One','Two','Three','Four','Five','Six','Seven','Eight','Nine'};
nOne = [     0,    1,    2,      3,     4,     5,    6,      7,      8,     9];
cTwe = { 'Ten','Eleven','Twelve'};
nTwe = [    10,      11,      12];
cTee = {'Thir','Four','Fif','Six','Seven','Eigh','Nine'};%teen
nTee = [    13,    14,   15,   16,     17,    18,    19];
cTyy = {'Twen','Thir','For','Fif','Six','Seven','Eigh','Nine'};%ty
nTyy = [    20,    30,   40,   50,   60,     70,    80,    90];
%
rOne = w2nJoin(cOne);
rTwe = w2nJoin(cTwe);
rTee = w2nJoin(cTee,'teen');
rTyy = w2nJoin(cTyy,'ty');
%
r099 = sprintf('(%s(%s(%s))?|%s|%s|%s)',rTyy,rHyp,rOne,rTee,rTwe,rOne); % [0-99]
rFra = sprintf('%sPoint(%s(%s))+',rSpc,rSpc,rOne); % .[0-9]+
%
nAll = [nOne,nTwe,nTee,nTyy];
cAll = [cOne,cTwe,strcat(cTee,'teen'),strcat(cTyy,'ty')];
fAll = @(d)sprintf(' %d',nAll(strcmpi(d,cAll)));
dAll = w2nJoin({rTyy,rTee,rTwe,rOne});
dOne = sprintf('(?<=(\\s|%s|%s))(%s)',rTyy,rOne,dAll);
%
% E-notation has better precision than using 10^N:
vMag = [1e6;1e9;1e12;1e15;1e18;1e21;1e24;1e27;1e30;1e33;1e36;1e39;1e42;1e45;1e48;1e51;↙
1e54;1e57;1e60;1e63;1e66;1e69;1e72;1e75;1e78;1e81;1e84;1e87;1e90;1e93;1e96;1e99;1e102;↙
1e105;1e108;1e111;1e114;1e117;1e120;1e123;1e126;1e129;1e132;1e135;1e138;1e141;1e144;↙
1e147;1e150;1e153;1e156;1e159;1e162;1e165;1e168;1e171;1e174;1e177;1e180;1e183;1e186;↙
1e189;1e192;1e195;1e198;1e201;1e204;1e207;1e210;1e213;1e216;1e219;1e222;1e225;1e228;↙
1e231;1e234;1e237;1e240;1e243;1e246;1e249;1e252;1e255;1e258;1e261;1e264;1e267;1e270;↙
1e273;1e276;1e279;1e282;1e285;1e288;1e291;1e294;1e297;1e300;1e303;1e306;1e309];
%
% Magnitudes:
```

```
switch stpo.scale
    case 'yllion'
        [num,spl,sgf] = w2nYllion(txt,stpo,pWht,pHyp,rHyp,rSgn,rSpc,r099,rFra,rPfx,rSfx,↙
dOne,fAll);
        return
    case 'indian'
        vMag = [   1e5,    1e7,   1e9,    1e11,      1e13,   1e15,    1e17,        1e19];
        cMag = {'Lakh','Crore','Arab','Kharab','N(i|ee)l','Padma','Shankh','Mahashankh'};
    case 'rowlett'
        cMag =↙
{'M','G','Tetr','Pent','Hex','Hept','Okt','Enn','Dek','Hendek','Dodek','Trisdek','Tetrade↙
k','Pentadek','Hexadek','Heptadek','Oktadek','Enneadek','Icos','Icosihen','Icosid','Icosi↙
tr','Icositetr','Icosipent','Icosihex','Icosihept','Icosiokt','Icosienn','Triacont','Tria↙
contahen','Triacontad','Triacontatr','Triacontatetr','Triacontapent','Triacontahex','Tria↙
contahept','Triacontaokt','Triacontaenn','Tetracont','Tetracontahen','Tetracontad','Tetra↙
contatr','Tetracontatetr','Tetracontapent','Tetracontahex','Tetracontahept','Tetracontaok↙
t','Tetracontaenn','Pentacont','Pentacontahen','Pentacontad','Pentacontatr','Pentacontate↙
tr','Pentacontapent','Pentacontahex','Pentacontahept','Pentacontaokt','Pentacontaenn','He↙
