```
1
2
3       typedef double XFLOAT
4       typedef double OTA_FLOAT
5
6   using  std
7
8
9   {
10
11  typedef std::vector<XFLOAT>FVector
12
13  /*****************************************************************************/
14
15  static XFLOAT const TA_OVERLAPFAC     = (XFLOAT)0.5
16  static int   const TA_WINLEN_SIGNAL   =
17      213 * TA_SAMPLING_RATE / 8000
18  static int   const TA_MIN_SEGLEN      = 512
19  static int   const TA_SHIFT_TOLERANCE =
20      round(0.005f * TA_SAMPLING_RATE)
21  static XFLOAT const TA_INFINITE_CORR  = (XFLOAT)100.0
22
23   SQFUNCS_POLQA_INTERNAL
24  {
25  /*****************************************************************************/
26
27  SQTA_ResampResult::SQTA_ResampResult ()
28  {
29      type = kRESNone
30      fMeanResamplingFac = 1.0f
31      fResamplingFactors = NULL
32      iNumSentences = 0
33      iNumTA = 0
34  }
35
36  SQTA_ResampResult::SQTA_ResampResult (SQTA_ResampResult const *toCopy)
37  {
38      type = toCopy->type
39      fMeanResamplingFac = toCopy->fMeanResamplingFac
40      iNumSentences = toCopy->iNumSentences
41      iNumTA = toCopy->iNumTA
42      if (iNumSentences)
43      {
44          fResamplingFactors = (XFLOAT*)matMalloc(iNumSentences * sizeof(XFLOAT))
45          for (int i = 0  i < iNumSentences  i++)
46          {
47              fResamplingFactors[i] = toCopy->fResamplingFactors[i]
48          }
49      }
50      else
51      {
52          fResamplingFactors = NULL
53      }
54  }
55
56  SQTA_ResampResult::~SQTA_ResampResult ()
57  {
58      if(fResamplingFactors)
59          matFree(fResamplingFactors)
60  }
61
62  void SQTA_ResampResult::assign (SQTA_ResampResult const *toCopy)
63  {
64      type = toCopy->type
65      fMeanResamplingFac = toCopy->fMeanResamplingFac
66      iNumSentences = toCopy->iNumSentences
67      matFree(fResamplingFactors)
68      if (iNumSentences)
```

```
    {
        fResamplingFactors = (XFLOAT*)matMalloc(iNumSentences * sizeof(XFLOAT))
        for (int i = 0  i < iNumSentences  i++)
        {
            fResamplingFactors[i] = toCopy->fResamplingFactors[i]
        }
    }
    else
    {
        fResamplingFactors = NULL
    }
}

/*****************************************************************************/

SQTimeAlignment::SQTimeAlignment (SQTimeAlignment const &inputTA, bool copySignals, MAT_HANDLE
inMatHandle, FILE* pLogFile)
:   mRef                (NULL),
    mDeg                (NULL),
    mSNRDeg             ((XFLOAT)40.0),
    mMatchQuality       (0.0),
    mExtremeMatchFound  (false),
    mMinDelay           (0),
    mMaxDelay           (0),
    mRefLen             (-1),
    mDegLen             (-1),
    mTargetLen          (-1),
    mTargetRate         (-1),
    mTargetBitRes       (-1),
    mActSpeechThr       ((XFLOAT)-45.0),
    mNumSentences       (-1),
    mCrudeDelay         (0),
    mCurResolution      (-1),
    mCurSegmentRate     (-1),
    mSegments           (NULL),
    mMergedSegments     (NULL),
    mUnusedDegSegments  (NULL),
    mMaxSigLenBuff      (NULL),
    matHandle           (inMatHandle),
    mpLogFile           (pLogFile)
{
    OPTTRY
    {
        mSNRDeg             = inputTA.SNRDeg()
        mMatchQuality       = inputTA.MatchQuality()
        mExtremeMatchFound  = inputTA.ExtremeMatchFound()
        mMinDelay           = inputTA.MinDelay()
        mMaxDelay           = inputTA.MaxDelay()
        mTargetLen          = inputTA.TargetLen()
        mTargetRate         = inputTA.SamplingRate()
        mTargetBitRes       = inputTA.TargetBitRes()
        mActSpeechThr       = inputTA.ActiveSpeechThr()
        mCrudeDelay         = inputTA.CrudeDelay()
        mCurResolution      = inputTA.CurResolution()
        mRefLen             = inputTA.RefLen()
        mDegLen             = inputTA.DegLen()
        mCurSegmentRate     = inputTA.CurSegmentRate()
        mNumSentences       = inputTA.NumSentences()

        if (copySignals)
        {
            mRef = new SQSignal (*inputTA.refSignal(), inputTA.refSignal()->SamplingFreq(),
inputTA.refSignal()->BitResolution())
            mDeg = new SQSignal (*inputTA.degSignal(), inputTA.degSignal()->SamplingFreq(),
inputTA.degSignal()->BitResolution())
            mRef->SetNumPause (inputTA.refSignal()->NumPause())
            mRef->SetPauseCenter (inputTA.refSignal()->PauseCenter(),
inputTA.refSignal()->SamplingFreq())
```

```
133             mDeg->SetNumPause (inputTA.degSignal()->NumPause())
134             mDeg->SetPauseCenter (inputTA.degSignal()->PauseCenter(),
    inputTA.degSignal()->SamplingFreq())
135         }
136
137         mSegments = new TA_SegList()
138         mSegments->assign (inputTA.Segments()->begin(), inputTA.Segments()->end())
139
140         mMergedSegments = new TA_SegList()
141         mMergedSegments->assign (inputTA.MergedSegments()->begin(), inputTA.MergedSegments()->end())
142
143         mUnusedDegSegments = new TA_SegList()
144         mUnusedDegSegments->assign (inputTA.UnusedDegSegments()->begin(),
    inputTA.UnusedDegSegments()->end())
145
146     }
147     OPTCATCH((string errorMsg))
148     {
149         delete mRef
150         delete mDeg
151         delete mSegments
152         delete mMergedSegments
153         delete mUnusedDegSegments
154         OPTTHROW(( string("ERROR in SQTimeAlignment::SQTimeAlignment: " + errorMsg + "\n")))
155     }
156     OPTCATCH( (...))
157     {
158         delete mRef
159         delete mDeg
160         delete mSegments
161         delete mMergedSegments
162         delete mUnusedDegSegments
163         OPTTHROW ((string("Unspecified error in SQTimeAlignment::SQTimeAlignment.")))
164     }
165 }
166 SQTimeAlignment::SQTimeAlignment (SQSignal const &sigRef, SQSignal const &sigDeg,
167                                   XFLOAT * const maxSigLenBuff, XFLOAT degResampFac, MAT_HANDLE
    inMatHandle, FILE* pLogFile)
168 :   mRef                (NULL),
169     mDeg                (NULL),
170     mSNRDeg             ((XFLOAT)40.0),
171     mMatchQuality       (0.0),
172     mExtremeMatchFound  (false),
173     mMinDelay           (0),
174     mMaxDelay           (0),
175     mRefLen             (-1),
176     mDegLen             (-1),
177     mTargetLen          (-1),
178     mTargetRate         (-1),
179     mTargetBitRes       (-1),
180     mActSpeechThr       ((XFLOAT)-45.0f),
181     mNumSentences       (-1),
182     mCrudeDelay         (0),
183     mCurResolution      (-1),
184     mCurSegmentRate     (-1),
185     mSegments           (NULL),
186     mMergedSegments     (NULL),
187     mUnusedDegSegments  (NULL),
188     mMaxSigLenBuff      (maxSigLenBuff),
189     matHandle           (inMatHandle),
190     mpLogFile           (pLogFile)
191 {
192     OPTTRY
193     {
194         if (sigRef.Data() == NULL || sigDeg.Data() == NULL ||
195             (sigRef.End()-sigRef.Start()) / sigRef.SamplingFreq() < MIN_SPEECH_DURATION ||
196             (sigDeg.End()-sigDeg.Start()) / sigDeg.SamplingFreq() < MIN_SPEECH_DURATION ||
197             sigRef.Start() < 0 || sigDeg.Start() < 0 ||
```

```
198             sigRef.BitResolution() <= 2 || sigDeg.BitResolution() <= 2)
199             OPTTHROW (string("Input signals inexistent / invalid / activity too short."))
200         if (!sigRef.Preprocessed() || !sigDeg.Preprocessed())
201             OPTTHROW (string("Input signals must undergo preprocessing prior to time alignment."))
202         if (sigRef.SamplingFreq() != sigDeg.SamplingFreq())
203             OPTTHROW (string("Input signals must have same Fs before time alignment."))
204
205         mTargetLen    = sigRef.NrOfSamples()
206         mTargetRate   = sigRef.SamplingFreq()
207         mTargetBitRes = sigRef.BitResolution()
208
209         //Preprocess ref and deg signal
210
211         ScaleFilterAndResample(sigRef, mRefLen, mRef)
212
213         mRef->SlidingWinMeanRemoval(TA_WINLEN_SIGNAL)
214
215         mNumSentences  = sigRef.NumPause() + 1
216
217         ScaleFilterAndResample(sigDeg, mDegLen, mDeg, degResampFac)
218
219         mDeg->SlidingWinMeanRemoval(TA_WINLEN_SIGNAL)
220
221         //Compute common threshold for active speech
222         CalcActSpeechThr(sigDeg)
223
224         //Estimate overall delay
225         CalcCrudeDelay()
226
227         //Refine overall delay estimation
228         RefineCrudeDelay()
229
230         //Compute segment-wise delay.
231         FineDelay()
232
233         //Scale crude delay to mTargetRate, seg lists already scaled in FineDelay().
234         mCrudeDelay *= mTargetRate/TA_SAMPLING_RATE
235
236         //Compute min and max delay in the entire signal
237         CalcDelaySpread()
238     }
239     OPTCATCH( (string errorMsg))
240     {
241         delete mRef
242         delete mDeg
243         delete mSegments
244         delete mMergedSegments
245         delete mUnusedDegSegments
246         OPTTHROW(( string("ERROR in SQTimeAlignment::SQTimeAlignment: " + errorMsg + "\n")))
247     }
248     OPTCATCH( (...))
249     {
250         delete mRef
251         delete mDeg
252         delete mSegments
253         delete mMergedSegments
254         delete mUnusedDegSegments
255         OPTTHROW(( string("Unspecified error in SQTimeAlignment::SQTimeAlignment.")))
256     }
257 }
258
259 SQTimeAlignment::~SQTimeAlignment()
260 {
261     delete mRef
262     delete mDeg
263     delete mSegments
264     delete mMergedSegments
265     delete mUnusedDegSegments
```

```
266
267     mRef = mDeg = NULL
268     mSegments = mMergedSegments = mUnusedDegSegments = NULL
269 }
270
271 XFLOAT SQTimeAlignment::SNRDeg() const
272 {
273     return mSNRDeg
274 }
275
276 XFLOAT SQTimeAlignment::MatchQuality() const
277 {
278     return mMatchQuality
279 }
280
281 bool SQTimeAlignment::ExtremeMatchFound() const
282 {
283     return mExtremeMatchFound
284 }
285
286 int SQTimeAlignment::MinDelay() const
287 {
288     return mMinDelay
289 }
290
291 int SQTimeAlignment::MaxDelay() const
292 {
293     return mMaxDelay
294 }
295
296 long SQTimeAlignment::RefLen() const
297 {
298     return mRefLen
299 }
300
301 long SQTimeAlignment::DegLen() const
302 {
303     return mDegLen
304 }
305
306 long SQTimeAlignment::TargetLen() const
307 {
308     return mTargetLen
309 }
310
311 int SQTimeAlignment::TargetBitRes() const
312 {
313     return mTargetBitRes
314 }
315
316 XFLOAT SQTimeAlignment::ActiveSpeechThr() const
317 {
318     return mActSpeechThr
319 }
320
321 long SQTimeAlignment::CurResolution() const
322 {
323     return mCurResolution
324 }
325
326 int SQTimeAlignment::CrudeDelay() const
327 {
328     return mCrudeDelay
329 }
330
331 int SQTimeAlignment::SamplingRate() const
332 {
333     return mTargetRate
```

```
}

short SQTimeAlignment::NumSentences() const
{
    return mNumSentences
}

int SQTimeAlignment::CurSegmentRate() const
{
    return mCurSegmentRate
}

TA_SegList const* SQTimeAlignment::Segments() const
{
    return mSegments
}

TA_SegList const* SQTimeAlignment::MergedSegments() const
{
    return mMergedSegments
}

TA_SegList const* SQTimeAlignment::UnusedDegSegments() const
{
    return mUnusedDegSegments
}

SQSignal const* SQTimeAlignment::refSignal() const
{
    return mRef
}

SQSignal const* SQTimeAlignment::degSignal() const
{
    return mDeg
}

/*****************************************************************************/

void SQTimeAlignment::ScaleFilterAndResample(SQSignal const &Input,
                                             long &lResLen,
                                             SQSignal* &Output,
                                             XFLOAT resampFac)
{
    int      iOrigFs        = Input.SamplingFreq()
    int      iGCD           = gcd(iOrigFs, TA_SAMPLING_RATE)
    XFLOAT   *fDelayLine    = NULL
    XFLOAT   *fPaddedInput  = NULL
    XFLOAT   *fResampledData= NULL
    XFLOAT   *dFiltCoeffs   = NULL
    int      iTapsLen       = 0

    OPTTRY
    {
        if (iGCD <= 0 || Input.Data() == NULL || Input.NrOfSamples() <= 1 ||
            iOrigFs < TA_SAMPLING_RATE || iOrigFs < 1 || resampFac <= 0.0f)
            OPTTHROW (string("Invalid input arguments / signal data."))

        int     iUpFac      = TA_SAMPLING_RATE / iGCD
        int     iDownFac    = iOrigFs / iGCD

        if (iUpFac != 1) //Downsampling only, upsampling not supported
        {
            stringstream errorStream
            errorStream << "Signal rate must be an integer multiple of " << TA_SAMPLING_RATE << "."
