1. **integerize.m**

function cl = integerize(A)

cls = {'int8'; 'int16'; 'int32'; 'int64'};

cl = 'NONE';

mx = max(A(:));

mn = min(A(:));

for ii = 1:length(cls)

if intmax(cls{ii}) >= mx && intmin(cls{ii}) <= mn

cl = cls{ii};

break;

end

end

end

OR

function Name = integerize (A) DataType = {'int8', 'int16', 'int32', 'int64', 'NONE'};

Limit = [2^7, 2^15, 2^31, 2^63, realmax];

A(A<0) = A(A<0) + 1; % must do this for negatives!!!!!

Name = DataType{max(abs(A(:))) < Limit};

end

1. **year2016.m**

function month = year2016(m)

if ~isscalar(m) || m < 1 || m > 12 || m ~= floor(m)

month = [];

return;

end

days = ([31 29 31 30 31 30 31 31 30 31 30 31]);

ms = {'January'; 'February'; 'March'; 'April'; 'May'; 'June'; ...

'July'; 'August'; 'September'; 'October'; 'November'; 'December'};

ds = {'Sun'; 'Mon'; 'Tue'; 'Wed'; 'Thu'; 'Fri'; 'Sat'};

start = 4; % Jan 1, 2016 was a Friday. US week starts on Sunday.

% We'll add ii and 1 below because rem(n,7) returns numbers

% 0-6 and we need indexes 1-7.

start = start + sum(days(1:m-1));

for ii = 1:days(m)

month(ii).month = ms{m};

month(ii).date = ii;

month(ii).day = ds{rem(start+ii,7)+1};

end

end

1. **palin\_product.m**

**Naive implementation. Slow for many cases:**

function n = palin\_product(dig, lim)

n = 0;

for ii = 10^(dig-1):10^dig-1

for jj = 10^(dig-1):ii

p = ii\*jj;

if p >= lim

continue;

elseif palindrome(p) && p > n

n = p;

end

end

end

end

function isp = palindrome(p)

txt = num2str(p);

isp = strcmp(txt,txt(end:-1:1));

end

**Version optimized for speed. Most of the time, the inner loop ends early:**

function n = palin\_product(dig, lim)

n = 0;

for ii = 10^dig-1 : -1 : 10^(dig-1) % going from large to small

for jj = min(10^dig-1,floor((lim-1)/ii)) : -1 : 10^(dig-1) % check numbers under lim

p = ii\*jj;

if p < n % gone under the current max

continue; % no need to go further in the inner loop

elseif palindrome(p)

n = p; % found a larger one

continue; % no need to go further in the inner loop

end

end

end

end

function isp = palindrome(p)

txt = num2str(p);

isp = strcmp(txt,txt(end:-1:1));

end

**Vectorized solution, but needs lots of memory:**

function n = palin\_product (dig,lim)

% a is the smallest dig-digit number that can be formed. If the smallest possible

% product (a^2) is smaller than the specified limit, we determine b, the largest

% dig-digit number that can be formed. We then build the square outer product of a:b.

% Logically indexing into to this matrix for elements less than lim creates a column

% vector P of candidate products. We convert each of these to a string, reverse its

% characters, and convert it back to a number, to form the column vector Q. Finally,

% we return the maximum element in P which has the same value in both P and Q.

n = 0;

a = 10^(dig-1);

if lim>a^2

b = 10^dig - 1;

P = (a:b)' \* (a:b);

P = P(P<lim);

Q = str2num(fliplr(num2str(P)));

n = max(P(P==Q));

end

end

1. **dial.m**

***Traditional Solution***

function num = dial(str)

num = uint64(0);

if length(str) > 16

return;

end

for ii = 1:length(str)

if str(ii) >= 'A' && str(ii) <= 'Z'

str(ii) = map(str(ii));

elseif ~(str(ii) >= '0' && str(ii) <= '9')

return;

end

end

num = uint64(str2num(str));

end

function ch = map(ch)

m = '22233344455566677778889999';

ch = m(ch - 'A' + 1);

end

***Vectorized Version***

function ph = dial(str)

code = '0123456789xxxxxxx22233344455566677778889999'; % x represents invalid character

ph = '0';

n = str-'0'+1; % index into the vector code

if ~((sum(str(n <= 0)) + sum(n > length(code))) || ... % checks for indexes out of range

sum(code(n) == 'x') || ... % checks for any x-s

length(str) > 16 ) % checks too long input

ph = code(n); % mapping with a single command

end

ph = uint64(str2num(ph)); % convert string to number and uint64

end

***OR***

function n = dial (s)

if ~all(ismember(s,['0':'9','A':'Z'])) || length(s) > 16

n = uint64(0);

else

map = '22233344455566677778889999';

s(s>='A') = map(s(s>='A')-64);

n = uint64(str2double(s)) ;

end

end

1. **logi\_unpack.m**

function L = logiunpack(cv)

n = length(cv);

L = false(n);

for ii = 1:n

for jj = 1:length(cv{ii})

L(ii,cv{ii}(jj)) = true;

end

end

end

1. **logi\_pack.m**

function cv = logipack(L)

[r c] = size(L);

cv = cell(1,r);

for ii = 1:r

cv{ii} = find(L(ii,:));

if isempty(cv{ii}) % find can return 1x0 empty arrays

cv{ii} = []; % so we make sure it is 0x0

end

end

end

1. **centuries.m**

function c = centuries(n)

if ~isscalar(n) || n < 1 || n > 3000 || n ~= floor(n)

c = '';

else

cents = {'I'; 'II'; 'III'; 'IV'; 'V'; 'VI'; 'VII'; 'VIII'; 'IX'; 'X';

'XI'; 'XII'; 'XIII'; 'XIV'; 'XV'; 'XVI'; 'XVII'; 'XVIII'; 'XIX'; 'XX';

'XXI'; 'XXII'; 'XXIII'; 'XXIV'; 'XXV'; 'XXVI'; 'XXVII'; 'XXVIII'; 'XXIX'; 'XXX'};

c = cents{ceil(n/100)};

end

end

OR

function c = centuries (y)

c = '';

if isscalar(y) && rem(y,1)==0 && y>0 && y<=3000

c = A2R(fix((y-1)/100)+1);

end

end

function R = A2R (A)

% Converts Arabic numbers to Roman strings.

Roman = {'I' 'IV' 'V' 'IX' 'X' 'XL' 'L' 'XC' 'C' 'CD' 'D' 'CM' 'M'};

Arabic = {1 4 5 9 10 40 50 90 100 400 500 900 1000};

R = ''; k = 13;

while k>0 % remove largest modulii first

if A>=Arabic{k} % if value>current modulus

A = A-Arabic{k}; % remove modulus from value

R = [R Roman{k}]; % append Roman character

else

k = k-1; % else consider next smaller modulus

end

end

end

1. **find\_zero.m**

function x = find\_zero (f, x1,x2)

x = (x1+x2)/2.0; % find interval midpoint

while abs(f(x)) > 1e-10 % are we there yet?

if f(x1)\*f(x)>0 % if f(left) and f(mid) have the same sign

x1 = x; % move left to mid

else

x2 = x; % move right to mid

end

x = (x1+x2)/2.0; % recalculate midpoint

end

end