xacont','Hexacontahen','Hexacontad','Hexacontatr','Hexacontatetr','Hexacontapent','Hexaco↙
ntahex','Hexacontahept','Hexacontaokt','Hexacontaenn','Heptacont','Heptacontahen','Heptac↙
ontad','Heptacontatr','Heptacontatetr','Heptacontapent','Heptacontahex','Heptacontahept',↙
'Heptacontaokt','Heptacontaenn','Oktacont','Oktacontahen','Oktacontad','Oktacontatr','Okt↙
acontatetr','Oktacontapent','Oktacontahex','Oktacontahept','Oktacontaokt','Oktacontaenn',↙
'Enneacont','Enneacontahen','Enneacontad','Enneacontatr','Enneacontatetr','Enneacontapent↙
','Enneacontahex','Enneacontahept','Enneacontaokt','Enneacontaenn','Hect','Hectahen','Hec↙
tad'};
    otherwise % short & long & peletier
        cMag = {'M','B','Tr','Quadr','Quint','Sext','Sept','Oct','No(n|ve[nm]↙
t)','Dec','Undec','Du?odec','Tredec','Quattuordec','Quin(qua)?dec','Se[sx]?dec','Sept(e↙
[mn])?dec','Octodec','Nove[mn]dec','Vigint','Unvigint','Du?ovigint','Tres?↙
vigint','Quattuorvigint','Quin(qua)?vigint','Se[sx]vigint','Sept(e[mn])?↙
vigint','Octovigint','Nove[mn]vigint','Trigint','Untrigint','Du?otrigint','Tres?↙
trigint','Quattuortrigint','Quin(qua)?trigint','Se[sx]trigint','Sept(e[mn])?↙
trigint','Octotrigint','Nove[mn]trigint','Quadragint','Unquadragint','Du?↙
oquadragint','Tres?quadragint','Quattuorquadragint','Quin(qua)?quadragint','Se[sx]↙
quadragint','Sept(e[mn])?quadragint','Octoquadragint','Nove[mn]↙
quadragint','Quinquagint','Unquinquagint','Du?oquinquagint','Tres?↙
quinquagint','Quattuorquinquagint','Quin(qua)?quinquagint','Se[sx]quinquagint','Sept(e↙
[mn])?quinquagint','Octoquinquagint','Nove[mn]quinquagint','Sexagint','Unsexagint','Du?↙
osexagint','Tres?sexagint','Quattuorsexagint','Quin(qua)?sexagint','Se[sx]?↙
sexagint','Sept(e[mn])?sexagint','Octosexagint','Nove[mn]↙
sexagint','Septuagint','Unseptuagint','Du?oseptuagint','Tres?↙
septuagint','Quattuorseptuagint','Quin(qua)?septuagint','Se[sx]?septuagint','Sept(e[mn])?↙
septuagint','Octoseptuagint','Nove[mn]septuagint','Octogint','Unoctogint','Du?o?↙
octogint','Tres?octogint','Quattuoroctogint','Quin(qua)?octogint','Se[sx]octogint','Sept↙
(e[mn])?octogint','Octo?octogint','Nove[mn]octogint','Nonagint','Unnonagint','Du?↙
ononagint','Tres?nonagint','Quattuornonagint','Qui(n(qua)?)?nonagint','Se[sx]?↙
nonagint','Sept(e[mn]?)?nonagint','Octononagint','Nove[mn]?nonagint','Cent','Uncent'};
end
%
switch stpo.scale
```

```
    case {'short','rowlett'}
        sSfx = 'illion';
        dMag = strcat('^',cMag,sSfx);
    case 'long'
        vMag = vMag(1:2:end);
        sSfx = 'illion';
        cMag = cMag(1:ceil(numel(cMag)/2));
        dMag = strcat('^',cMag,sSfx);
    case 'peletier'
        sSfx = 'illi(on|ard)';
        cMag = cMag(1:ceil(numel(cMag)/2));
        dMag = [strcat('^',cMag,'illion');strcat('^',cMag,'illiard')];
