This is a 64-channel version tandem algorithm implemented by K. Hu for the paper “[Unvoiced speech separation from nonspeech interference via CASA and spectral subtraction](http://www.cse.ohio-state.edu/~dwang/papers/Hu-Wang.taslp11.pdf),” by **K. Hu** and D. L. Wang in IEEE Trans. Audio, Speech, and Lang. Process., vol. 19, pp. 1600-1609, 2011.

The tandem algorithm is a voiced speech separation and pitch tracking algorithm described in “A tandem algorithm for pitch estimation and voiced speech segregation,”  by G. Hu and D. L. Wang in IEEE Trans. Audio, Speech, Lang. Process., vol. 18, pp. 2067-2079, 2010.

**Usage**: tandem in out  
 tandem in out cross evCross  
 tandem in out cross evCross eng

Inputs:

* in: An ASCII file containing a 20 kHz waveform noisy speech signal

Outputs:

* out.64.pitchT.dat: Estimated pitch contours w/ T-segments (see the definition of T-segments in Sect. V-C of the Hu & Wang’10 paper)
* out.64.maskT.dat: Estimated binary masks w/ T-segments
* out.64.pitch.dat: Estimated pitch contours w/o T-segments (optional)
* out.64.mask.dat: Estimated ratio masks w/o T-segments (optional)
* cross: Cross-channel correlations (optional)
* evCross: Envelope cross-channel correlations (optional)
* eng: Energy of T-F units (optional)

Run an example:

* tandem sample/mixture.20k out

More about the inputs and outputs

* This algorithm uses a 64-channel gammatone filterbank
* Sampling frequency of the input signal must be 20kHz
* out.64.pitchT.dat: The two numbers in the first row denote number of pitch contours and number of frames, and each following row denotes an estimated pitch contour
* out.64.maskT.dat: Simultaneous streams. Each row is a frame-level binary mask (64-dimensional) associated with a pitch point. The order of rows matches that of the pitch points (from left to right in a pitch contour) and pitch contours (from top to bottom)
* The "net" folder needs to be in the same directory as the executable