            OPTTHROW( errorStream.str())
        }
        if (iOrigFs != 32000) //Cannot use precomputed filter taps, compute new ones
```

```
402         {
403
404             XFLOAT dLowFrq  =  700.0 / iOrigFs
405             XFLOAT dHighFrq = 3000.0 / iOrigFs
406
407             iTapsLen        = (((iDownFac*64) < (512)) ? (iDownFac*64) : (512))
408
409             if(mpLogFile)
410             {
411                 fprintf(mpLogFile, "Bandpass Taps: nrTaps %d\n", iTapsLen)
412             }
413             switch(iOrigFs)
414             {
415                 case 8000:
416                     dFiltCoeffs = sqBPTaps8k
417                     break
418
419                 case 16000:
420                     dFiltCoeffs = sqBPTaps16k
421                     break
422
423                 case 48000:
424                     dFiltCoeffs = sqBPTaps48k
425                     break
426
427                 default:
428
429                     if(mpLogFile)
430                     {
431                         fprintf(mpLogFile, "Dynamic Generation of Taps\n")
432                     }
433                     dFiltCoeffs     = (XFLOAT*)matMalloc(iTapsLen * sizeof(XFLOAT))
434                     if (matGenBandPassCoefficients(dLowFrq, dHighFrq, dFiltCoeffs, iTapsLen,
    MAT_WinBlackman) != 0)
435                         OPTTHROW( string("Bandpass generation in mathlib failed."))
436                 break
437             }
438
439         }
440         else //Use precomputed filter taps
441         {
442             iTapsLen    = TA_INITS_iBPCoeffLen_32k
443             dFiltCoeffs = TA_INITS_dBPCoeffs_32k
444         }
445
446         long lPaddedLen = Input.NrOfSamples() + 2*iTapsLen
447         int  iNumIters  = lPaddedLen / iDownFac
448         XFLOAT fMaxAmplitude = (((MAX_AMP_32BIT) < (pow((XFLOAT)2.0, Input.BitResolution() - 1))) ?
    (MAX_AMP_32BIT) : (pow((XFLOAT)2.0, Input.BitResolution() - 1)))
449         XFLOAT fScaleFac = pow((XFLOAT)2.0, STD_BIT_RESOLUTION - 1) / fMaxAmplitude
450
451         fPaddedInput    = (XFLOAT*)matMalloc(lPaddedLen * sizeof(XFLOAT))
452         fResampledData  = (XFLOAT*)matMalloc(iNumIters * iUpFac * sizeof(XFLOAT))
453
454         matbZero(fPaddedInput, lPaddedLen)
455         vsmul(Input.Data(), fScaleFac, fPaddedInput+iTapsLen, Input.NrOfSamples())
456
457         matRunFIRMRFilter(fPaddedInput, fResampledData, iNumIters, dFiltCoeffs, iTapsLen, iUpFac,
    iDownFac)
458
459         if (Output != NULL)
460             delete Output
461         lResLen     = Input.NrOfSamples() / iDownFac
462         int iOffSet = (((iNumIters-lResLen) < (iTapsLen/iDownFac + iTapsLen/2/iDownFac + 1)) ?
    (iNumIters-lResLen) : (iTapsLen/iDownFac + iTapsLen/2/iDownFac + 1))
463         Output      = new SQSignal(fResampledData + iOffSet, lResLen,
464                                    TA_SAMPLING_RATE, STD_BIT_RESOLUTION)
465
```

```
466         if (dFiltCoeffs &&
467             dFiltCoeffs != TA_INITS_dBPCoeffs_32k &&
468             dFiltCoeffs != sqBPTaps8k &&
469             dFiltCoeffs != sqBPTaps16k &&
470             dFiltCoeffs != sqBPTaps48k)
471             matFree(dFiltCoeffs)
472         dFiltCoeffs = NULL
473 
474         if(fPaddedInput)
475             matFree(fPaddedInput)
476         if(fResampledData)
477             matFree(fResampledData)
478         fPaddedInput = fResampledData = NULL
479     }
480     OPTCATCH((...))
481     {
482         matFree(fPaddedInput)
483         matFree(fResampledData)
484         fPaddedInput = fResampledData = NULL
485         if(dFiltCoeffs && dFiltCoeffs != TA_INITS_dBPCoeffs_32k && dFiltCoeffs != sqBPTaps8k &&
    dFiltCoeffs != sqBPTaps16k && dFiltCoeffs != sqBPTaps48k)
486         {
487             matFree(dFiltCoeffs)
488             dFiltCoeffs = NULL
489         }
490 
491         OPTTHROW (string("ERROR in SQTimeAlignment::ScaleFilterAndResample using Mathlib\n"))
492     }
493 }
494 
495 void SQTimeAlignment::CalcActSpeechThr(SQSignal const &originalDegSig)
496 {
497     if (mRef == NULL || mDeg == NULL || mRef->Data() == NULL || mDeg->Data() == NULL)
498         OPTTHROW (string("SQTimeAlignment::CalcActSpeechThr failed.\n"))
499 
500     XFLOAT const FRAMELEN = (XFLOAT)0.032
501     mRef->CalcEnvelope(FRAMELEN, TA_OVERLAPFAC, MIN_LEVEL_DB, mMaxSigLenBuff)
502     mDeg->CalcEnvelope(FRAMELEN, TA_OVERLAPFAC, MIN_LEVEL_DB, mMaxSigLenBuff)
503 
504     //Estimate the noise level
505     mRef->CalcASLandNoiseLevel(FRAMELEN, mMaxSigLenBuff)
506     mDeg->CalcASLandNoiseLevel(FRAMELEN, mMaxSigLenBuff)
507 
508     //The bandpass in the constructor may leave too little speech in case of strong BG noise.
509     if (mDeg->CurrentASL() - mDeg->CurrentNoiseLevel() < 3.0f)
510     {
511         delete mDeg
512         mDeg = new SQSignal(originalDegSig)
513         mDeg->Resample(TA_SAMPLING_RATE)
514         mDegLen = mDeg->NrOfSamples()
515         mDeg->SlidingWinMeanRemoval(TA_WINLEN_SIGNAL)
516 
517         mDeg->CalcEnvelope(FRAMELEN, TA_OVERLAPFAC, MIN_LEVEL_DB, mMaxSigLenBuff)
518         mDeg->CalcASLandNoiseLevel(FRAMELEN, mMaxSigLenBuff)
519     }
520 
521     mSNRDeg = mDeg->CurrentASL() - mDeg->CurrentNoiseLevel()
522 
523     mRef->LevelAlign(REF_ASL_LEVEL)
524     mDeg->LevelAlign(REF_ASL_LEVEL)
525 
526     //Set the threshold for active speech, taking the max between ref and deg.
527     mActSpeechThr = ((((mRef->CurrentASL() + 3 * mRef->CurrentNoiseLevel()) / 4.0f) >
    ((mDeg->CurrentASL() + 3 * mDeg->CurrentNoiseLevel()) / 4.0f)) ? ((mRef->CurrentASL() + 3 *
    mRef->CurrentNoiseLevel()) / 4.0f) : ((mDeg->CurrentASL() + 3 * mDeg->CurrentNoiseLevel()) /
    4.0f))
528     mActSpeechThr = (((mRef->CurrentASL()-3.0f) < (mActSpeechThr)) ? (mRef->CurrentASL()-3.0f) :
    (mActSpeechThr))
```

```
529     return
530 }
531
532 void SQTimeAlignment::CalcCrudeDelay()
533 {
534     XFLOAT const FCORRTHRESH      = (XFLOAT)0.42
535     XFLOAT      fMaxCorr
536     int         iDelay
537     XFLOAT      fTA_framelen  = (XFLOAT)0.18
538     XFLOAT const    FMINFRAMESTEP = (XFLOAT)0.008
539     int         iNrOfShifts   =
540         round((2*(((mRef->NrOfSamples()) > (mDeg->NrOfSamples())) ? (mRef->NrOfSamples()) :
    (mDeg->NrOfSamples())) - mRef->NrOfSamples()/2)
541                 /(fTA_framelen*TA_SAMPLING_RATE*TA_OVERLAPFAC)) | 0x1
542     int const   ISTEPSIZE     = 3
543     XFLOAT const WINLEN_ENV   = (XFLOAT)0.45
544
545     XFLOAT *fRefEnvNorm = NULL, *fDegEnvNorm = NULL, *fFBStmp1 = NULL, *fFBStmp2 = NULL, *fSWPNtmp =
    NULL
546
547     OPTTRY
548     {
549         if (mRef == NULL || mDeg == NULL || mActSpeechThr >= 0.0f ||
550             mRef->Data() == NULL || mDeg->Data() == NULL)
551             OPTTHROW( string("Invalid or NULL signals or parameters."))
552
553         int maxNumberOfFrames = (int)ceil((((mRef->NrOfSamples()) > (mDeg->NrOfSamples())) ?
    (mRef->NrOfSamples()) : (mDeg->NrOfSamples())) /
554             (FMINFRAMESTEP*TA_SAMPLING_RATE))
555         if ((fFBStmp1 = (XFLOAT*)matMalloc(iNrOfShifts*sizeof(XFLOAT))) == NULL ||
556             (fFBStmp2 = (XFLOAT*)matMalloc(iNrOfShifts*sizeof(XFLOAT))) == NULL ||
557             (fSWPNtmp = (XFLOAT*)matMalloc(maxNumberOfFrames*sizeof(XFLOAT))) == NULL)
558             OPTTHROW((string("ippsMalloc failed.")))
559
560         mCrudeDelay = 0
561
562         //Find start and end of signal activity in Ref.
563         mRef->FindActBoundaries(mActSpeechThr, 5, FMINFRAMESTEP, 0, false)
564
565         for (  fTA_framelen * TA_OVERLAPFAC > FMINFRAMESTEP  fTA_framelen /= ISTEPSIZE)
566         {
567             XFLOAT curEnvStepSize = fTA_framelen * TA_OVERLAPFAC * TA_SAMPLING_RATE
568             int   refOffset      = (int)(mRef->Start() / curEnvStepSize)
569
570             //Threshold the signal envelopes using the active speech threshold
571             mRef->CalcEnvelope(fTA_framelen, TA_OVERLAPFAC, mActSpeechThr, mMaxSigLenBuff)
572             mDeg->CalcEnvelope(fTA_framelen, TA_OVERLAPFAC, mActSpeechThr, mMaxSigLenBuff)
573             vsadd(mRef->Env(), -mActSpeechThr, mRef->Env(), mRef->NrOfFrames())
574             vsadd(mDeg->Env(), -mActSpeechThr, mDeg->Env(), mDeg->NrOfFrames())
575
576             //Apply sliding window power normalization to a copy of the envelopes
577             fRefEnvNorm = (XFLOAT*)matMalloc(mRef->NrOfFrames()*sizeof(XFLOAT))
578             fDegEnvNorm = (XFLOAT*)matMalloc(mDeg->NrOfFrames()*sizeof(XFLOAT))
579             if (fRefEnvNorm == NULL || fDegEnvNorm == NULL)
580                 OPTTHROW( string("matMalloc failed."))
581
582             XFLOAT meanLevdB
583             rmvesq(mRef->Env(), &meanLevdB, mRef->NrOfFrames())
584             meanLevdB = 20.0f * log10(meanLevdB+1e-16f) / mActSpeechThr
585
586             SlidingWinPowNorm(mRef->Env(), fRefEnvNorm, mRef->NrOfFrames(), MIN_LEVEL_DB,
587                 round(WINLEN_ENV / (fTA_framelen*TA_OVERLAPFAC)), -mActSpeechThr, meanLevdB, false,
    0, fSWPNtmp)
588             SlidingWinPowNorm(mDeg->Env(), fDegEnvNorm, mDeg->NrOfFrames(), MIN_LEVEL_DB,
589                 round(WINLEN_ENV / (fTA_framelen*TA_OVERLAPFAC)), -mActSpeechThr, meanLevdB, false,
    0, fSWPNtmp)
590
591             //Compute cross-correlation
```

```
592             FindBestShift(fRefEnvNorm+refOffset, mRef->NrOfFrames() - refOffset,
593                           fDegEnvNorm, mDeg->NrOfFrames(),
594                           iNrOfShifts, refOffset + round(mCrudeDelay / curEnvStepSize),
595                           fMaxCorr, iDelay, false, false, fFBStmp1, fFBStmp2)
596 
597             if (fRefEnvNorm != NULL) matFree(fRefEnvNorm)
598             if (fDegEnvNorm != NULL) matFree(fDegEnvNorm)
599 
600             fRefEnvNorm = fDegEnvNorm = NULL
601 
602             if (fMaxCorr >= FCORRTHRESH)
603                 mCrudeDelay += round(iDelay * curEnvStepSize)
604             else
605                 break
606 
607             iNrOfShifts = ISTEPSIZE+2
608         }
609 
610         if (fFBStmp1 != NULL) matFree(fFBStmp1)
611         if (fFBStmp2 != NULL) matFree(fFBStmp2)
612         if (fSWPNtmp != NULL) matFree(fSWPNtmp)
613 
614         fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
615     }
616     OPTCATCH((string errorMsg))
617     {
618         if (fRefEnvNorm != NULL) matFree(fRefEnvNorm)
619         if (fDegEnvNorm != NULL) matFree(fDegEnvNorm)
620         if (fFBStmp1    != NULL) matFree(fFBStmp1)
621         if (fFBStmp2    != NULL) matFree(fFBStmp2)
622         if (fSWPNtmp    != NULL) matFree(fSWPNtmp)
623 
624         fRefEnvNorm = fDegEnvNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
625         mCrudeDelay = 0
626         OPTTHROW( string("ERROR in SQTimeAlignment::CalcCrudeDelay: " + errorMsg + "\n"))
627     }
628 
629     //Store last resolution for RefineCrudeDelay()
630     fTA_framelen *= ISTEPSIZE
631     mCurResolution = 2 * round(fTA_framelen * TA_OVERLAPFAC * TA_SAMPLING_RATE)
632 }
633 
634 void SQTimeAlignment::RefineCrudeDelay()
635 {
636     XFLOAT *fRefNorm = NULL, *fDegNorm = NULL,
637            *fFBStmp1 = NULL, *fFBStmp2 = NULL,
638            *fSWPNtmp = NULL
639 
640     OPTTRY
641     {
642         if (mCurResolution < 3)
643             return
644         if (mRef == NULL || mDeg == NULL || mRef->Data() == NULL || mDeg->Data() == NULL ||
645             mRef->NrOfSamples() < 2 || mRef->NrOfSamples() <= mRef->Start() ||
646             mDeg->NrOfSamples() < 2)
647             OPTTHROW( string("Invalid input parameters."))
648 
649         if ((mCurResolution | 1) != mCurResolution)
650             mCurResolution++
651 
652         //Apply overall delay computed so far
653         int  iDegOffset = (((0) > (mRef->Start() + mCrudeDelay)) ? (0) : (mRef->Start() +
    mCrudeDelay))
654         int  iRefOffset = mRef->Start() + mCrudeDelay < 0 ?
