Fast implementation of logsum for MATLAB

Idea: sort all n terms to be added in an increasing order, then add all possible pairs of consecutive terms. In MATLAB, this is much faster than a linear loop (term by term), especially when n is large. Iterations are denoted with $k = 0, 1, ..., k_{\max}$, where $k_{\max} = \lfloor \log_2 n \rfloor$. For each iteration k:

- The number of terms remaining from the previous iteration $k-1$ is $p_k = \lceil n \cdot 2^{-k} \rceil$.
- The number of logsum operation to do is $q_k = \lfloor \frac{p_k}{2} \rfloor$.

An example is given in Fig. 1, for n=14.

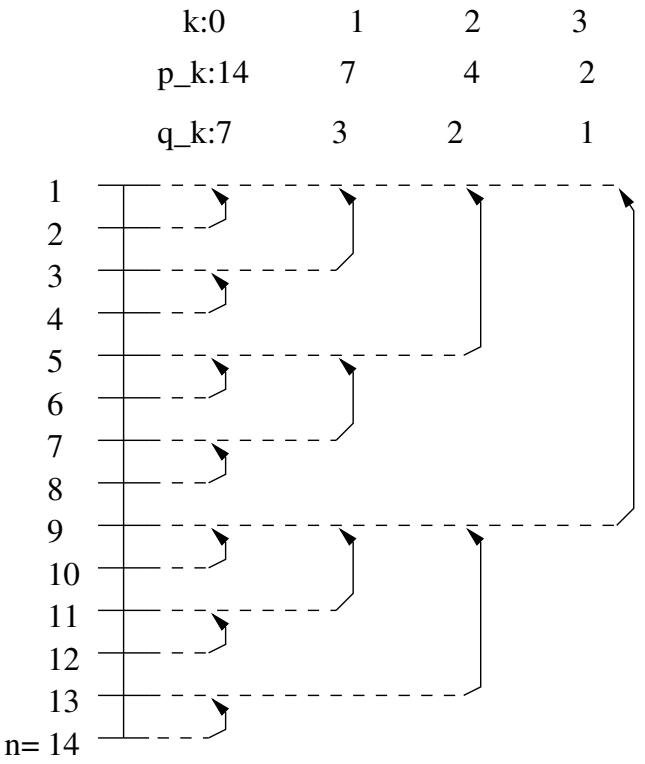

Figure 1: Example of logsum implementation for n=14 terms. k=0,1,2,3 designate the 4 iterations. Arrows indicate which pairs of terms are summed at each iterations, and point at where the result is stored. Note that the n terms are in increasing order.

Below is the MATLAB code:

```
% FUNCTION Y = MY_LOGSUM_FAST( LOG_X )
% log_x : vector ( or matrix )containing log(xi)
% y     : scalar ( or row ), estimation of log(sum over i of xi)
%
% In case of a matrix we sum the columns
%
% Note that :
%  my_logsum( [ -Inf -Inf ] ) returns -Inf
%  my_logsum( [ -Inf -Inf 2 ] ) returns 2
%
% FUNCTION Y = MY_LOGSUM_FAST( LOG_X, LOG_Y )
%
% Term-by-term log summation of two matrices of same dimension and
% size. Returns Y, a matrix of same size as LOG_X and LOG_Y:
%
% Y = LOG( EXP( LOG_X ) + EXP( LOG_Y ) )

function y = my_logsum_fast( log_x, opt_log_y )

  if nargin < 1
    error( 'logsum needs at least one input argument' );
  end

  if nargin > 1
    % Two matrices
    siz_x = size( log_x );
    siz_y = size( opt_log_y );
    if length( siz_x ) ~= length( siz_y )
      error( 'logsum: the two matrices must have same dimension and same size!' );
    end
    if ~all( siz_x == siz_y )
      error( 'logsum: the two matrices must have same dimension and same size!' );
    end

    if ~any( siz_x )
      % Empty matrices
      y = log_x;
      return;
    end

    M = max( log_x, opt_log_y );
    y = M + log( 1 + exp( min( log_x, opt_log_y ) - M ) );

    % Return output value
    return;
```

```
end

siz = size( log_x );

n = length( find( siz > 1 ) );

if n > 2
  error( [ mfilename ': can only deal with 2-D matrices.' ] );
end

if n == 0
  % Nothing to do
  y = log_x;
  return;
end

if n == 1
  % Vector
  log_x = log_x(:);
end

% General case: matrix
n_terms = size( log_x, 1 );

% Start with the smallest value
log_x = sort( log_x, 1 );

% We do it in a hierarchical manner
% At each iteration (k = 0...k_max),
% we sum as many pairs of terms as possible
k_max = log2( n_terms );

for k = 0:k_max

  p_k = ceil( n_terms * (2^-k) );
  q_k = floor( p_k * 0.5 );
  t_k = 1 + ( 0:q_k-1 ) * 2 ^ (k+1);
  u_k = t_k + 2^k;

  log_x( t_k, : ) = log_x( u_k, : ) + ...
     log( 1 + exp( log_x( t_k, : ) - log_x( u_k, : ) ) );

  [r,c] = find( isnan( log_x( t_k, : ) ) );
  if ~isempty( r )
    log_x( sub2ind( size( log_x ), t_k( r ).', c ) ) = -Inf;
  end
```

```
end

% Return the result
y = log_x( 1,: );
```