        dMag = reshape(dMag,1,[]);
    case 'indian'
        sSfx = '';
        dMag = strcat('^',cMag);
end
%
if isempty(sSfx) % indian
    jMag = w2nJoin(cMag(end:-1:1));
    rSpl = sprintf('(HUNDRED|THOUSAND|POINT|%s)',jMag);
else % short & long & peletier & rowlett
    jMag = w2nJoin(cMag,sSfx);
    rSpl = sprintf('(HUNDRED|THOUSAND|POINT|%s)',sSfx);
end
%
rXXa = sprintf('%sHundred(%s%s)?(%s)?',rSpc,rAnd,r099,rFra); % H&[0-99].[0-9]+
rXXb = sprintf('%s%sHundred',rFra,rSpc); % .[0-9]+H
rXX1 = sprintf('%s%s(%s)(%s|%s)',rCom,rSpc,rOne,rXXa,rXXb); % ,[0-9]("|")
rXX2 = sprintf('%s%s(%s)?',rAnd,r099,rFra); % &[0-99].[0-9]+
rXXX = sprintf('(%s|%s|%s)?',rXX1,rXX2,rFra); % ("|"|")
%
rEnd = sprintf('(?(?@%s)%s)',pNoP{1},rXXX);
%
if strcmpi(stpo.scale,'long') && strcmpi(stpo.mag,'simple')
    % First part:
    r1zz = sprintf('%s(%sHundred)?(%sThousand)?',rFra,rSpc,rSpc); % .[0-9]+HT
    r1yy = sprintf('%sThousand%s',rSpc,rXXX); % T("|"|")
    r2xx = sprintf('%s(%sThousand)?',rFra,rSpc); % .[0-9]+T
    r1xx = sprintf('%sHundred(%s%s)?(%s|%s)?',rSpc,rAnd,r099,r1yy,r2xx); % H&[0-99]("|")
    r1st = sprintf('(%s|%s|%s)',r1xx,r1yy,r1zz); % ("|"|")
    % Dynamic part:
    r2nd = sprintf('(%s(%sThousand)?|%sThousand(%s)?)?',rFra,rSpc,rSpc,rFra);
    % Magnitudes:
    rTh0 = sprintf('%sHundred(%s%s)?(%s)?',rSpc,rAnd,r099,rFra); % H&[0-99].[0-9]+
    rTh1 = sprintf('(%s)(%s|%s%sHundred)',rOne,rTh0,rFra,rSpc); % [0-9]("|.[0-9]+H)
    rTh2 = sprintf('%s(%s)?',r099,rFra); % [0-99].[0-9]+
    rTho = sprintf('(%s%s(%s|%s)%sThousand)?',rCom,rSpc,rTh1,rTh2,rSpc); % ,("|")T
    rMag = jMag;
else % short & indian & rowlett & peletier
    % First part:
```

```
    r1xx = sprintf('%sHundred(%s%s)?(%s)?',rSpc,rAnd,r099,rFra); % H&[0-99]
    r1yy = sprintf('%s(%sHundred)?',rFra,rSpc); % .[0-9]+H
    r1st = sprintf('(%s|%s)',r1xx,r1yy); % ("|")
    % Dynamic part:
    r2nd = sprintf('(%s)?',rFra);
    % Magnitudes:
    rTho = '';
    rMag = sprintf('(Thousand|%s)',jMag);
end
% Dynamic prefix:
if isempty(sSfx) % indian
    rDyn = sprintf('%s%s(%s%s)%s',rCom,rSpc,r099,r2nd,rSpc);
else % short & long & peletier & rowlett
    rDyn = sprintf('%s%s((%s)%s|%s%s)%s',rCom,rSpc,rOne,r1st,r099,r2nd,rSpc);
end
% All groups:
switch stpo.mag
    case 'compound'
        grp = cell(1,4);
        grp{4} = sprintf('(?(3)(%s%s)*)',rSpc,rMag);
    case 'simple'
        g = 7-isempty(rTho);
        grp = cell(1,g);
        grp{4} = sprintf('(?(3)(%s%s%s%s)?)',rSpc,pBef{1},rMag,pIni{1});
        grp{5} = sprintf('(?(4)%s)',pDyn{1});
        grp{6} = sprintf('(?(4)%s)',rTho);
        grp{g} = sprintf('(?(4)%s)',rEnd);
end
grp{1} = sprintf('(%s)',rSgn);
grp{2} = sprintf('(Infinit[ey]|Not%s\a%sNum(ber)?)?',rHyp,rHyp);