655             -mCrudeDelay : mRef->Start()
656         long lRefNorm = mRef->NrOfSamples() - iRefOffset
657         long lDegNorm = mDeg->NrOfSamples() - iDegOffset
658         lRefNorm = lDegNorm = (((lRefNorm) < (lDegNorm)) ? (lRefNorm) : (lDegNorm))
```

```
659         if (lDegNorm <= 10 || lRefNorm <= 10)
660             OPTTHROW( string("Length of aligned Ref and/or Deg too short."))
661 
662         //Normalize a copy of both signals
663         if ((fRefNorm = (XFLOAT*)matMalloc(lRefNorm*sizeof(XFLOAT))) == NULL ||
664             (fDegNorm = (XFLOAT*)matMalloc(lDegNorm*sizeof(XFLOAT))) == NULL ||
665             (fFBStmp1 = (XFLOAT*)matMalloc(mCurResolution*sizeof(XFLOAT))) == NULL ||
666             (fFBStmp2 = (XFLOAT*)matMalloc(mCurResolution*sizeof(XFLOAT))) == NULL ||
667             (fSWPNtmp = (XFLOAT*)matMalloc((((lRefNorm) > (lDegNorm)) ? (lRefNorm) : (lDegNorm)) *
    sizeof(XFLOAT))) == NULL)
668             OPTTHROW((string("ippsMalloc failed.")))
669 
670         SlidingWinPowNorm(mRef->Data()+iRefOffset, fRefNorm, lRefNorm, mActSpeechThr,
671             TA_WINLEN_SIGNAL, pow(2.0f, mRef->BitResolution()-1), REF_ASL_LEVEL, true, 0, fSWPNtmp)
672         SlidingWinPowNorm(mDeg->Data()+iDegOffset, fDegNorm, lDegNorm, mActSpeechThr,
673             TA_WINLEN_SIGNAL, pow(2.0f, mDeg->BitResolution()-1), REF_ASL_LEVEL, true, 0, fSWPNtmp)
674 
675         //Find the shifting with the maximum correlation
676         XFLOAT fMaxCorr = (XFLOAT)0.0
677         int iBestShift = 0
678         FindBestShift(fRefNorm, lRefNorm,
679                       fDegNorm, lDegNorm,
680                       mCurResolution, 0,
681                       fMaxCorr, iBestShift,
682                       true, false, fFBStmp1, fFBStmp2)
683 
684         if (fMaxCorr > (XFLOAT)0.7)
685         {
686             mCrudeDelay += iBestShift
687             mCurResolution = 1
688         }
689     }
690     OPTCATCH((string errorMsg))
691     {
692         if (fRefNorm != NULL) matFree(fRefNorm)
693         if (fDegNorm != NULL) matFree(fDegNorm)
694         if (fFBStmp1 != NULL) matFree(fFBStmp1)
695         if (fFBStmp2 != NULL) matFree(fFBStmp2)
696         if (fSWPNtmp != NULL) matFree(fSWPNtmp)
697 
698         fRefNorm = fDegNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
699         OPTTHROW( string("Error in SQTimeAlignment::RefineCrudeDelay: " + errorMsg + "\n"))
700     }
701     if (fRefNorm != NULL) matFree(fRefNorm)
702     if (fDegNorm != NULL) matFree(fDegNorm)
703     if (fFBStmp1 != NULL) matFree(fFBStmp1)
704     if (fFBStmp2 != NULL) matFree(fFBStmp2)
705     if (fSWPNtmp != NULL) matFree(fSWPNtmp)
706 
707     fRefNorm = fDegNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
708     return
709 }
710 
711 void SQTimeAlignment::FineDelay()
712 {
713     XFLOAT *fRefNorm = NULL, *fDegNorm = NULL, *fRefEnv = NULL, *fFBStmp1 = NULL, *fFBStmp2 = NULL,
    *fSWPNtmp = NULL
714 
715     OPTTRY
716     {
717         if (mRef == NULL || mDeg == NULL || mRef->Data() == NULL || mDeg->Data() == NULL ||
718             mRef->NrOfSamples() < 2 || mDeg->NrOfSamples() < 2 || mActSpeechThr > 0)
719             OPTTHROW( string("Invalid input parameters."))
720 
721         //Local constants definitions
722         int   const FD_MAX_SEGLEN      =
723             (((TA_MIN_SEGLEN+1) > (round(1.5 * TA_SAMPLING_RATE))) ? (TA_MIN_SEGLEN+1) : (round(1.5
    * TA_SAMPLING_RATE)))
```

```
724         int   const FD_MIN_SPANLEN     = 425
725         XFLOAT const FD_MIN_SPANDEC     = (XFLOAT)0.15
726         XFLOAT const FD_FRAMELEN        = (XFLOAT)0.016
727         XFLOAT const FD_OVERLAPFAC      = (XFLOAT)0.75
728         int   const FD_FRAMESTEP       = round(TA_SAMPLING_RATE * FD_FRAMELEN * FD_OVERLAPFAC)
729         int   const FD_MAXSHIFT        =
730         XFLOAT const FD_CORRTHRESH_SPAN = (XFLOAT)0.2
731         XFLOAT const FD_CORRTHRESH_LOUD = (XFLOAT)0.32
732         XFLOAT const FD_CORRTHRESH_QUIET= (XFLOAT)0.5
733         XFLOAT const FD_MIN_LEVEL       =
734             (XFLOAT)(((mActSpeechThr) > (-55.0)) ? (mActSpeechThr) : (-55.0))
735
736         if(mpLogFile)
737             fprintf(mpLogFile, "\nFineDelay()\n")
738
739         //Normalize a copy of both signals.
740         XFLOAT fRefTargetMeanLevel = mRef->SpeechActivity() > (XFLOAT)0.2 ?
741             REF_ASL_LEVEL + powTodB(mRef->SpeechActivity()) : REF_ASL_LEVEL
742         XFLOAT fDegTargetMeanLevel = mDeg->SpeechActivity() > (XFLOAT)0.2 ?
743             REF_ASL_LEVEL + powTodB(mDeg->SpeechActivity()) : REF_ASL_LEVEL
744         int lRefNorm = mRef->NrOfSamples()
745         int lDegNorm = mDeg->NrOfSamples()
746         if ((fRefNorm = (XFLOAT*)matMalloc(lRefNorm*sizeof(XFLOAT)))    == NULL ||
747             (fDegNorm = (XFLOAT*)matMalloc(lDegNorm*sizeof(XFLOAT)))    == NULL ||
748             (fFBStmp1 = (XFLOAT*)matMalloc(FD_MAXSHIFT*sizeof(XFLOAT))) == NULL ||
749             (fFBStmp2 = (XFLOAT*)matMalloc(FD_MAXSHIFT*sizeof(XFLOAT))) == NULL ||
750             (fSWPNtmp = (XFLOAT*)matMalloc((((lRefNorm) > (lDegNorm)) ? (lRefNorm) : (lDegNorm)) *
    sizeof(XFLOAT))) == NULL)
751             OPTTHROW((string("matMalloc failed.")))
752
753         SlidingWinPowNorm(mRef->Data(), fRefNorm, lRefNorm, mActSpeechThr, TA_WINLEN_SIGNAL,
754             pow((XFLOAT)2.0, mRef->BitResolution()-1), fRefTargetMeanLevel, false,
    round(70e-3*TA_SAMPLING_RATE), fSWPNtmp)
755         SlidingWinPowNorm(mDeg->Data(), fDegNorm, lDegNorm, mActSpeechThr, TA_WINLEN_SIGNAL,
756             pow((XFLOAT)2.0, mDeg->BitResolution()-1), fDegTargetMeanLevel, false,
    round(70e-3*TA_SAMPLING_RATE), fSWPNtmp)
757
758         mRef->CalcEnvelope(FD_FRAMELEN, FD_OVERLAPFAC, MIN_LEVEL_DB, mMaxSigLenBuff)
759         int lRefFrames = mRef->NrOfFrames()
760         if (lRefFrames < 3)
761             OPTTHROW( string("Reference envelope too short."))
762
763         //Create temp. envelope to find utterances in ref signal
764         //Utterances will be removed from this env. as they are matched.
765         fRefEnv = (XFLOAT*)matMalloc(lRefFrames * sizeof(XFLOAT))
766         vmov(mRef->Env(), fRefEnv, lRefFrames)
767         XFLOAT fUttThresh =
768
769         //Loop through all utterances in the reference, starting with the loudest one.
770         //Try to match each utt. in the ref. with one in the deg. signal.
771         delete mSegments
772         mSegments = new TA_SegList
773         ReserveVectorMemory(FD_MIN_SPANLEN)
774         mCurSegmentRate = mRef->SamplingFreq()
775         bool  doMatchLoudestUtt = true,
776               doIgnorePauses    = true,
777               silenceFoundRef   = false,
778               silenceFoundDeg   = false
779         int   uttStart, uttEnd
780         XFLOAT fMaxCorr  = (XFLOAT)0.0,
781               fSpanCorr = (XFLOAT)0.0,
782               fCorrThr  = FD_CORRTHRESH_LOUD
783         int   iBestShift, iGuessedShift,
784               iDegStart, iMaxDegLen,
785               uttLen, spanLen, spanStart,
786               gapSearchPos = 0
787
788         XFLOAT guessFac
```

```
789
790         int segCounter = 0
791         while (true)
792         {
793             //Pick the next reference segment to match, using the ref envelope
794             if (GetNextRefSegmentToMatch(uttStart, uttEnd, fUttThresh, FD_MIN_LEVEL,
795                                          doMatchLoudestUtt, doIgnorePauses, gapSearchPos,
796                                          fRefEnv, lRefFrames, FD_FRAMESTEP, lRefNorm)
797                 != 0)
798                 break  //No segment left to match, exit loop.
