Solutions to Homework 6

Help Center

Problem neighbor

```
function w = neighbor(v)
    w = [];
    if min(size(v)) == 1                    % must be a vector
        for ii = 1:length(v)-1              % if length is less than 2, loop won't do a
nything
            w(ii) = abs(v(ii+1) - v(ii));
        end
    end
end
```

Problem neighbor (alternative solution)

no explicit loop

```
function w = neighbor(v)
    if length(v) < 2 || min(size(v)) ~= 1  % must be a vector of at least two elements
        w = [];
    else
        w = abs(v(1:end-1)-v(2:end));      % take the difference of two subvectors
    end                                    % of length (n-1)
end
```

Problem replace_me

builds up the output one element at a time

```
function w = replace_me(v,a,b,c)
    if nargin < 3
        b = 0;
    end
    if nargin < 4
        c = b;
    end
    w = [];
    for k = 1:length(v);
```

```
        if v(k) == a        % if a is found,
            w = [w,b,c];    % we insert b and c at the end of the current w
        else                % otherwise,
            w = [w,v(k)];   % we insert the original element of v
        end
    end
end
```

Problem replace_me (alternative solution)

only changes the output vector when an instance of a is found

```
function w = replace_me(v,a,b,c)
    if nargin < 3
        b = 0;
    end
    if nargin < 4
        c = b;
    end
    w = v;                                      % make w the same as v
    wi = 1;                                     % wi is used to index into w
    for vi = 1:length(v)
        if v(vi) == a
            w = [w(1:wi-1) b c w(wi+1:end)];    % insert b and c at position wi
            wi = wi + 1;                        % increment wi
        end
        wi = wi + 1;                            % wi is incremented in either case
    end
end
```

Problem halfsum

using nested loops

```
function s = halfsum(A)
    [row col] = size(A);
    s = 0;
    for ii = 1:row
        for jj = ii:col         % the column index only starts at the current row index
            s = s + A(ii,jj);
        end
    end
end
```

Problem halfsum (alternative solution)

using a single loop and sum

```
function s = halfsum(A)
    [nr,~] = size(A);
    s = 0;
    for r = 1:nr                    % for each row
        s = s + sum(A(r,r:end));    % sum adds up the elements right of the diagonal (inclusive)
    end                             % in the current row
end
```

Problem large_elements

```
function found = large_element(A)
    [row col] = size(A);
    found = [];
    for ii = 1:row
        for jj = 1:col
            if A(ii,jj) > ii + jj       % if the element is larger than the sum of its indexes
                found = [found; ii jj]; % add a new row to the output matrix
            end
        end
    end
end
```

problem one_per_n

using while-loop

```
function n = one_per_n(x)
    n = 0;
    sum = 0;
    while sum < x && n <= 10000
        n = n + 1;
        sum = sum + 1/n;
    end
    if n > 10000
        n = -1;
    end
end
```

problem one_per_n (alternative solution)

using for-loop

```
function n = one_per_n(x)
    s = 0;
    for n = 1:1e4
        s = s + 1/n;
        if s >= x
            return;
        end
    end
    n = -1;
end
```

Problem approximate_pi

```
function [a,k] = approximate_pi(delta)
    k = 0;
    f = sqrt(12);                      % compute sqrt(12) only once
    a = f;                             % the value of a for k == 0
    while abs(pi-a) > delta            % while we are further away than delta
        k = k + 1;                     % increment k
        a = a + f*(-3)^(-k)/(2*k+1);   % add increment to current value of a
    end
end
```

Problem separate_by_two

using division and rounding

```
function [even,odd] = separate_by_two(A)
    even = A(fix(A/2) == A/2)';  % if A is even, rounding does not do anything to A/2
    odd  = A(fix(A/2) ~= A/2)';  % if A is odd, it gets rid of the .5 part, so they wo
n't be equal
end
% note that this will put non-integers into odd
```

Problem separate_by_two (alternative solution)

using mod (or rem)

```
function [even, odd] = separate_by_two(A)
```

```
    even = A(mod(A,2) == 0)';   % mod gives 0 if even
    odd  = A(mod(A,2) == 1)';   % mod gives 1 if odd
end
% note that this one will not put non-integers in any of the outputs
```

Problem separate_by_two (alternative solution)

using mod (or rem)

```
function [even,odd] = separate_by_two(A)
    mod2 = logical(mod(A,2));
    even = A(~mod2)';       % modulo 2 is zero for even numbers (logical false), so we need to negate it
    odd  = A(mod2)';        % modulo 2 is non-zero for odd numbers, that is, logical true
end
% note that this will put non-integers into odd
```

Problem divvy

```
function A = divvy (A,k)
    L = (mod(A,k) ~= 0);    % creates a logical matrix based on divisibility by k
    A(L) = k * A(L);        % changes only the non-divisible elements of A by multiplying them by k
end
% uses A as both input and output, so we only need to modify some elements of A
```

Problem divvy (alternative solution)

single line solution

```
function I = divvy(I,k)
    I(mod(I,k) ~= 0) = I(mod(I,k) ~= 0) * k;
end
% same solution as above, but it repeats the modulo computation
```

Problem square_wave

using a for-loop

```
function sq = square_wave(n)
    t = 0 : 4*pi/1000 : 4*pi;           % setup vector according to the specs
    sq = zeros(1,length(t));            % initialize output to 0
    for ii = 1:2:2*n                    % run for first n odd numbers (2k-1)
        sq = sq + cos(ii*t-pi/2)/ii;    % add the next cosine term
    end
end
```

Problem square_wave (alternative solution)

tricky code with no explicit loops

```
function s = square_wave(n)
    t = 0 : 4*pi/1000 : 4*pi;   % setup vector according to the specs
    idx = (2*(1:n)' - 1);       % make column vector of fist n odd numbers (2k-1)
    % idx*t makes a matrix; each row is (2k-1)*t, for a given k
    % idx*ones(size(t)) also makes a matrix; each element of row k is just (2k-1)
    % sum down the columns
    s = sum(sin(idx*t) ./ (idx*ones(size(t))),1);
end

% the second argument to sum is needed in case n is 1
% remember that sum(x) sums x along columns unless x is a row vector!
```

Problem my_prime

using a for-loop

```
function a = myprime(n)
    a = false;
    if n > 1                    % 1 is by definition not prime
        for ii = 2:sqrt(n)      % see explanation below
            if ~mod(n,ii)
                return;
            end
        end
        a = true;
    end
end
% x is prime if it is NOT divisible by all integers from 2 to sqrt(x)
% because factors have to come in pairs -- one bigger than sqrt(x) and
% one smaller (or both equal)
```

Problem my_prime (alternative solution)

with no explicit loops

```
function prim = myprime(p)
    v = 2:sqrt(p);
    v = v(rem(p,v) == 0);           % if p is prime, none of the remainders can be 0
    prim = ~length(v) && (p ~= 1);  % so if v has any elements, p is not prime
end                                 % 1 is handled by the (p ~= 1) condition
```

Published with MATLAB® R2014a

Created Tue 2 Jun 2015 10:21 AM PDT

Last Modified Mon 23 Nov 2015 6:11 PM PST