grp{3} = sprintf('(?(2)|%s%s?)',r099,r1st);
%
% Adjust character case and replace placeholders:
rgx = sprintf('%s',grp{:});
[rgx,igm] = w2nCase(rgx,stpo.case,pWht,pHyp,pDyn,pBef,pIni,pAft,pNoP);
rgx = sprintf('%s',rPfx,rgx,rSfx);
%
% Call REGEXP in a local function:
[tkn,spl] = w2nRgx(@fDyn,@fBef,@fIni,@fAft,txt,rgx,igm,'tokens','split');
%
% No numbers found:
if numel(spl)<2
    return
end
%
tkn = vertcat(tkn{:});
tkn = regexprep(tkn,'([^A-Z]+|(?<!THOUS)AND)','','ignorecase');
tkn = regexprep(tkn,rSpl,'$& ','ignorecase');
%
%% Number Parsing %%
%
```

```
sgn = 1-2*strcmpi(tkn(:,1),'NEGATIVE');
nmc = size(tkn,1);
num = zeros(1,nmc,stpo.class);
sgf =   nan(1,nmc);
idi = strncmpi(tkn(:,2),'INFINIT',7);
idn = strncmpi(tkn(:,2),'NOTANUM',7);
num(idi) = Inf*sgn(idi);
num(idn) = NaN*sgn(idn);
%
for ii = reshape(find(~(idi|idn)),1,[])
    mStr = sprintf(' %s',tkn{ii,3:end});
    mStr = regexprep(mStr,dOne,'${fAll($&)}','ignorecase');
    [mDig,mTyp] = regexpi(mStr,'[\d\s]+','match','split');
    mTyp(1) = [];
    mCnt = numel(mDig);
    mNum = zeros(mCnt,1);
    mMag =  ones(mCnt,1);
    % Magnitudes:
    for jj = mCnt:-1:1
        if isempty(mTyp{jj})
            mMag(1:jj) = 1;
        elseif strcmpi(mTyp{jj},'POINT')
            mMag(jj) = mMag(jj+1);
        elseif strcmpi(mTyp{jj},'HUNDRED')
            mMag(jj) = mMag(jj)*100;
        elseif strcmpi(mTyp{jj},'THOUSAND')
            mMag(1:jj) = mMag(jj)*1000;
        else
            mag = vMag(~cellfun('isempty',regexpi(mTyp{jj},dMag,'once')));
            switch stpo.mag
                case 'simple'
                    mMag(1:jj) = mag;
                case 'compound'
                    mMag(1:jj) = mag.*mMag(jj);
            end
        end
    end
    % Whole digits:
    mPoi = [find(strcmpi(mTyp,'POINT')),mCnt];
    for jj = 1:mPoi(1)
        mNum(jj) = sum(sscanf(mDig{jj},'%d'));
    end
    % Fraction digits:
    if numel(mPoi)>1
        [frc,int,sgd] = w2nFrac(sgn(ii),mDig{1+mPoi(1)},mMag(mPoi(1)));
        fNum = frc+sum(cast(int,stpo.class),'native');
    else
        fNum = 0;
        sgd = NaN;
    end
    % Sum of digits*magnitudes:
```

```
    num(ii) = fNum+sum(sgn(ii).*cast(mNum.*mMag,stpo.class),'native');
    % Significant figures:
    fgs = bsxfun(@plus,log10(mMag),floor(log10([mNum,mod(mNum,10)])));
    fgs(isinf(fgs)) = NaN;
    sgf(ii) = 1+max([fgs(:);sgd(:)])-min([fgs(:);sgd(:)]);
end
%
%% Nested Functions %%
%
    function fBef(str)
        % Store before magnitude:
        sIni = str;
    end
    function fAft(str)
        % Store after magnitude:
        sCur = str;
        sMag = str(1+numel(sIni):end);
    end
    function fIni(str)
        % Reset magnitude index:
        xNxt = numel(dMag);
        fAft(str);