799
800             if (fUttThresh <= FD_MIN_LEVEL)
801                 fCorrThr = FD_CORRTHRESH_QUIET
802
803             //Check both the selected ref segment and the available deg signal
804             CheckCurrentRefSeg(fRefNorm, lRefNorm, uttStart, uttEnd,
805                                silenceFoundRef, FD_MAX_SEGLEN)
806
807             if (silenceFoundRef && doIgnorePauses && !doMatchLoudestUtt)
808             {
809                 gapSearchPos = uttEnd+1
810                 continue
811             }
812
813             CheckCurrentDegSeg(uttStart, fDegNorm, lDegNorm, !silenceFoundRef,
814                                silenceFoundDeg, iDegStart, iMaxDegLen)
815
816             if (iMaxDegLen <= 0)
817             {
818                 uttLen = uttEnd - uttStart + 1
819                 GuessBestShift(uttStart, iDegStart, iMaxDegLen, fDegNorm, lDegNorm,
820                                uttLen, iGuessedShift)
821                 InsertSegment(uttStart, iGuessedShift, uttLen,
822                               TA_SEG_MISSING, lDegNorm,
823                               doMatchLoudestUtt, fRefEnv, lRefFrames,
824                               FD_FRAMESTEP, MIN_LEVEL_DB)
825                 continue
826             }
827
828             //Constrain segment length based on available deg signal
829             uttLen    = (((uttEnd - uttStart + 1) < (iMaxDegLen)) ? (uttEnd - uttStart + 1) :
    (iMaxDegLen))
830             spanLen   = uttLen
831             spanStart = uttStart
832             fMaxCorr  = fSpanCorr = 0.0f
833
834             //Perform matching by cross-correlation
835             if (!silenceFoundRef &&
836                 !silenceFoundDeg &&
837                 uttLen >= TA_MIN_SEGLEN)
838             {
839                 GuessBestShift(uttStart, iDegStart, iMaxDegLen, fDegNorm, lDegNorm,
840
841                                uttLen, iGuessedShift, &guessFac)
842
843                 //Perform cross-correlation, and determine delay of current segment
844                 FindBestShift(fRefNorm+uttStart,  uttLen,
845                               fDegNorm+iDegStart, iMaxDegLen,
846                               (((uttLen+iMaxDegLen) < (FD_MAXSHIFT)) ? (uttLen+iMaxDegLen) :
    (FD_MAXSHIFT)),
847                               uttStart - iDegStart + iGuessedShift,
848                               fMaxCorr, iBestShift, true,
849                               mSegments->size() >= 1,
850                               fFBStmp1, fFBStmp2,
851
852                               guessFac)
853
854             iBestShift += iGuessedShift
```

```
855
856                 if (abs(iBestShift-iGuessedShift) <= 1 * TA_SAMPLING_RATE / 8000 &&
857                     mSegments->size() >= 2)
858                     iBestShift = iGuessedShift
859
860                 if (fMaxCorr >= 0.2f ||
861                     (fMaxCorr >= 0.1f && uttLen >= 1024) ||
862                     (mSegments->size() >= 2 &&
863                      fMaxCorr >  0.0f && uttLen >= 2667))
864
865                     fSpanCorr = SpanCorr(fRefNorm, fDegNorm, lDegNorm, iBestShift,
866                                          FD_CORRTHRESH_SPAN, fMaxCorr, spanStart,
867                                          spanLen, fFBStmp1, fFBStmp2, FD_MIN_SPANLEN)
868
869             }
870
871             //Write found delay to segments list
872             if (silenceFoundRef ||
873                 silenceFoundDeg ||
874                 uttLen  < TA_MIN_SEGLEN ||
875                 spanLen < FD_MIN_SPANLEN ||
876                 fSpanCorr < fCorrThr ||
877                 (doMatchLoudestUtt && (XFLOAT)spanLen /
878                  (((uttLen) < (TA_SAMPLING_RATE*0.375f)) ? (uttLen) : (TA_SAMPLING_RATE*0.375f))
879                   < FD_MIN_SPANDEC))
880             {
881                 if (!doMatchLoudestUtt)
882                 {
883                     TA_SEG_TYPE segType = silenceFoundRef ? TA_SEG_PAUSE : TA_SEG_GUESSED
884                     GuessBestShift(uttStart, iDegStart, iMaxDegLen, fDegNorm, lDegNorm,
885                                    uttLen, iGuessedShift)
886                     InsertSegment(uttStart, iGuessedShift, uttLen,
887                                   segType, lDegNorm,
888                                   doMatchLoudestUtt, fRefEnv, lRefFrames,
889                                   FD_FRAMESTEP, MIN_LEVEL_DB, (((fMaxCorr/FD_CORRTHRESH_QUIET) <
    (1.0f)) ? (fMaxCorr/FD_CORRTHRESH_QUIET) : (1.0f)))
890                 }
891                 else
892                     RemoveUttFromEnv(fRefEnv, lRefFrames, FD_FRAMESTEP, MIN_LEVEL_DB,
893                                      uttStart, uttEnd-uttStart+1)
894             }
895             else
896             {
897                 WriteMatchedSegment(spanStart, spanLen, iBestShift,
898                                     iDegStart, iMaxDegLen, lDegNorm,
899                                     doMatchLoudestUtt, fRefEnv, lRefFrames,
900                                     FD_FRAMESTEP, MIN_LEVEL_DB)
901             }
902
903             segCounter++
904         }
905
906         //Perform post-processings
907
908         MergeConsecutiveSegments()
909
910         mMatchQuality = CalcMatchQuality(mSegments)
911
912         if (mMatchQuality < (XFLOAT)0.75)
913         {
914
915         FixExtremeMatches(fRefNorm, fDegNorm, lRefNorm, lDegNorm)
916
917         FixIncorrectMatches()
918
919         }
920
921         FineDelayPostProc(fDegNorm, lDegNorm, lRefNorm)
```

```
922
923         matFree(fRefEnv)
924         fRefEnv = NULL
925
926         if (fRefNorm != NULL) matFree(fRefNorm)
927         if (fDegNorm != NULL) matFree(fDegNorm)
928         if (fFBStmp1 != NULL) matFree(fFBStmp1)
929         if (fFBStmp2 != NULL) matFree(fFBStmp2)
930         if (fSWPNtmp != NULL) matFree(fSWPNtmp)
931
932         fRefNorm = fDegNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
933     }
934     OPTCATCH((string errorMsg))
935     {
936         matFree(fRefEnv)
937         fRefEnv = NULL
938
939         if (fRefNorm != NULL) matFree(fRefNorm)
940         if (fDegNorm != NULL) matFree(fDegNorm)
941         if (fFBStmp1 != NULL) matFree(fFBStmp1)
942         if (fFBStmp2 != NULL) matFree(fFBStmp2)
943         if (fSWPNtmp != NULL) matFree(fSWPNtmp)
944
945         fRefNorm = fDegNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
946         OPTTHROW( string("ERROR in FineDelay: " + errorMsg + "\n"))
947     }
948     OPTCATCH((...))
949     {
950         matFree(fRefEnv)
951         fRefEnv = NULL
952
953         if (fRefNorm != NULL) matFree(fRefNorm)
954         if (fDegNorm != NULL) matFree(fDegNorm)
955         if (fFBStmp1 != NULL) matFree(fFBStmp1)
956         if (fFBStmp2 != NULL) matFree(fFBStmp2)
957         if (fSWPNtmp != NULL) matFree(fSWPNtmp)
958
959         fRefNorm = fDegNorm = fFBStmp1 = fFBStmp2 = fSWPNtmp = NULL
960         OPTTHROW( string("ERROR in FineDelay: Unknown error.\n"))
961     }
962 }
963
964 XFLOAT SQTimeAlignment::CalcMatchQuality(TA_SegList const *segList)
965 {
966     OPTTRY
967     {
968         if (segList == NULL || segList->size() == 0)
969             OPTTHROW( string("FineDelay did not execute successfully / invalid segments list."))
970
971         int const TA_SCALED_SHIFT_TOLERANCE =
972             round(TA_SHIFT_TOLERANCE * mCurSegmentRate / (XFLOAT)TA_SAMPLING_RATE)
973
974         int totMatchedLen   = 0,
975             totMatchableLen = 0
976
977         //Go through entire segments list.
978         //TA_SEG_MACHED segments count towards the match quality.
979         //TA_SEG_GUESSED are also counted towards the match quality if they
980         //are adjacent to TA_SEG_MATCHED egments and have a similar delay.
981         for (unsigned int i = 0  i < segList->size()  i++)
982         {
983             if ((*segList)[i].segType == TA_SEG_PAUSE)
984                 continue
985             if ((*segList)[i].segType == TA_SEG_MISSING)
986             {
987                 totMatchableLen += (*segList)[i].segLen
988                 continue
989             }
```

```
990
991             TA_SEG_TYPE curSegType = (*segList)[i].segType
992             int curShift       = (*segList)[i].degPos - (*segList)[i].refPos,
993                 curSegLen      = (*segList)[i].segLen,
994                 matchedSegsLen = curSegType == TA_SEG_MATCHED ? curSegLen : 0,
995                 guessedSegsLen = curSegType != TA_SEG_MATCHED ? curSegLen : 0,
996                 lastConsecSeg
997             XFLOAT matchedVSguessedRatio = (XFLOAT)0.0
998
999             for (lastConsecSeg = i+1
1000                 lastConsecSeg < (int)segList->size() &&
1001                 (*segList)[lastConsecSeg].segType != TA_SEG_MISSING &&
1002                 (*segList)[lastConsecSeg].segType != TA_SEG_PAUSE   &&
1003                 abs( (*segList)[lastConsecSeg].degPos - (*segList)[lastConsecSeg].refPos - curShift
    ) <= TA_SCALED_SHIFT_TOLERANCE
1004                 lastConsecSeg++)
1005             {
1006                 curSegLen  = (*segList)[lastConsecSeg].segLen
1007                 curSegType = (*segList)[lastConsecSeg].segType
1008
1009                 if (curSegType == TA_SEG_MATCHED)
1010                     matchedSegsLen += curSegLen
1011                 else
1012                     guessedSegsLen += curSegLen
1013             }
1014
1015             i = lastConsecSeg - 1
1016             totMatchableLen += matchedSegsLen + guessedSegsLen
1017
1018             if (matchedSegsLen > 0)
1019             {
1020                 matchedVSguessedRatio = matchedSegsLen / (XFLOAT)(matchedSegsLen + guessedSegsLen)
1021                 totMatchedLen += (int)(matchedSegsLen + matchedVSguessedRatio * guessedSegsLen)
1022             }
1023         }
1024
1025         if (totMatchableLen <= 0.0f)
1026             OPTTHROW( string("Total non-pause signal length in segment list = 0 ?!"))
1027
1028         return (totMatchedLen / (XFLOAT)totMatchableLen)
1029     }
1030     OPTCATCH((string errorMsg))
1031     {
1032         OPTTHROW( string("ERROR in CalcMatchQuality: " + errorMsg + "\n"))
1033     }
1034 }
1035
1036 void SQTimeAlignment::CalcDelaySpread()
1037 {
1038     OPTTRY
1039     {
1040         if (mSegments == NULL || mSegments->size() == 0)
1041             OPTTHROW( string("Empty segments list."))
1042
1043         int maxDelay, minDelay, i = 0
1044
1045         while (i < (int)mSegments->size() && (*mSegments)[i].segType != TA_SEG_MATCHED)
1046             i++
1047         if (i == (int)mSegments->size())
1048             return
1049
1050         maxDelay = minDelay =
1051             (*mSegments)[i].degPos - (*mSegments)[i].refPos
1052
1053         for (  i < (int)mSegments->size()  i++)
1054         {
1055             if ((*mSegments)[i].segType != TA_SEG_MATCHED)
1056                 continue
```

```
1057
1058            maxDelay = ((((*mSegments)[i].degPos - (*mSegments)[i].refPos) > (maxDelay)) ?
    ((*mSegments)[i].degPos - (*mSegments)[i].refPos) : (maxDelay))
1059            minDelay = ((((*mSegments)[i].degPos - (*mSegments)[i].refPos) < (minDelay)) ?
    ((*mSegments)[i].degPos - (*mSegments)[i].refPos) : (minDelay))
1060        }
1061
1062        mMinDelay = minDelay
1063        mMaxDelay = maxDelay
1064    }
1065    OPTCATCH((string errorMsg))
1066    {
1067        OPTTHROW( string("ERROR in CalcDelaySpread: " + errorMsg + "\n"))
1068    }
1069 }
1070
1071 void SQTimeAlignment::SlidingWinPowNorm(XFLOAT const *fIn, XFLOAT *fOut,
1072                                         long len,
1073                                         XFLOAT const &sigThreshdB,
1074                                         int WINLEN,
1075                                         XFLOAT const &signalMaxAmp,
1076                                         XFLOAT const &meanOutLevel,
1077                                         bool doAvoidShortBursts,
1078                                         int const hystLenInSamples,
1079                                         XFLOAT *tempBuff)
1080 {
1081    OPTTRY
1082    {
1083
1084        const double ROUNDFACTOR = 1e8
1085
1086        if ((WINLEN | 0x1) != WINLEN)
1087            WINLEN--
1088
1089        if (fIn == NULL || fOut == NULL || len < WINLEN+3 ||
1090            fIn == fOut || WINLEN < 2)
1091            OPTTHROW( string("Invalid input parameters. Note: In-place operation is not
    supported."))
1092
1093        double dWinEnergy  = 0.0
1094        double dEnergThr   =
1095               signalMaxAmp*signalMaxAmp * dBtoPow(sigThreshdB) * WINLEN
1096        int    hystCounter = 0
1097        XFLOAT fTemp
1098        rmvesq(fIn, &fTemp, WINLEN/2)
1099
1100        dWinEnergy = floor(ROUNDFACTOR*fTemp*fTemp * (int)(WINLEN/2))/ROUNDFACTOR
1101
1102        double *fInSquared = tempBuff != NULL ? tempBuff : (double*)matMalloc(len * sizeof(double))
1103
1104        matbSqr2(fIn, fInSquared, len)
1105
1106        //Process first WINLEN/2 data points
1107        for (int i = 0  i < WINLEN/2+1  i++)
1108        {
1109            if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
1110            {
1111                fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
1112                hystCounter = dWinEnergy > dEnergThr ?
1113                    (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
    :
1114                    (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
1115            }
1116            else
1117            {
1118                fOut[i]     = (XFLOAT)0.0
1119                hystCounter = 0
1120            }
```

```
1121
1122            dWinEnergy += floor(ROUNDFACTOR * fIn[i+WINLEN/2]*fIn[i+WINLEN/2])/ROUNDFACTOR
1123
1124        }
1125
1126        //Main loop
1127        for (int i = WINLEN/2+1  i < len - WINLEN/2  i++)
1128        {
1129            if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
1130            {
1131                fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
1132                hystCounter = dWinEnergy > dEnergThr ?
1133                    (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
    :
1134                    (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
1135            }
1136            else
1137            {
1138                fOut[i]     = (XFLOAT)0.0
1139                hystCounter = 0
1140            }
1141
1142            dWinEnergy = floor(matSum(&fInSquared[i - WINLEN/2], WINLEN) * ROUNDFACTOR)/ROUNDFACTOR
1143
1144        }
1145
1146        //Process last WINLEN/2 data points
1147        for (int i = len - WINLEN/2  i < len  i++)
1148        {
1149            if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
1150            {
1151                fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
1152                hystCounter = dWinEnergy > dEnergThr ?