**Some explanations about source files**:

* tandem.cpp: The main C++ file consisting of functions corresponding to different processing stages
  + Major functions
    - voicedMaskEst: The iterative estimation of pitch contours and simultaneous streams (Sect. V-A and V-B of the paper)
    - onOffSeg: Onset/offset-based segmentation (Sect. V-C)
    - expandVoicedMask: Generate and incorporate T-segments (Sect. V-C)
* gammaTone.cpp (and gammaTone.h): Gammatone filtering
  + Class gammaToneFilter
    - Data members
      * cf: Gammatone filter center frequency
      * bw: Gammatone filter bandwidth
    - Functions
      * Filtering: Filter the input signal using a gammatone filter with a specific center frequency
      * oneStep: Take a single value in the input signal and do the filtering
  + Class gammaToneFilterBank
    - Data members
      * lowerCF: Lower cutoff frequency in gammatone filtering
      * upperCF: Higher cutoff frequency in gammatone filtering
      * gf: Gammatone filters
      * sf: Sampling frequency
      * response: Filtered signals
    - Functions
      * HzToERBRate: Convert Hz to ERB rate
      * ERBRateToHz: Convert ERB rate to Hz
      * filtering: Filter a signal using a gammatone filterbank
* feture.cpp (and feature.h): Extracting features such as autocorrelations, cross-channel correlations, and the 6-dimensional pitch-based feature
  + Class feature
    - Data members
      * acfLen: Length of signal in computing auto-correlation
      * acfOrder: Used for zero-padding in FFT
      * bandPass: Bandpass filters
      * envelope: Envelopes of filtered signals
      * window: Length of a frame
      * min\_delay: Minimum delay in computing auto-correlation function
      * max\_delay: Maximum delay in computing auto-correlation function
      * Theta\_p: A threshold for single-unit probabilities (See Eq. (17), .5 by default)
      * corrLgm: A struct storing 6-dimensional features
      * data: Temporary memory used by FFT
    - Functions
      * computeFeature: Extract envelopes, auto-correlations and cross-channel correlations
      * fftACF: Auto-correlations based on FFT
      * computeCross: Compute cross-channel correlations
      * newFeature: Initialize variables storing various features
      * deleteFeature: Free dynamically allocated variables
* pitch.cpp (and pitch.h): Pitch and mask estimation
  + Class pitchMask
    - Data members
      * sNet: MLP for single-unit labeling
      * mNet: MLP for multiple-unit labeling
      * pNet: MLP for differentiating pitch and its integer multiples
      * Pitch: Estimated pitch contours
    - Functions
      * readNet: Load trained MLPs
      * singleUnitProb: Unit labeling based on 6-dimensional features (Sect. III-A in the paper)
      * multiUnitProb: Unit labeling based on neighboring T-F units (Sect. III-C)
      * maskToPitchACF: Estimate pitch based on auto-correlation functions
      * maskToPitchML: Estimate pitch based on probabilities (Eq. (10) in Sect. IV-A)
      * maskToPitchML2: Estimate pitch based on mask labels (Eq. (9) in Sect. IV-A)
      * maskToPitchMAP: Estimate pitch and then test whether it is an octave error (using function compareTwoCandidateMAP)
      * compareTwoCandidateMAP: Differentiate true pitch from its integer multiples (Sect. IV-B). Note that in this implementation the first quantity of the 3-dimensional vector described in Sect. IV-B of the paper is split into its integral and fractional parts, resulting in a 4-dimensional feature and a little better performance.
* voicedMask.cpp (and voicedMask.h): The iterative procedure (Sect. V)
  + Class voicedMask
    - Data members
      * thd\_cross: Threshold for cross-channel correlations (See Eq. (13), .935 by default)
      * thd\_evCross: Threshold for envelope cross-channel correlations (See Eq. (13), .94 by default)
    - Functions
      * dtmPitchMask: The iterative procedure (Sect. V)
      * initPitchEst: Initial estimation (Sect. V-A)
      * iterativePitchEst: Iterative estimation (Sect. V-B)
      * maskToPitch: Estimate pitch contours based on masks (also deal with the splitting of a pitch contour)
      * expandMask: Expand pitch contours
      * findContour: Generating pitch contours from pitch points
      * checkPitchCon: Check pitch continuity
      * checkMaskCon: Check mask continuity
      * isOverlap: Refine two pitch contours overlapping in time
      * isConnected: Check whether two pitch contours can be connected together
      * mergeContour: Merge pitch contours
      * reDetermineMask: Estimate masks at a frame containing multiple harmonic sources (Sect. III-B)
      * removeDuplicate: Remove pitches with very similar F0s
      * switchCandidate: Switch pitch points as well as the corresponding masks between two pitch contours
      * convCont: Refine pitch estimates and produce pitch contours
      * reEstimatePitch: Re-estimate pitch points of a contour as well as the corresponding masks using maskToPitchMAP
      * checkContour: Check the pitch and mask continuity of a pitch contour
      * developeContour: Re-estimate non-continuous pitch points based on neighboring continuous pitch points
* mScaleInten.cpp and segmentation.cpp (and mScaleInten.h and segment.h): Onset/offset-based segmentation
  + See the paper “[Auditory segmentation based on onset and offset analysis](http://www.cse.ohio-state.edu/~dwang/papers/Hu-Wang.taslp07.pdf)” in IEEE Trans. Audio, Speech, and Lang. Process., vol. 15, pp. 396-405, 2007, by Hu G. and Wang D.L. for details
* common.h, tool.h, and tool.cpp: Utility files
* filter.h and filter.cpp: Low-pass/band-pass filters

**More details on training the MLPs**

The following is based on personal communication with Guoning Hu.

* Training of the single-unit based MLP (Sect. III-A)
  1. Training data
     1. 4620 sentences from the training part of the TIMIT database are divided into 2 equal parts
     2. Part one and part two are mixed one by one at 0 dB randomly, which yields 2310 mixtures. A couple of rules for mixing sentences:
        1. Each sentence was used once strictly
        2. 2 sentences from a same speaker were never mixed together
     3. Each sentence of part one was randomly mixed with one of the 100 intrusions (<http://www.cse.ohio-state.edu/pnl/corpus/HuCorpus.html>) at 0 dB, which yields another 2310 mixtures
     4. The training is performed on the above 4620 mixtures
  2. Training steps
     1. MATLAB neural-network toolbox was used for training
     2. A channel (ch) in the low frequency range is first picked (I cannot recall which channel, maybe the first one) and the MLP for this channel is trained with a random initial value. Let model(ch) represent the resulting MLP for this channel
     3. To speed up training, model(ch) is used as the initial value for channel ch+1. Then after training channel ch+1, model(ch+1) is used as the initial value to train channel ch+2, and so on. Channels ch-1, ch-2, ..., and 1 are trained similarly
     4. Step b and c were repeated several times. Each time it generated a set of 64 MLPs. The set that gave the best performance on testing data (Sect. IV-A) was selected
* Training of the multiple-unit based MLP (Sect. III-C)
  1. Training data: The same data as above
  2. Training steps
     1. MATLAB neural-network toolbox was used for training
     2. For each mixture, a T-F soft mask is first generated using the single-unit based MLP
     3. Given a soft mask, for each T-F unit *uc,m*, the probabilities of the surrounding T-F units are used as inputs to train an MLP to label *uc,m*. Ideal binary masks provide the desired outputs. We note that in the 64-channel version the frequency neighborhood size is reduced to 4 since only 64 gammatone filters are used. The time neighborhood size remains to be 2