    end
    function out = fDyn()
        % Create regular expression of possible magnitudes:
        xNxt = find(~cellfun('isempty',regexpi(sMag,dMag(1:xNxt),'once')))-1;
        if numel(regexpi(sCur,'POINT','once'))
            out = '(?!)';
            return
        elseif strcmpi(stpo.scale,'long')
            if strcmpi(sMag,'MILLION')
                out = '(?!)';
                return
            else
                tmp = w2nJoin(cMag(1:xNxt),sSfx);
            end
        elseif strcmpi(sMag,'THOUSAND')
            out = '(?!)';
            return
        elseif strcmpi(sMag,'MILLION') || strcmpi(sMag,'LAKH')
            tmp = 'Thousand';
        elseif strcmpi(stpo.scale,'peletier')
            xPel = 1:xNxt/2;
            if strcmpi('N',sMag(end)) % eg: (M|B|Tr)illi(on|ard)
                tmp = sprintf('Thousand|%s',w2nJoin(cMag(xPel),sSfx));
            else % eg: Trillion
                tmp = sprintf('Thousand|%s',dMag{xNxt}(2:end));
                if xNxt>2 % eg: Trillion|(M|B)illi(on|ard)
                    tmp = sprintf('%s|%s',rMag,w2nJoin(cMag(xPel),sSfx));
                end
            end
```

```
        else % short & indian & rowlett
            tmp = sprintf('Thousand|%s',w2nJoin(cMag(1:xNxt),sSfx));
        end
        out = sprintf('%s%s(%s)%s',rDyn,pBef{1},tmp,pAft{1});
        out = w2nCase(out,stpo.case,pWht,pHyp,pBef,pAft);
    end
%
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%words2num
function s = w2nJoin(c,t)
% Concatenate a cell array of strings with '|' as delimiter.
s = sprintf('|%s',c{:});
s(1) = [];
if nargin>1 && numel(t)
    s = sprintf('(%s)%s',s,t);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nJoin
function [rgx,igm] = w2nCase(rgx,opt,varargin)
% Adjust the case of the regular expression.
switch opt
    case 'title'
        rpl = '[Aa]';
        igm = 'matchcase';
    case 'upper'
        rpl = 'A';
        rgx = upper(rgx);
        igm = 'matchcase';
    case 'lower'
        rpl = 'a';
        rgx = lower(rgx);
        igm = 'matchcase';
    otherwise
        rpl = '[Aa]';
        igm = 'ignorecase';
end
% Replace \a in ' AND' & '-A-':
rgx = strrep(rgx,char(7),rpl);
% Replace placeholder text:
for k = 1:numel(varargin)
    rgx = strrep(rgx,varargin{k}{:});
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nCase
function varargout = w2nRgx(fDyn,fBef,fIni,fAft,varargin) %#ok<INUSL>
% Call REGEXP in a local function to allow for dynamic regular expressions.
[varargout{1:nargout}] = regexp(varargin{:});
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nRgx
function [num,spl,sgf] = w2nYllion(txt,stpo,pWht,pHyp,rHyp,rSgn,rSpc,r099,rDeF,rPfx,rSfx,dOne,fAll) %#ok<INUSD>
```

```
% Donald Knuth's exponential scale (aka "-Yllion").