1153                    (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
    :
1154                    (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
1155            }
1156            else
1157            {
1158                fOut[i]     = (XFLOAT)0.0
1159                hystCounter = 0
1160            }
1161
1162            dWinEnergy -= floor(ROUNDFACTOR * fIn[i-WINLEN/2-1]*fIn[i-WINLEN/2-1])/ROUNDFACTOR
1163
1164        }
1165
1166        if (doAvoidShortBursts)
1167        {
1168            int actStart = 0
1169            for (int i = 0  i < len  i++)
1170            {
1171                while ((i < len) && ( fOut[i] == (XFLOAT)0.0 )) i++
1172                actStart = i
1173                while ( (i < len ) && (fOut[i] != (XFLOAT)0.0) ) i++
1174                if (i - actStart < 2*WINLEN)
1175                    for (int j = actStart  j < i  j++)
1176                        fOut[j] = (XFLOAT)0.0
1177            }
1178        }
1179
1180        //Rescale processed data to desired level
1181        XFLOAT fScalefac
1182        rmvesq(fOut, &fScalefac, len)
1183        if (fScalefac > (XFLOAT)0.0)
1184        {
1185            fScalefac = pow((XFLOAT)10.0, (XFLOAT)meanOutLevel/(XFLOAT)20.0) / (fScalefac /
    signalMaxAmp)
```

```
            vsmul(fOut, fScalefac, fOut, len)
        }

        if(fInSquared && fInSquared != tempBuff)
            matFree(fInSquared)
    }
    OPTCATCH((string errorMsg))
    {
        OPTTHROW( string("ERROR in SQTimeAlignment::SlidingWinPowNorm:" + errorMsg + "\n"))
    }
}

void SQTimeAlignment::FindBestShift(XFLOAT const* fRef, int const lRef,
                                    XFLOAT const* fDeg, int const lDeg,
                                    int iNumLags, int const iDegOffset,
                                    XFLOAT &fMaxCorr, int &iMaxPos,
                                    bool  const doFindAbsMax,
                                    bool  const doWeightDegOffset,
                                    XFLOAT *fCorrs, XFLOAT *fWeighted,
                                    XFLOAT const guessReliability,
                                    short const normalizationType,
                                    bool  const checkPeakiness)
{
    OPTTRY
    {
        if (fRef == NULL || fDeg == NULL || fCorrs == NULL || fWeighted == NULL ||
            lRef <= 0 || lDeg <= 0 || iNumLags  < 2)
            OPTTHROW( string("Invalid input arguments."))
        if ((iNumLags | 0x1) != iNumLags)
            iNumLags++

        XFLOAT const FBS_MIN_PEAKINESS_FAC = (XFLOAT)20.0

        int   const FBS_MAXIMA_NUM   = 30
        XFLOAT const FBS_MAXIMA_RATIO =  (XFLOAT)0.9
        int   const FBS_NON_PENALIZED_ZONE
            = 256 * TA_SAMPLING_RATE / 8000
        int   const FBS_RAMP_LEN
            = 2000* TA_SAMPLING_RATE / 8000

        //Initialize cosine weights
        static bool   isCosineBowInitialized = false
        static XFLOAT cosineBowWeighting[2*FBS_RAMP_LEN+1]
        if (!isCosineBowInitialized)
        {
            for (int i = -FBS_RAMP_LEN  i <= FBS_RAMP_LEN  i++)
                cosineBowWeighting[i+FBS_RAMP_LEN] =
                    0.5f  *  ( cos(i/(XFLOAT)FBS_RAMP_LEN * (XFLOAT)PI) + 1 )
            isCosineBowInitialized = true
        }

        int numNonZeroSampRef = 0, numNonZeroSampDeg = 0
        for (int i = 0  i < lRef  i++)
            if (fRef[i] != 0.0f)
                numNonZeroSampRef++
        for (int i = 0  i < lDeg  i++)
            if (fDeg[i] != 0.0f)
                numNonZeroSampDeg++

        //Compute the normalizing factor:
        XFLOAT fDivisor
        int numNonZeroSampBoth
        XFLOAT fAutocorrRef  svesq(fRef, &fAutocorrRef, lRef)
        XFLOAT fAutocorrDeg  svesq(fDeg, &fAutocorrDeg, lDeg)

        switch (normalizationType)
        {
        case NORM_NONE:
```

```
1254            fDivisor = (XFLOAT)1.0
1255            break
1256        case NORM_SIMPLE_GEOMETRIC_MEAN:
1257            fDivisor = sqrt (fAutocorrRef*fAutocorrDeg)
1258            break
1259        case NORM_WEIGHTED_GEOMETRIC_MEAN:
1260            numNonZeroSampBoth = (((numNonZeroSampRef) < (numNonZeroSampDeg)) ? (numNonZeroSampRef)
    : (numNonZeroSampDeg))
1261            fDivisor = sqrt( (fAutocorrRef/numNonZeroSampRef) *
1262                             (fAutocorrDeg/numNonZeroSampDeg) )
1263                             * numNonZeroSampBoth
1264            break
1265        default:
1266            OPTTHROW( string ("No valid normalization type for correlation."))
1267            break
1268        }
1269
1270        //Compute the cross-correlation at iNumLags different shifts
1271        int lowestLag = -iNumLags/2 + iDegOffset
1272
1273        OPTTRY
1274        {
1275            matCrossCorr(matHandle, fRef, lRef, fDeg, lDeg, fCorrs, iNumLags, lowestLag)
1276        }
1277        OPTCATCH((...))
1278        {
1279            char* dbgs = "matCrossCorr Crash"
1280            OPTTHROW( string (dbgs))
1281        }
1282
1283        //Normalize the correlation vector
1284        vsdiv(fCorrs, fDivisor, fWeighted, iNumLags)
1285        if (doFindAbsMax)
1286            matbAbs2(fWeighted, fCorrs, iNumLags)
1287        else
1288            vmov(fWeighted, fCorrs, iNumLags)
1289
1290        //Weight correlations vector by distance from center shift
1291        if (doWeightDegOffset)
1292        {
1293            XFLOAT FBS_MIN_WEIGHT = (XFLOAT)0.50
1294
1295            if (guessReliability < 0)
1296                OPTTHROW( string("This should never happen!"))
1297            else
1298                FBS_MIN_WEIGHT = limit((XFLOAT)0.8 - guessReliability * (XFLOAT)0.3, (XFLOAT)0.5,
    (XFLOAT)0.8)
1299
1300            matbSet(FBS_MIN_WEIGHT, fWeighted, iNumLags)
1301
1302            int nonPenZoneStart = (((0) > (iNumLags/2 - FBS_NON_PENALIZED_ZONE/2)) ? (0) :
    (iNumLags/2 - FBS_NON_PENALIZED_ZONE/2)),
1303                nonPenZoneEnd   = (((iNumLags) < (iNumLags/2 + FBS_NON_PENALIZED_ZONE/2)) ?
    (iNumLags) : (iNumLags/2 + FBS_NON_PENALIZED_ZONE/2))
1304            matbSet(1.0f, fWeighted+nonPenZoneStart, nonPenZoneEnd-nonPenZoneStart+1)
1305
1306            int rampStart = (((0) > (nonPenZoneStart - (((iNumLags/2) < ((int)FBS_RAMP_LEN)) ?
    (iNumLags/2) : ((int)FBS_RAMP_LEN)))) ? (0) : (nonPenZoneStart - (((iNumLags/2) <
    ((int)FBS_RAMP_LEN)) ? (iNumLags/2) : ((int)FBS_RAMP_LEN)))),
1307                rampEnd   = (((iNumLags) < (nonPenZoneEnd + (((iNumLags/2) < ((int)FBS_RAMP_LEN)) ?
    (iNumLags/2) : ((int)FBS_RAMP_LEN)))) ? (iNumLags) : (nonPenZoneEnd + (((iNumLags/2)
    < ((int)FBS_RAMP_LEN)) ? (iNumLags/2) : ((int)FBS_RAMP_LEN))))
1308
1309            if (mMaxSigLenBuff == NULL || iNumLags > (((lRef) > (lDeg)) ? (lRef) : (lDeg)))
1310            {
1311                matAddProduct1(cosineBowWeighting + FBS_RAMP_LEN - (nonPenZoneStart-rampStart),
    1-FBS_MIN_WEIGHT,
1312                           fWeighted + rampStart,       nonPenZoneStart-1 - rampStart + 1)
```

```
                matAddProduct1(cosineBowWeighting + FBS_RAMP_LEN + 1,
1-FBS_MIN_WEIGHT,
                        fWeighted + nonPenZoneEnd+1, rampEnd-1 - (nonPenZoneEnd+1) + 1)
                vmul    (fWeighted, fCorrs, fWeighted, iNumLags)
            }
            else
            {
                vsmul(cosineBowWeighting + FBS_RAMP_LEN - (nonPenZoneStart-rampStart),
1-FBS_MIN_WEIGHT,
                      mMaxSigLenBuff + rampStart,       nonPenZoneStart-1 - rampStart + 1)
                vsmul(cosineBowWeighting + FBS_RAMP_LEN + 1,
1-FBS_MIN_WEIGHT,
                      mMaxSigLenBuff + nonPenZoneEnd+1, rampEnd-1 - (nonPenZoneEnd+1) + 1)
                vadd (fWeighted + rampStart,       mMaxSigLenBuff + rampStart,
                      fWeighted + rampStart,       nonPenZoneStart-1 - rampStart + 1)
                vadd (fWeighted + nonPenZoneEnd+1, mMaxSigLenBuff + nonPenZoneEnd+1,
                      fWeighted + nonPenZoneEnd+1, rampEnd-1 - (nonPenZoneEnd+1) + 1)
                vmul(fWeighted,      fCorrs, mMaxSigLenBuff, iNumLags)
                vmov(mMaxSigLenBuff,         fWeighted,      iNumLags)
            }
        }

        //Find max cross-corr in the (possibly weighted) vector
        int iMaxIndex = -1, tempIndex = -1, finalIndex, searchIdx
        XFLOAT tempMaxCorr = 0.0f, finalCorr
        if (!doWeightDegOffset)
            fMaxCorr = matMaxExt(fCorrs, iNumLags, &iMaxIndex)
        else
        {
            fMaxCorr = matMaxExt(fWeighted, iNumLags, &iMaxIndex)
            fWeighted[iMaxIndex] = 0.0f
            finalIndex = iMaxIndex
            finalCorr  = fMaxCorr

            for (int i = 0  fMaxCorr > 0.0f && i < FBS_MAXIMA_NUM-1  i++)
            {
                searchIdx = (((iNumLags-1-finalIndex) < (finalIndex)) ? (iNumLags-1-finalIndex) :
(finalIndex))
                tempMaxCorr = matMaxExt(fWeighted+searchIdx, 2*(iNumLags/2-searchIdx), &tempIndex)
                tempIndex += searchIdx

                if (tempMaxCorr < FBS_MAXIMA_RATIO * fMaxCorr || searchIdx == iNumLags/2)
                    break

                fWeighted[tempIndex] = 0.0f
                finalIndex = tempIndex
                finalCorr  = tempMaxCorr
            }
            iMaxIndex = finalIndex
            fMaxCorr  = finalCorr
        }

        //Clean up and leave directly if there is no cross-correlation whatsoever.
        if (fMaxCorr <= 0.0f)
        {
            iMaxPos = 0
            return
        }

        //Determine the peakiness of the *original* (non-weighted) cross-corr vector.
        if (checkPeakiness)
        {
            int nonZeroStart = 0, nonZeroEnd = iNumLags-1
            while (fCorrs[nonZeroStart] < (XFLOAT)0.01 && nonZeroStart < iNumLags-1)
                nonZeroStart++
            while (fCorrs[nonZeroEnd]   < (XFLOAT)0.01 && nonZeroEnd > 0)
                nonZeroEnd--
            if (nonZeroEnd-nonZeroStart+1 >= 21)
```

```
            {
                XFLOAT fCorrAvg
                fCorrAvg = matMean(fCorrs+nonZeroStart, nonZeroEnd-nonZeroStart+1)
                if (fCorrAvg > (XFLOAT)0.0 && fMaxCorr / fCorrAvg > FBS_MIN_PEAKINESS_FAC)
                    fMaxCorr = TA_INFINITE_CORR
            }
        }

        iMaxPos = iMaxIndex - iNumLags/2
    }
    OPTCATCH((string errorMsg))
    {
        OPTTHROW( string("ERROR in SQTimeAlignment::FindBestShift: " + errorMsg + "\n"))
    }
    OPTCATCH((...))
    {
        OPTTHROW( string("ERROR in SQTimeAlignment::FindBestShift: Unknown error.\n"))
    }
}

void SQTimeAlignment::ReserveVectorMemory(int minSegLen)
{
    if (mSegments == NULL || mRef == NULL || mDeg == NULL || minSegLen <= 0)
        OPTTHROW( string("ERROR in ReserveVectorMemory: Invalid input data.\n"))

    int const RVM_MAX_NUMSEGS = 100
    int const RVM_MIN_NUMSEGS =  10

    //Try to guess the number of segments ultimately used
    int approxNumSegs = ((((int)ceil(mRef->NrOfSamples() * mRef->SpeechActivity() / minSegLen)) >
((int)ceil(mDeg->NrOfSamples() * mDeg->SpeechActivity() / minSegLen))) ?
((int)ceil(mRef->NrOfSamples() * mRef->SpeechActivity() / minSegLen)) :
((int)ceil(mDeg->NrOfSamples() * mDeg->SpeechActivity() / minSegLen)))

    mSegments->reserve(limit(approxNumSegs, RVM_MIN_NUMSEGS, RVM_MAX_NUMSEGS))
}

int SQTimeAlignment::GetNextRefSegmentToMatch (int &uttStart, int &uttEnd,
                                               XFLOAT &uttThresh,
                                               XFLOAT const &FD_MIN_LEVEL,
                                               bool &doMatchLoudestUtt,
                                               bool &doIgnorePauses,
                                               int &gapSearchPos,
                                               XFLOAT const *env, int const &lEnv,
                                               int const &frameStep, int const &lSignal)
{
    OPTTRY
    {
        if (env == NULL || lEnv <= 0 || frameStep < 1 || lSignal < frameStep)
            OPTTHROW( string("Invalid input arguments."))

        while (true)
        {
            if (doMatchLoudestUtt) //Find loudest continuous speech above set threshold in Ref
            {
                bool uttFound
                FindLoudestUtt(env, lEnv, lSignal, frameStep,
                               uttThresh, uttFound,
                               uttStart, uttEnd)
                if (!uttFound && uttThresh > FD_MIN_LEVEL)
                {
                    uttThresh  = FD_MIN_LEVEL
                    continue
                }
                else if (!uttFound)
                    doMatchLoudestUtt = false
                else
                    return 0
```

```
1442            }
1443
1444            //Once no speech to match is left in the ref envelope,
1445            //find gaps in the segments list and match those.
