```python
# Generated with SMOP  0.41
from libsmop import *


@function
def pool_adjacent_violators(h=None,inc=None,*args,**kwargs):
    varargin = pool_adjacent_violators.varargin
    nargin = pool_adjacent_violators.nargin

    # Compute the isotonic regression of an histogram h.
    # inc = boolean value saying if we want the non-decreasing (inc = 1) or decreasing regression
    #   Copyright (c) 2016 Julie Delon

    if (inc == 0):
        g=copy(h)
        for i in arange(1,length(h)).reshape(-1):
            som=g(i)
            for j in arange(i - 1,1,- 1).reshape(-1):
                if (j == 1 or (dot(g(j),(i - j)) >= som)):
                    som=som / (i - j)
                    for k in arange(j + 1,i).reshape(-1):
                        g[k]=som
                    break
                som=som + g(j)
    if (inc == 1):
        g=copy(h)
        for i in arange(length(h) - 1,1,- 1).reshape(-1):
            som=g(i)
            for j in arange(i + 1,length(h)).reshape(-1):
                if (j == length(h) or (dot(g(j),(j - i)) >= som)):
                    som=som / (j - i)
                    for k in arange(i,j - 1).reshape(-1):
                        g[k]=som
                    break
                som=som + g(j)
    return g



@function
def max_entropy(h=None,a=None,b=None,e=None,inc=None,*args,**kwargs):
    varargin = max_entropy.varargin
    nargin = max_entropy.nargin

    # Compute the maximum entropy of the histogram h(a:b) for the increasing or decreasing hypothesis
    # inc = boolean value indicating if we test the increasing or decreasing hypothesis
    # h = histogram
    # e = parameter used to compute the entropy
    # See
    #   Copyright (c) 2016 Julie Delon
    g=h(arange(a,b))
    decreas=pool_adjacent_violators(g,inc)
    L=length(g)
    # integrate signals
    g=cumsum(g)
    decreas=cumsum(decreas)
    # meaningfullness threshold
    N=g(L)
    seuil=(log(dot(L,(L + 1)) / (2)) + dot(e,log(10))) / N
    # search the most meaningfull segment (gap or mode)
    max_entrop=0.0
    for i in arange(1,L).reshape(-1):
        for j in arange(i,L).reshape(-1):
            if (i == 1):
                r=g(j)
            else:
                r=g(j) - g(i - 1)
            r=r / N
            if (i == 1):
                p=decreas(j)
            else:
                p=decreas(j) - decreas(i - 1)
            p=p / N
            v=entrop(r,p)
            if (v > max_entrop):
                max_entrop=copy(v)
    max_entrop=dot((max_entrop - seuil),N)
    return max_entrop



@function
def entrop(x=None,y=None,*args,**kwargs):
    varargin = entrop.varargin
    nargin = entrop.nargin
```

```
    # function computing the entropy between x and y. x and y must be in the interval [0,1]
    if x == 0.0:
        v=- log10(1 - y)
    else:
        if x == 1.0:
            v=- log10(y)
        else:
            v=(dot(x,log10(x / y)) + dot((1.0 - x),log10((1.0 - x) / (1.0 - y))))
    return v



#################
# The following functions implement the Fine to Coarse Histogram Segmentation described in
# J. Delon, A. Desolneux, J-L. Lisani and A-B. Petro, A non parametric approach for histogram segmentation
# IEEE Transactions on Image Processing, vol.16, no 1, pp.253-261, Jan. 2007.
# Usage :
#   u = double(imread('../images/lena.png'));
#   H = hist(u(:),0:255);
#   idx=FTC_Seg(H,0);
# idx should contain the list of all minima separating the modes of H
#   Copyright (c) 2016 Julie Delon
#################
@function
def FTC_Seg(H=None,e=None,*args,**kwargs):
    varargin = FTC_Seg.varargin
    nargin = FTC_Seg.nargin

    # FTC_seg
    # H = the integer-valued histogram to be segmented.
    # e = parameter of the segmentation
    # (corresponds to e = -log10(epsilon) in the paper)
    # large e => coarse segmentation
    # small e => fine segmentation

    lH=length(H)
    ## find the list of local minima and maxima of H
    p,idx_max=findpeaks(H,nargout=2)
    p,idx_min=findpeaks(- H,nargout=2)
    idx=sort(concat([idx_min,idx_max]))
    if idx(1) != 1:
        idx=concat([1,idx])

    if idx(end()) != lH:
        idx=concat([idx,lH])

    # find if idx starts with a minimum or a maximum
    if H(idx(1)) < H(idx(2)):
        begins_with_min=1
    else:
        begins_with_min=0

    ## FILL THE LIST OF ENTROPIES FOR ALL MODES
    # The merging of two contiguous modes [a,b] and [b,c] can be done in two ways,
    # either by using the maximum M1 on [a,b] and by testing the decreasing hypothesis on [M1,c],
    # or by using the maximum M2 on [b,c] and by testing the increasing hypothesis on [a,M2].
    # For each configuration, we compute the entropy of the worst interval against the considered hypothesis.
    K=length(idx)
    val=zeros(1,K - 3)
    # Loop on all optimas
    for k in arange(1,K - 3).reshape(-1):
        # decide if we want to test the increasing or decreasing hypothesis on
        # [idx(k),idx(k+3)]
        if (logical_or((begins_with_min and mod(k,2) == 1),(logical_not(begins_with_min) and mod(k,2) == 0))):
            inc=1
        else:
            inc=0
        # compute the max entropy on the interval [k,k+3]
        val[k]=max_entropy(H,idx(k),idx(k + 3),e,inc)

    ######## MERGING of MODES
    valmin,kmin=min(val,nargout=2)

    while (logical_not(isempty(val)) and valmin < 0):
        # update the list of min, max
        idx=concat([idx(arange(1,kmin)),idx(arange(kmin + 3,end()))])
        val=concat([val(arange(1,min(kmin,end()))),val(arange(kmin + 3,end()))])
        val=val(arange(1,length(idx) - 3))
        for j in arange(max(kmin - 2,1),min(kmin,length(val))).reshape(-1):
            # decide if increasing or decreasing
            if logical_or((begins_with_min and mod(j,2) == 1),(logical_not(begins_with_min) and mod(j,2) == 0)):
                inc=1
            else:
                inc=0
```

```
            # update the max entropy on the interval [k,k+3]
            val[j]=max_entropy(H,idx(j),idx(j + 3),e,inc)
        valmin,kmin=min(val,nargout=2)

    if (begins_with_min):
        idx=idx(arange(1,end(),2))
    else:
        idx=idx(arange(2,end(),2))

    ## Display the segmentation
    bar(H,'r')
    hold('on')
    for k in arange(1,length(idx)).reshape(-1):
        line(concat([idx(k),idx(k)]),concat([0,max(ravel(H))]))

    hold('off')
    return idx
```