%
% E-notation has better precision than using 10^N:
vMag = [1e8, 1e16, 1e32,    1e64,   1e128,  1e256];
cMag = {'M',  'B', 'Tr', 'Quadr', 'Quint', 'Sext'};%yllion
sSfx = 'yllion';
dMag = [{'Hundred';'Myriad'};strcat(cMag(:),sSfx)];
vMag = [       1e2;      1e4;             vMag(:)];
%
%% Create Regular Expression %%
%
% Normal:
rExt = sprintf(    '%s(%sHundred(%s%s)?)?',r099,rSpc,rSpc,r099);
rExt = sprintf(     '%s(%sMyriad(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf(    '%s(%sMyllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf(    '%s(%sByllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf(   '%s(%sTryllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf('%s(%sQuadryllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf('%s(%sQuintyllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
rExt = sprintf( '%s(%sSextyllion(%s%s)?)?',rExt,rSpc,rSpc,rExt);
% Highest:
rDgt = sprintf('%s',r099,rDeF); % [0-99].[0-9]+
rHgh = strrep(sprintf('(\b%syllion)?',cMag{:}),char(8),rSpc);
rHgh = sprintf('(%s(%sHundred)?(%sMyriad)?%s)',rDgt,rSpc,rSpc,rHgh);
% Not-a-number and infinite:
rNaN = sprintf('Infinit[ey]|Not%s\a%sNum(ber)?',rHyp,rHyp);
%
rgx = sprintf('(%s)(%s|%s|%s(%s)?)',rSgn,rNaN,rHgh,rExt,rDeF);
[rgx,igm] = w2nCase(rgx,stpo.case,pWht,pHyp);
rgx = sprintf('%s',rPfx,rgx,rSfx);
%
%% Locate any Number Substrings in String %%
%
num = zeros(0,0,stpo.class);
sgf = zeros(0,0);
%
[tkn,spl] = regexp(txt,rgx, 'tokens','split', igm);
%
if isempty(tkn) % no numbers found
    return
end
%
rSpl = sprintf('(HUNDRED|MYRIAD|POINT|%s)',sSfx);
%
tkn = vertcat(tkn{:});
tkn = regexprep(tkn,'[^A-Z]+','','ignorecase');
tkn = regexprep(tkn,rSpl,'$& ','ignorecase');
%
%% Convert Number Substrings to Numeric %%
%
sgn = 1-2*strcmpi(tkn(:,1),'NEGATIVE');
```

```
nmc = numel(sgn);
num = zeros(1,nmc,stpo.class);
sgf =   nan(1,nmc);
idi = strncmpi(tkn(:,2),'INFINIT',7);
idn = strncmpi(tkn(:,2),'NOTANUM',7);
num(idi) = Inf*sgn(idi);
num(idn) = NaN*sgn(idn);
%
for ii = reshape(find(~(idi|idn)),1,[])
    mStr = sprintf(' %s',tkn{ii,2:end});
    mStr = regexprep(mStr,dOne,'${fAll($&)}','ignorecase');
    [mDig,mTyp] = regexpi(mStr,'[\d\s]+','match','split');
    mTyp(1) = [];
    mCnt = numel(mDig);
    mNum = zeros(mCnt,1);
    mMag =  ones(mCnt,1);
    % Magnitudes:
    prv = [];
    for jj = mCnt:-1:1
        if jj>numel(mTyp) || isempty(mTyp{jj})
            mMag(1:jj) = 1;
        elseif strcmpi(mTyp{jj},'POINT')
            mMag(jj) = mMag(jj+1);
        else
            mag = vMag(~cellfun('isempty',regexpi(mTyp{jj},dMag,'once')));
            prv = [prv(prv>mag),mag];
            mMag(1:jj) = prod(prv);
        end
    end
    % Whole digits:
    mPoi = [find(strcmpi(mTyp,'POINT')),mCnt];
    for jj = 1:mPoi(1)
        mNum(jj) = sum(sscanf(mDig{jj},'%d'));
    end
    % Fraction digits:
    if numel(mPoi)>1
        [frc,int,sgd] = w2nFrac(sgn(ii),mDig{1+mPoi(1)},mMag(mPoi(1)));
        fNum = frc+sum(cast(int,stpo.class),'native');
    else
        fNum = 0;
        sgd = NaN;
    end
    % Sum of digits*magnitudes:
    num(ii) = fNum+sum(sgn(ii).*cast(mNum.*mMag,stpo.class),'native');
    % Significant figures:
    fgs = bsxfun(@plus,log10(mMag),floor(log10([mNum,mod(mNum,10)])));
    fgs(isinf(fgs)) = NaN;
    sgf(ii) = 1+max([fgs(:);sgd(:)])-min([fgs(:);sgd(:)]);
end
%
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nYllion
function [frc,int,sgd] = w2nFrac(sgn,txt,mag)
% Convert string of fraction digits into whole and fractional numeric.
dgt = sscanf(txt,'%u',[1,Inf]);
adj = mag./(10.^(1:numel(dgt)));
idx = round(adj)<1;
tmp = 10.^(nnz(idx):-1:0);
frc = sgn.*sum((dgt(1,idx).*tmp(2:end)))./tmp(1);
int = sgn.*dgt(~idx).*adj(~idx);
sgd = log10(adj);
sgd(1:find(dgt,1,'first')-1) = [];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nFrac
function stpo = w2nOptions(stpo,opts)
% Options check: only supported fieldnames with suitable option values.