1446            if (!doMatchLoudestUtt)
1447            {
1448                if(mSegments->size() == 0)
1449                {
1450
1451                    InsertSegment((((0) > (-mCrudeDelay)) ? (0) : (-mCrudeDelay)),
1452                                  mCrudeDelay,
1453                                  (((mRef->NrOfSamples() - (((0) > (-mCrudeDelay)) ? (0) :
    (-mCrudeDelay))) < (mDeg->NrOfSamples() - (((0) > (mCrudeDelay)) ?
    (0) : (mCrudeDelay)))) ? (mRef->NrOfSamples() - (((0) >
    (-mCrudeDelay)) ? (0) : (-mCrudeDelay))) : (mDeg->NrOfSamples() -
    (((0) > (mCrudeDelay)) ? (0) : (mCrudeDelay)))),
1454                                  TA_SEG_GUESSED,
1455                                  mDeg->NrOfSamples(), false,
1456                                  (XFLOAT*)env, lEnv, frameStep, MIN_LEVEL_DB)
1457                }
1458                int gapStart, gapEnd
1459                if (mSegments->findNextGapInRefSegList(gapStart, gapEnd,
1460                                                       gapSearchPos, lSignal)
1461                    < 0)
1462                {
1463                    if (!doIgnorePauses)
1464                        return -1
1465                    else
1466                    {
1467                        doIgnorePauses = false
1468                        gapSearchPos   = 0
1469                        continue
1470                    }
1471                }
1472                else
1473                {
1474                    if (gapStart > gapEnd)
1475                        OPTTHROW( string("findNextGapInRefSegList() returned invalid gap,\n\
1476                            probably due to a corruped segments list."))
1477
1478                    uttStart = gapStart
1479                    uttEnd   = gapEnd
1480                    return 0
1481                }
1482            }
1483        }
1484    }
1485    OPTCATCH((string errorMsg))
1486    {
1487        OPTTHROW( string("ERROR in GetNextRefSegmentToMatch: " + errorMsg + "\n"))
1488    }
1489 }
1490
1491 void SQTimeAlignment::FindLoudestUtt (XFLOAT const *fEnv, int const &envLen,
1492                                       int const &sigLen, int const &frameStep,
1493                                       XFLOAT const &uttThresh,
1494                                       bool &foundUtt,
1495                                       int &uttStartSamples, int &uttEndSamples)
1496 {
1497     OPTTRY
1498     {
1499         if (fEnv == NULL || envLen <= 3 || frameStep < 1 || sigLen < frameStep)
1500             OPTTHROW( string("Invalid input parameters."))
1501
1502         XFLOAT uttSum     = (XFLOAT)0.0,
1503                maxUtt     = (MIN_LEVEL_DB - 10.0f) * envLen
1504         int    uttLen     = 0,
1505                uttStartFrames = 0, uttEndFrames = 0
```

```
1506        int const FLU_MIN_UTTLEN = TA_MIN_SEGLEN / frameStep
1507
1508        foundUtt = false
1509        for (int i = 0  i < envLen  i++)
1510        {
1511            if (fEnv[i] >= uttThresh)
1512            {
1513                uttSum  += fEnv[i] - MIN_LEVEL_DB
1514                uttLen++
1515                if (i == envLen-1 &&
1516                    ((uttSum > maxUtt &&
1517                      (uttEndFrames - uttStartFrames + 1 < FLU_MIN_UTTLEN ||
1518                       uttLen - 1 >= FLU_MIN_UTTLEN))
1519                     ||
1520                     (uttEndFrames - uttStartFrames + 1 < FLU_MIN_UTTLEN &&
1521                      uttLen > uttEndFrames - uttStartFrames + 1)))
1522                {
1523                    maxUtt = uttSum
1524                    uttStartFrames = i-uttLen
1525                    uttEndFrames   = i
1526                    foundUtt = true
1527                }
1528            }
1529            else if (uttLen > 0)
1530            {
1531                if ((uttSum > maxUtt &&
1532                     (uttEndFrames - uttStartFrames + 1 < FLU_MIN_UTTLEN ||
1533                      uttLen - 1 >= FLU_MIN_UTTLEN))
1534                    ||
1535                    (uttEndFrames - uttStartFrames + 1 < FLU_MIN_UTTLEN &&
1536                     uttLen > uttEndFrames - uttStartFrames + 1))
1537                {
1538                    maxUtt = uttSum
1539                    uttStartFrames = i-uttLen
1540                    uttEndFrames   = i-1
1541                    foundUtt = true
1542                }
1543                uttSum = (XFLOAT)0.0
1544                uttLen = 0
1545            }
1546        }
1547
1548        if (foundUtt)
1549        {
1550            uttStartSamples = (int)(uttStartFrames * frameStep)
1551            uttStartSamples = ((((((0) > (uttStartSamples)) ? (0) : (uttStartSamples))) < (sigLen))
    ? ((((0) > (uttStartSamples)) ? (0) : (uttStartSamples))) : (sigLen))
1552            uttEndSamples   = (int)((uttEndFrames+1) * frameStep)
1553            uttEndSamples   = ((((((0) > (uttEndSamples)) ? (0) : (uttEndSamples))) < (sigLen)) ?
    ((((0) > (uttEndSamples)) ? (0) : (uttEndSamples))) : (sigLen))
1554        }
1555    }
1556    OPTCATCH((string errorMsg))
1557    {
1558        OPTTHROW( string("ERROR in FindLoudestUtt: " + errorMsg + "\n"))
1559    }
1560 }
1561
1562 void SQTimeAlignment::GuessBestShift (int const iRefPos,
1563                                       int const degStart,
1564                                       int const degLen,
1565                                       XFLOAT const *degSig,
1566                                       int const totDegLen,
1567                                       int const uttLen,
1568                                       int &iGuessedShift,
1569                                       XFLOAT *guessReliability,
1570                                       int const endOf1stHalf)
1571 {
```

```
1572    OPTTRY
1573    {
1574        if (mSegments == NULL || iRefPos < 0 || degStart < 0 || degSig == NULL ||
1575            (guessReliability != NULL && degLen <= 0))
1576            OPTTHROW( string("Invalid input arguments."))
1577
1578        if (guessReliability != NULL)
1579            *guessReliability = -1.0f
1580
1581        TA_SegList::iterator nextSeg = mSegments->findInsLoc(iRefPos)
1582        TA_SegList::iterator prevSeg = nextSeg == mSegments->begin() ?
1583            mSegments->end() : nextSeg - 1
1584
1585        if (nextSeg != mSegments->end())
1586        {
1587            while (nextSeg != mSegments->end() &&
1588                   (nextSeg->segType == TA_SEG_MISSING || nextSeg->segType == TA_SEG_PAUSE))
1589                nextSeg++
1590
1591            if (endOf1stHalf > 0 && nextSeg != mSegments->end() &&
1592                iRefPos < endOf1stHalf && nextSeg->refPos > endOf1stHalf)
1593                nextSeg = mSegments->end()
1594        }
1595        if (prevSeg != mSegments->end())
1596        {
1597            while (prevSeg != mSegments->begin() &&
1598                   (prevSeg->segType == TA_SEG_MISSING || prevSeg->segType == TA_SEG_PAUSE))
1599                prevSeg--
1600            if (prevSeg->segType == TA_SEG_MISSING || prevSeg->segType == TA_SEG_PAUSE)
1601                prevSeg = mSegments->end()
1602
1603            if (endOf1stHalf > 0 && prevSeg != mSegments->end() &&
1604                iRefPos > endOf1stHalf && prevSeg->refPos < endOf1stHalf)
1605                prevSeg = mSegments->end()
1606        }
1607
1608        if (nextSeg == mSegments->end() && prevSeg == mSegments->end())
1609        {
1610            iGuessedShift = mCrudeDelay
1611
1612            iGuessedShift = (((0 - iRefPos) > (iGuessedShift)) ? (0 - iRefPos) : (iGuessedShift))
1613
1614            if (degLen > 0)
1615            {
1616                iGuessedShift = (((degStart - iRefPos) > (iGuessedShift)) ? (degStart - iRefPos) :
    (iGuessedShift))
1617                iGuessedShift = ((((degStart+degLen-1) - (iRefPos+uttLen-1)) < (iGuessedShift)) ?
    ((degStart+degLen-1) - (iRefPos+uttLen-1)) : (iGuessedShift))
1618            }
1619            return
1620        }
1621
1622        if (prevSeg == mSegments->end() ||
1623            (nextSeg != mSegments->end() &&
1624             iRefPos+uttLen == nextSeg->refPos &&
1625             prevSeg->refPos+prevSeg->segLen < iRefPos))
1626        {
1627            iGuessedShift = nextSeg->degPos - nextSeg->refPos
1628            if (guessReliability != NULL)
1629                *guessReliability = RateGuessReliability(nextSeg->refPos - (iRefPos + uttLen - 1),
1630                                                         uttLen, degStart, degLen, degSig,
    totDegLen)
1631        }
1632
1633        else if (nextSeg == mSegments->end() ||
1634                 (prevSeg != mSegments->end() &&
1635                  iRefPos == prevSeg->refPos+prevSeg->segLen &&
1636                  nextSeg->refPos > iRefPos+uttLen))
```

```
1637        {
1638            iGuessedShift = prevSeg->degPos - prevSeg->refPos
1639            if (guessReliability != NULL)
1640                *guessReliability = RateGuessReliability(iRefPos - (prevSeg->refPos +
    prevSeg->segLen - 1),
1641                                                         uttLen, degStart, degLen, degSig,
    totDegLen)
1642        }
1643
1644        else if (iRefPos+uttLen == nextSeg->refPos &&
1645                 iRefPos == prevSeg->refPos+prevSeg->segLen)
1646        {
1647            if (prevSeg->segType == nextSeg->segType)
1648                iGuessedShift = prevSeg->segLen >= nextSeg->segLen ?
1649                    prevSeg->degPos - prevSeg->refPos :
1650                    nextSeg->degPos - nextSeg->refPos
1651            else
1652                iGuessedShift = prevSeg->segType == TA_SEG_MATCHED ?
1653                    prevSeg->degPos - prevSeg->refPos :
1654                    nextSeg->degPos - nextSeg->refPos
1655
1656            if (guessReliability != NULL)
1657                *guessReliability = 1
1658        }
1659
1660        else
1661        {
1662            int gapLen        = nextSeg->refPos - (prevSeg->refPos + prevSeg->segLen) + 1
1663            int distToPrevSeg = iRefPos - (prevSeg->refPos + prevSeg->segLen - 1)
1664            int distToNextSeg = nextSeg->refPos - (iRefPos + uttLen - 1)
1665            int prevSegShift  = prevSeg->degPos - prevSeg->refPos
1666            int nextSegShift  = nextSeg->degPos - nextSeg->refPos
1667            if (gapLen <= 0 || gapLen-uttLen <= 0)
1668            {
1669                iGuessedShift = prevSegShift
1670                if (guessReliability != NULL)
1671                    *guessReliability = 0.0f
1672            }
1673            else
1674            {
1675                iGuessedShift = (prevSegShift * (gapLen-uttLen-distToPrevSeg) +
1676                                 nextSegShift * (gapLen-uttLen-distToNextSeg)) / (gapLen-uttLen)
1677                if (guessReliability != NULL)
1678                    *guessReliability = RateGuessReliability((((distToPrevSeg) < (distToNextSeg)) ?
    (distToPrevSeg) : (distToNextSeg)),
1679                                                             abs(prevSegShift - nextSegShift))
1680            }
1681        }
1682
1683        int uncorrectedGuess = iGuessedShift
1684
1685        iGuessedShift = (((0 - iRefPos) > (iGuessedShift)) ? (0 - iRefPos) : (iGuessedShift))
1686
1687        if (degLen > 0)
1688        {
1689            iGuessedShift = (((degStart - iRefPos) > (iGuessedShift)) ? (degStart - iRefPos) :
    (iGuessedShift))
1690            iGuessedShift = ((((degStart+degLen-1) - (iRefPos+uttLen-1)) < (iGuessedShift)) ?
    ((degStart+degLen-1) - (iRefPos+uttLen-1)) : (iGuessedShift))
1691        }
1692        else
1693        {
1694            iGuessedShift = degStart - (iRefPos+uttLen/2)
1695            iGuessedShift = (((0 - iRefPos) > (iGuessedShift)) ? (0 - iRefPos) : (iGuessedShift))
1696            iGuessedShift = (((totDegLen - (iRefPos+uttLen)) < (iGuessedShift)) ? (totDegLen -
    (iRefPos+uttLen)) : (iGuessedShift))
1697        }
1698
```

```
1699        if (iGuessedShift != uncorrectedGuess && guessReliability != NULL)
1700            *guessReliability = 0.0f
1701
1702        return
1703    }
1704    OPTCATCH((string errorMsg))
1705    {
1706        OPTTHROW( string("ERROR in GuessBestShift: " + errorMsg + "\n"))
1707    }
1708    OPTCATCH((...))
1709    {
1710        OPTTHROW( string("ERROR in GuessBestShift: Unknown error.\n"))
1711    }
1712 }
1713
1714 XFLOAT SQTimeAlignment::RateGuessReliability(int distToClosestSeg,
1715                                              int shiftDiffBetweenSegs)
1716 {
1717     if (distToClosestSeg < 0)
1718         return (XFLOAT)-1.0
1719
1720     int const RGR_MAX_NONPENALIZED_DIST      =
1721         6000 * TA_SAMPLING_RATE / 8000
1722     int const RGR_MAX_DIST                   =
1723         round(1.5f * TA_SAMPLING_RATE)
1724     int const RGR_MAX_NONPENALIZED_SHIFTDIFF =
1725         400 * TA_SAMPLING_RATE / 8000
1726     int const RGR_MAX_SHIFTDIFF              =
1727         6000 * TA_SAMPLING_RATE / 8000
1728
1729     XFLOAT reliability
1730
1731     reliability = limit((XFLOAT)1.0 - (distToClosestSeg - RGR_MAX_NONPENALIZED_DIST) /
1732                         (XFLOAT)(RGR_MAX_DIST - RGR_MAX_NONPENALIZED_DIST),
1733                        (XFLOAT)0.0, (XFLOAT)1.0)
1734
1735     reliability *= limit((XFLOAT)1.0 - (shiftDiffBetweenSegs - RGR_MAX_NONPENALIZED_SHIFTDIFF) /
1736                          (XFLOAT)(RGR_MAX_SHIFTDIFF - RGR_MAX_NONPENALIZED_SHIFTDIFF),
1737                         (XFLOAT)0.0, (XFLOAT)1.0)
1738
1739     return reliability
1740 }
1741
1742 XFLOAT SQTimeAlignment::RateGuessReliability(int distToClosestSeg, int uttLen,
1743                                              int degStart, int availDegLen,
1744                                              XFLOAT const *degSig, int totDegLen)
1745 {
1746     if (distToClosestSeg < 0 || uttLen < 1 || degStart < 0 || availDegLen < 1 ||
1747         degSig == NULL || totDegLen < 1 || totDegLen < degStart+availDegLen)
1748         OPTTHROW( string("ERROR in RateGuessReliability: Invalid input arguments.\n"))
1749
1750     XFLOAT const RGR_MAX_NONPENALIZED_CHOICEFAC =
1751         (XFLOAT)3.0
1752     XFLOAT const RGR_MAX_CHOICEFAC              =
1753         (XFLOAT)8.0
1754     int   const RGR_MIN_UTTLEN                  =
1755         round(0.25f * TA_SAMPLING_RATE)
1756
1757     XFLOAT reliability = RateGuessReliability(distToClosestSeg, 0)
1758
1759     int numNonSilentSamples = availDegLen, cnt
1760     for (int i = degStart  i < degStart+availDegLen  i++)
1761     {
1762         for (cnt = 0  i < degStart+availDegLen && degSig[i] == 0.0f  i++, cnt++)
1763         if (cnt > 40)
1764             numNonSilentSamples -= cnt
1765     }
1766     numNonSilentSamples = (((numNonSilentSamples) > (0)) ? (numNonSilentSamples) : (0))
```

```
1767
1768     XFLOAT choiceFac = numNonSilentSamples / (XFLOAT)(((uttLen) > (RGR_MIN_UTTLEN)) ? (uttLen) :
    (RGR_MIN_UTTLEN))
1769
1770     reliability *= limit((XFLOAT)1.0 - (choiceFac - RGR_MAX_NONPENALIZED_CHOICEFAC) /
1771                          (RGR_MAX_CHOICEFAC - RGR_MAX_NONPENALIZED_CHOICEFAC),
1772                         (XFLOAT)0.0, (XFLOAT)1.0)
1773
1774     return reliability
1775 }
1776
1777 void SQTimeAlignment::CheckCurrentDegSeg (int const &uttStart, XFLOAT const *degSignal,
1778                                           int const &lDegSignal,
1779                                           bool const &doCheck,
1780                                           bool &flag,
1781                                           int &iDegStart, int &iMaxDegLen)
1782 {
1783     OPTTRY
1784     {
1785         if (uttStart < 0 || lDegSignal < 1 || mSegments == NULL)
1786             OPTTHROW( string("Invalid input parameters."))
1787
1788         if (mSegments->size() == 0)
1789         {
1790             iDegStart  = 0
1791             iMaxDegLen = lDegSignal
1792             return
1793         }
1794
1795         TA_SegList::iterator nextSeg = mSegments->findInsLoc(uttStart)
1796         TA_SegList::iterator prevSeg = nextSeg == mSegments->begin() ?
1797             mSegments->end() : nextSeg - 1
1798
1799         //Find the segments surrounding uttStart to find a gap of unused degraded signal.
1800         if (nextSeg != mSegments->end())
1801         {
1802             for (
1803                  nextSeg != mSegments->end() &&
1804                  (nextSeg->segType == TA_SEG_MISSING || nextSeg->segType == TA_SEG_PAUSE)
1805                  nextSeg++)
1806         }
1807         if (prevSeg != mSegments->end())
1808         {
1809             for (
1810                  prevSeg != mSegments->begin() &&
1811                  (prevSeg->segType == TA_SEG_MISSING || prevSeg->segType == TA_SEG_PAUSE)
1812                  prevSeg--)
1813             if (prevSeg->segType == TA_SEG_MISSING || prevSeg->segType == TA_SEG_PAUSE)
1814                 prevSeg = mSegments->end()
1815         }
1816
1817         if (prevSeg == mSegments->end())
1818             iDegStart = 0
1819         else
1820             iDegStart = prevSeg->degPos + prevSeg->segLen
1821
1822         if (nextSeg == mSegments->end())
1823             iMaxDegLen = lDegSignal - iDegStart
1824         else
1825             iMaxDegLen = nextSeg->degPos - iDegStart
1826
1827         flag = false
1828         if (doCheck)
1829         {
1830             int leadingSilenceLen, trailingSilenceLen, i
1831
1832             for (i = iDegStart, leadingSilenceLen = 0
1833                  i < iDegStart+iMaxDegLen && degSignal[i] == 0.0
```

```
                 i++, leadingSilenceLen++)

            for (i = iMaxDegLen+iDegStart-1, trailingSilenceLen = 0
                 i >= iDegStart+leadingSilenceLen && degSignal[i] == 0.0
                 i--, trailingSilenceLen++)

            if (leadingSilenceLen + trailingSilenceLen >= iMaxDegLen)
                flag = true
        }

        iMaxDegLen = (((iMaxDegLen) < (lDegSignal - iDegStart)) ? (iMaxDegLen) : (lDegSignal - iDegStart))
        return
    }
    OPTCATCH((string errorMsg))
    {
        OPTTHROW( string("ERROR in CheckCurrentDegSeg: " + errorMsg + "\n"))
    }
}

XFLOAT SQTimeAlignment::SpanCorr(XFLOAT const *ref, XFLOAT const *deg,
                                 int const lDeg, int const iBestShift,
                                 XFLOAT const THR,
                                 XFLOAT const fMaxCorr,
                                 int& uttStartSamples, int& spanLen,
                                 XFLOAT *buff, XFLOAT *tmpBuff, int const FD_MIN_SPANLEN)
{
    OPTTRY
    {
        if (ref == NULL || deg == NULL || lDeg <= 0 || spanLen <= 0 || buff == NULL ||
            tmpBuff == NULL || uttStartSamples < 0 || THR < 0)
            OPTTHROW( string("Invalid input arguments."))

        //Shorten spanLen if iBestShift makes it go past the deg. signal boundaries
        if (uttStartSamples + iBestShift < 0)
        {
            spanLen        += uttStartSamples + iBestShift
            uttStartSamples = 0 - iBestShift
        }
        spanLen = (((spanLen) < (lDeg - (uttStartSamples + iBestShift))) ? (spanLen) : (lDeg - (uttStartSamples + iBestShift)))

        if (spanLen <= 0)
            return 0.0

        XFLOAT fRefAutocorr, fDegAutocorr, fDivisor
        svesq(ref+uttStartSamples, &fRefAutocorr, spanLen)  fRefAutocorr /= spanLen
        svesq(deg+uttStartSamples+iBestShift, &fDegAutocorr, spanLen)  fDegAutocorr /= spanLen
        fDivisor = sqrt(fRefAutocorr * fDegAutocorr)

        vmul(ref+uttStartSamples, deg+uttStartSamples+iBestShift, buff, spanLen)

        XFLOAT fTotCorr = 0.0, fSpanCorr = 0.0
        sve(buff, &fTotCorr, spanLen)
        if (fTotCorr < 0.0)
            vneg(buff, buff, spanLen)

        vsdiv(buff,    fDivisor, tmpBuff, spanLen)
        vsadd(tmpBuff, -THR,     buff,    spanLen)

        XFLOAT fMaxSum = 0.0, fSum = 0.0, fMaxSum2 = 0.0, fThr
        int spanStart = 0, spanEnd, curSpanEnd, spanLen2
        int curSpanStart = 0, curSpanLen = spanLen
        int const FBS_MAX_SPANSTART = spanLen - FD_MIN_SPANLEN

        //Compute span cross-correlation
        while (true)
        {
```

```
1900            for (
1901                 spanStart < FBS_MAX_SPANSTART && buff[spanStart] <= 0.0
1902                 spanStart++)
1903            if (spanStart > FBS_MAX_SPANSTART) break
1904
1905            curSpanEnd = spanEnd  = spanStart
1906            fSum       = fMaxSum2 = fThr = 0.0
1907            spanLen2   = 1
1908            do
1909            {
1910                fSum += buff[spanEnd] - fThr
1911
1912                if (fSum > fMaxSum2 && spanLen2 >= FD_MIN_SPANLEN)
1913                    fMaxSum2 = fSum, curSpanEnd = spanEnd
1914
1915                spanEnd++
1916                spanLen2 = spanEnd-spanStart+1
1917
1918                fThr = spanLen2 <= FD_MIN_SPANLEN ? 0.0 : (((0.2) <
    (((XFLOAT)(spanLen2-FD_MIN_SPANLEN)/FD_MIN_SPANLEN) * 0.7 *
    fMaxSum2/(curSpanEnd-spanStart+1))) ? (0.2) :
    (((XFLOAT)(spanLen2-FD_MIN_SPANLEN)/FD_MIN_SPANLEN) * 0.7 *
    fMaxSum2/(curSpanEnd-spanStart+1)))
1919            }
1920            while(spanEnd < spanLen && fSum > 0)
1921
1922            if (fMaxSum2 > fMaxSum)
1923            {
1924                fMaxSum       = fMaxSum2
1925                curSpanStart  = spanStart
1926                curSpanLen    = curSpanEnd+1-spanStart
1927            }
1928            spanStart = spanEnd
1929        }
1930
1931        //Also compute span correlation running backwards
1932        spanStart = 0
1933        spanEnd   = spanLen-1
1934        bool backwardSpanFound = false
1935        int curSpanStart2 = 0, curSpanEnd2 = spanLen-1, curSpanLen2 = spanLen
1936        while (true)
1937        {
1938            for (
1939                 spanEnd > FD_MIN_SPANLEN-1 && buff[spanEnd] <= 0.0
1940                 spanEnd--)
1941            if (spanEnd < FD_MIN_SPANLEN-1) break
1942
1943            curSpanStart2 = spanStart = spanEnd
1944            fSum          = fMaxSum2  = fThr = 0.0
1945            spanLen2      = 1
1946            do
1947            {
1948                fSum += buff[spanStart] - fThr
1949
1950                if (fSum > fMaxSum2 && spanLen2 >= FD_MIN_SPANLEN)
1951                    fMaxSum2 = fSum, curSpanStart2 = spanStart
1952
1953                spanStart--
1954                spanLen2 = spanEnd-spanStart+1
1955
1956                fThr = spanLen2 <= FD_MIN_SPANLEN ? 0.0 : (((0.2) <
    (((XFLOAT)(spanLen2-FD_MIN_SPANLEN)/FD_MIN_SPANLEN) * 0.3 *
    fMaxSum2/(spanEnd-curSpanStart2+1))) ? (0.2) :
    (((XFLOAT)(spanLen2-FD_MIN_SPANLEN)/FD_MIN_SPANLEN) * 0.3 *
    fMaxSum2/(spanEnd-curSpanStart2+1)))
1957            }
1958            while(spanStart >= 0 && fSum > 0)
1959
```

```
1960            if (fMaxSum2 > fMaxSum)
1961            {
1962                fMaxSum     = fMaxSum2
1963                curSpanEnd2 = spanEnd
1964                curSpanLen2 = curSpanEnd2+1-curSpanStart2
1965                backwardSpanFound = true
1966            }
1967            spanEnd = spanStart
1968        }
1969
1970        XFLOAT fSpanCorr2
1971        mve(buff+curSpanStart,  &fSpanCorr,  curSpanLen)
1972        mve(buff+curSpanStart2, &fSpanCorr2, curSpanLen2)
1973        if (backwardSpanFound && fSpanCorr2 > fSpanCorr)
1974        {
1975            curSpanStart = curSpanStart2
1976            curSpanEnd   = curSpanEnd2
1977            curSpanLen   = curSpanLen2
1978            fSpanCorr    = fSpanCorr2
1979        }
1980
1981        fSpanCorr       += THR
1982        uttStartSamples += curSpanStart
1983        spanLen          = curSpanLen
1984
1985        return fSpanCorr
1986    }
1987    OPTCATCH((string errorMsg))
1988    {
1989        OPTTHROW( string("ERROR in SpanCorr: " + errorMsg + "\n"))
1990    }
1991 }
1992
1993 void SQTimeAlignment::InsertSegment(int const &uttStart, int const &uttShift,
1994                                     int const &uttLen, int const &segType,
1995                                     int const &lDegSigLen,
1996                                     bool const &doMatchLoudestUtt, XFLOAT *env,
1997                                     int const &lEnv, int const &frameStep,
1998                                     XFLOAT const &envSilenceVal,
1999
2000                                     XFLOAT guessFac)
2001
2002 {
2003     OPTTRY
2004     {
2005         if (uttStart < 0 || uttLen <= 0 ||
2006             ((uttShift + uttStart < 0 || uttStart + uttLen + uttShift > lDegSigLen) &&
2007              (segType != TA_SEG_MISSING)))
2008             OPTTHROW( string ("Invalid input arguments."))
2009
2010         if (doMatchLoudestUtt)
2011             RemoveUttFromEnv(env, lEnv, frameStep, envSilenceVal, uttStart, uttLen)
2012
2013         if (guessFac < 0.0)
2014             switch(segType)
2015             {
2016                 case TA_SEG_MATCHED: guessFac = 1.0  break
2017                 case TA_SEG_GUESSED:
2018                 case TA_SEG_PAUSE:   guessFac = 0.5  break
2019                 case TA_SEG_MISSING: guessFac = 0.0  break
2020             }
2021
2022         int uttMaxEnd = 0
2023         switch(segType)
2024         {
2025         case TA_SEG_MATCHED:
2026         case TA_SEG_GUESSED:
2027
```

```
2028                                    mSegments->insert(mSegments->findInsLoc(uttStart),
2029                                                      TA_segStruct(uttStart,
2030                                                                   uttStart + uttShift,
2031                                                                   uttLen,
2032                                                                   (TA_SEG_TYPE)segType,
2033
2034                                                                   false, guessFac))
2035
2036                                break
2037            case TA_SEG_MISSING:
2038            case TA_SEG_PAUSE:   mSegments->insert(mSegments->findInsLoc(uttStart),
2039                                                   TA_segStruct(uttStart,
2040                                                                limit(uttStart + uttShift, 0,
    lDegSigLen-uttLen),
2041                                                                uttLen,
2042                                                                (TA_SEG_TYPE)segType,
2043
2044                                                                false, guessFac))
2045
2046                break
2047            default:             OPTTHROW( string("Unrecognized TA_SEG_TYPE."))