%
ofc = fieldnames(opts);
%
for k = 1:numel(ofc)
    ofn = ofc{k};
    idf = strcmpi(ofn,ofc);
    if nnz(idf)>1
        error('SC:words2num:options:DuplicateOptionNames',...
            'Duplicate field names:%s\b.',sprintf(' <%s>,',ofc{idf}))
    end
    arg = opts.(ofn);
    dfn = lower(ofn);
    switch dfn
        case {'and','comma','hyphen','sign','space'}
            w2nLogic()
        case {'prefix','suffix'}
            w2nRegex()
        case 'class'
            w2nString↙
('double','single','int8','int16','int32','int64','uint8','uint16','uint32','uint64')
        case 'case'
            w2nString('ignore','upper','lower','title')
        case 'scale'
            w2nString('short','long','indian','peletier','rowlett','yllion')
        case 'mag'
            w2nString('simple','compound')
        case 'white'
            w2nWhite()
        otherwise
            dfs = sort(fieldnames(stpo));
            error('SC:words2num:options:UnknownOptionName',...
                'Unknown option:%s\b.\nKey/Option names must be:%s\b.',...
                ofn,sprintf(' <%s>,',dfs{:}))
    end
    stpo.(dfn) = arg;
end
```

```
%
%% Nested Functions %%
%
    function w2nLogic() % logical scalar or empty numeric.
        assert(isequal(arg,0)||isequal(arg,1)||(isnumeric(arg)&&isequal(arg,[])),...
            sprintf('SC:words2num:%s:NotScalarLogicalNorEmptyNumeric',dfn),...
            'The <%s> value must be a scalar logical or an empty numeric.',dfn)
        arg = logical(arg);
    end
    function w2nRegex() % regular expression.
        assert(n2wIsCRV(arg) || isequal(arg,''),...
            sprintf('SC:words2num:%s:NotRegularExpression',dfn),...
            'The <%s> value must be a string scalar or a character row vector.',dfn)
    end
    function w2nString(varargin) % text.
        if ~n2wIsCRV(arg)||~any(strcmpi(arg,varargin))
            tmp = sprintf(' <%s>,',varargin{:});
            error(sprintf('SC:words2num:%s:UnknownOptionValue',dfn),...
                'The <%s> value must be one of:%s\b.',dfn,tmp);
        end
        arg = lower(arg);
    end
    function w2nWhite() % whitespace.
        assert(n2wIsCRV(arg),...
            sprintf('SC:words2num:%s:NotCharRowNorStringScalar',dfn),...
            'The <%s> value must be a string scalar or a character row vector.',dfn)
        assert(all('-'~=arg(2:end)),...
            sprintf('SC:words2num:%s:Minus1stChar',dfn),...
            'In <%s> minus can be the first character only.',dfn)
        assert(isempty(regexp(arg,'(?<!\\)\\$','once')),...
            sprintf('SC:words2num:%s:NotEscapedBackslash',dfn),...
            'In <%s> the backslash character must be escaped.',dfn)
        arg = sprintf('[%s]',arg);
        assert(isempty(regexpi('A':'Z',arg,'once')),...
            sprintf('SC:words2num:%s:AlphabetMatch',dfn),...
            'The <%s> regular expression must not match alphabetic characters.',dfn)
    end
%
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2nOptions
function out = n2wIsCRV(txt)
% TXT is character row vector.
szv = size(txt);
out = ischar(txt) && numel(szv)==2 && szv(1)==1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%n2wIsCRV
function arr = w2n1s2c(arr)
% If scalar string then extract the character vector, otherwise data is unchanged.
if isa(arr,'string') && isscalar(arr)
    arr = arr{1};
end
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%w2n1s2c
```