2048            }
2049        }
2050        OPTCATCH((string errorMsg))
2051        {
2052            OPTTHROW( string("ERROR in InsertSegment: " + errorMsg + "\n"))
2053        }
2054    }
2055
2056    void SQTimeAlignment::RemoveUttFromEnv(XFLOAT *env, int const &lEnv,
2057                                           int const &frameStep, XFLOAT const &envSilenceVal,
2058                                           int const &start, int const &len)
2059    {
2060        OPTTRY
2061        {
2062            if (env == NULL || lEnv < 1 || frameStep < 1 || start < 0 || len < 0)
2063                OPTTHROW( string("Invalid input arguments."))
2064
2065            //Convert from samples to frames
2066            int startFrames = (((start / frameStep) > (0)) ? (start / frameStep) : (0))
2067            int endFrames   = ((((start+len-1) / frameStep) < (lEnv-1)) ? ((start+len-1) / frameStep) :
    (lEnv-1))
2068
2069            for (int i = startFrames  i <= endFrames  i++)
2070                env[i] = envSilenceVal
2071
2072        }
2073        OPTCATCH((string errorMsg))
2074        {
2075            OPTTHROW( string("ERROR in RemoveUttFromEnv: " + errorMsg + "\n"))
2076        }
2077    }
2078
2079    void SQTimeAlignment::WriteMatchedSegment(int &spanStart, int &spanLen, int const &iBestShift,
2080                                              int const &iDegStart, int const &iMaxDegLen, int const
    &lDegNorm,
2081                                              bool const &doRemoveCurUtt, XFLOAT *env, int const &lEnv,
2082                                              int const &frameStep, XFLOAT const &envSilenceVal)
2083    {
2084        OPTTRY
2085        {
2086            if (spanStart < 0 || spanLen < 1 || iDegStart < 0 || iMaxDegLen < 1)
2087                OPTTHROW( string("Invalid input arguments."))
2088
2089            //Check iBestShift: Are we intruding into unavailable deg. parts?
2090            if (spanStart+iBestShift < iDegStart)
2091            {
2092                int overLapLen = (((iDegStart - (spanStart+iBestShift)) < (spanLen)) ? (iDegStart -
```

```
    (spanStart+iBestShift)) : (spanLen))
2093            InsertSegment(spanStart, iBestShift, overLapLen,
2094                          TA_SEG_MISSING, lDegNorm,
2095                          doRemoveCurUtt, env, lEnv,
2096                          frameStep, envSilenceVal)
2097            spanLen  -= overLapLen
2098            spanStart = iDegStart - iBestShift
2099        }
2100
2101        if (spanStart+spanLen+iBestShift > iDegStart+iMaxDegLen)
2102        {
2103            int overLapLen =
2104                (((spanStart+spanLen+iBestShift - (iDegStart+iMaxDegLen)) < (spanLen)) ?
    (spanStart+spanLen+iBestShift - (iDegStart+iMaxDegLen)) : (spanLen))
2105            InsertSegment(iDegStart+iMaxDegLen-iBestShift, iBestShift, overLapLen,
2106                          TA_SEG_MISSING, lDegNorm,
2107                          doRemoveCurUtt, env, lEnv,
2108                          frameStep, envSilenceVal)
2109            spanLen  -= overLapLen
2110        }
2111
2112        //Write remaining matched segment using iBestShift
2113        if (spanLen > 0)
2114            InsertSegment(spanStart, iBestShift, spanLen,
2115                          TA_SEG_MATCHED, lDegNorm,
2116                          doRemoveCurUtt, env, lEnv,
2117                          frameStep, envSilenceVal)
2118    }
2119    OPTCATCH((string errorMsg))
2120    {
2121        OPTTHROW( string("ERROR in WriteMatchedSegment: " + errorMsg + "\n"))
2122    }
2123 }
2124
2125 void SQTimeAlignment::CheckCurrentRefSeg(XFLOAT const *ref, int const lRef,
2126                                          int &uttStart, int &uttEnd,
2127                                          bool &flag, int const FD_MAX_SEGLEN)
2128 {
2129    OPTTRY
2130    {
2131        if (ref == NULL || lRef <= 0 || uttStart < 0 || uttEnd < 0)
2132            OPTTHROW( string("Invalid input arguments."))
2133
2134        uttEnd = (((uttEnd) < (mSegments->findMaxSegEndRef(uttEnd, lRef-1))) ? (uttEnd) :
    (mSegments->findMaxSegEndRef(uttEnd, lRef-1)))
2135
2136        //Avoid running into existing segments
2137        TA_SegList::iterator nextSeg = mSegments->findInsLoc(uttStart)
2138        TA_SegList::iterator prevSeg = nextSeg == mSegments->begin() ?
2139            mSegments->end() : nextSeg - 1
2140        if (prevSeg != mSegments->end())
2141            uttStart = __max(prevSeg->refPos + prevSeg->segLen - 1, uttStart)
2142        if (uttStart > uttEnd)
2143            return
2144
2145        flag = false
2146        int cnt = 0, i
2147        int const THR = 256
2148        for (i = uttStart  i < uttEnd+1  i++)
2149        {
2150            if (ref[i] == 0.0) cnt++
2151            else
2152            {
2153                if (cnt >= THR) break
2154                else            cnt = 0
2155            }
2156        }
2157        if (cnt >= THR)
```

```
2158        {
2159            if (i - cnt == uttStart)
2160            {
2161                flag   = true
2162                uttEnd = i - 1
2163            }
2164            else
2165            {
2166                flag   = false
2167                uttEnd = i - cnt - 1
2168            }
2169        }
2170
2171        if (!flag)
2172            uttEnd = (((uttEnd) < (uttStart + FD_MAX_SEGLEN - 1)) ? (uttEnd) : (uttStart +
    FD_MAX_SEGLEN - 1))
2173    }
2174    OPTCATCH((string errorMsg))
2175    {
2176        OPTTHROW( string("ERROR in CheckCurrentRefSeg: " + errorMsg + "\n"))
2177    }
2178    OPTCATCH((...))
2179    {
2180        OPTTHROW( string("ERROR in CheckCurrentRefSeg: Unknown error.\n"))
2181    }
2182 }
2183
2184 void SQTimeAlignment::FixExtremeMatches(XFLOAT const *fRefNorm, XFLOAT const *fDegNorm,
2185                                         int const &lRefNorm, int const &lDegNorm)
2186 {
2187     TA_SegList *SegListCopy = NULL
2188
2189     OPTTRY
2190     {
2191         if (lRefNorm < 1 || lDegNorm < 1 || fRefNorm == NULL || fDegNorm == NULL ||
2192             mSegments == NULL || mSegments->size() == 0)
2193             OPTTHROW( string("ERROR in FixExtremeMatches: Invalid input params or segments
    list.\n"))
2194
2195         int const FEM_MIN_NR_MATCHED_SEGS = 10
2196
2197         //Verify that there are enough matched segments for a good estimation
2198         int numMatchedSegs = 0
2199         for (int i = 0  i < (int)mSegments->size()  i++)
2200             if ((*mSegments)[i].segType == TA_SEG_MATCHED)
2201                 numMatchedSegs++
2202         if (numMatchedSegs < FEM_MIN_NR_MATCHED_SEGS)
2203             return
2204
2205         //Estimate the range of 'realistic' values for the shifts betw. ref and deg.
2206         int lowerOuterFence_1stHalf, lowerOuterFence_2ndHalf,
2207             upperOuterFence_1stHalf, upperOuterFence_2ndHalf,
2208             endOf1stHalf
2209         EstimateShiftsRange(numMatchedSegs, lRefNorm, endOf1stHalf,
2210                             lowerOuterFence_1stHalf, lowerOuterFence_2ndHalf,
2211                             upperOuterFence_1stHalf, upperOuterFence_2ndHalf)
2212
2213         //Search and remove extreme outliers (i.e. beyond fences)
2214         int prevGoodSeg = -1, nextGoodSeg = -1, firstBadSeg = -1, lastBadSeg = -1
2215         int numPrevBadActSegs = 0
2216         bool badSegFound = false
2217         for (int i = 0  i <= (int)mSegments->size()  i++)
2218         {
2219             if (i < (int)mSegments->size() &&
2220                 ((*mSegments)[i].segType == TA_SEG_MATCHED ||
2221                  (*mSegments)[i].segType == TA_SEG_GUESSED ||
2222
2223                  (badSegFound && (((*mSegments)[firstBadSeg].refPos < endOf1stHalf) ==
```

```
    ((*mSegments)[i].refPos < endOf1stHalf))))
2224
2225                &&
2226                (((*mSegments)[i].refPos < endOf1stHalf &&
2227                  ((*mSegments)[i].degPos - (*mSegments)[i].refPos < lowerOuterFence_1stHalf ||
2228                   (*mSegments)[i].degPos - (*mSegments)[i].refPos > upperOuterFence_1stHalf))
2229                 ||
2230                 ((*mSegments)[i].refPos >= endOf1stHalf &&
2231                  ((*mSegments)[i].degPos - (*mSegments)[i].refPos < lowerOuterFence_2ndHalf ||
2232                   (*mSegments)[i].degPos - (*mSegments)[i].refPos > upperOuterFence_2ndHalf))
2233                 ||
2234                 (badSegFound && (*mSegments)[i].segType == TA_SEG_MISSING))
2235                &&
2236                (((*mSegments)[i].segType != TA_SEG_MATCHED && (*mSegments)[i].segType !=
    TA_SEG_GUESSED)
2237                 ||
2238
2239                 CorrCheck(&(*mSegments)[i], fRefNorm, lRefNorm, fDegNorm, lDegNorm,
    numPrevBadActSegs)))
2240
2241            {
2242
2243                if (badSegFound)
2244                    lastBadSeg  = i
2245                else
2246                {
2247                    firstBadSeg = i
2248                    lastBadSeg  = i
2249                    badSegFound = true
2250                }
2251            }
2252            else
2253            {
2254                if (badSegFound)
2255                {
2256                    if (SegListCopy == NULL)
2257                        SegListCopy = new TA_SegList(*mSegments)
2258
2259                    nextGoodSeg = i
2260
2261                    int firstSpeechIdx = firstBadSeg, lastSpeechIdx = lastBadSeg
2262                    for (  firstSpeechIdx < (int)mSegments->size() &&
2263                         mSegments->at(firstSpeechIdx).segType == TA_SEG_PAUSE
2264                         firstSpeechIdx++)
2265                    for (  lastSpeechIdx >= firstSpeechIdx &&
2266                         mSegments->at(lastSpeechIdx ).segType == TA_SEG_PAUSE
2267                         lastSpeechIdx--)
2268                    int consecBadSpeechLen = mSegments->at(lastSpeechIdx).refPos +
2269                                             mSegments->at(lastSpeechIdx).segLen -
2270                                             mSegments->at(firstSpeechIdx).refPos
2271                    if (consecBadSpeechLen >= round(1.0f * TA_SAMPLING_RATE))
2272                    {
2273                        TA_SegList badSegs
2274                        badSegs.insert(badSegs.begin(),
2275                            mSegments->getIterator(firstBadSeg),
2276                            mSegments->getIterator(lastBadSeg+1))
2277                        XFLOAT longGroupMatchQual = (XFLOAT)-1.0
2278                        OPTTRY
2279                        {
2280                            longGroupMatchQual = CalcMatchQuality(&badSegs)
2281                            if (longGroupMatchQual > 0.75f)
2282                            {
2283                                prevGoodSeg       = i
2284                                badSegFound       = false
2285                                firstBadSeg       = lastBadSeg = -1
2286                                numPrevBadActSegs = 0
2287                                continue
2288                            }
```

```
2289                        }
2290                        OPTCATCH((...)){}
2291                    }
2292
2293                    mExtremeMatchFound = true
2294
2295                    int j = firstBadSeg-1
2296                    for (  j >= 0 &&
2297                         ((*mSegments)[j].segType == TA_SEG_MISSING ||
2298                          (*mSegments)[j].segType == TA_SEG_PAUSE)
2299                         j--)
2300                    if (j < firstBadSeg-1 && j >= 0 &&
2301                        (*mSegments)[j].segType != TA_SEG_MISSING &&
2302                        (*mSegments)[j].segType != TA_SEG_PAUSE)
2303                    {
2304                        prevGoodSeg = j
2305
2306                        if (((*mSegments)[j+1].refPos < endOf1stHalf) ==
    ((*mSegments)[firstBadSeg].refPos < endOf1stHalf) &&
2307                            (*mSegments)[j+1].refPos != endOf1stHalf)
2308
2309                            firstBadSeg = j+1
2310                    }
2311
2312                    for (j = lastBadSeg+1  j < (int)mSegments->size() &&
2313                         ((*mSegments)[j].segType == TA_SEG_MISSING ||
2314                          (*mSegments)[j].segType == TA_SEG_PAUSE)
2315                         j++)
2316                    if (j > lastBadSeg+1 && j < (int)mSegments->size() &&
2317                        (*mSegments)[j].segType != TA_SEG_MISSING &&
2318                        (*mSegments)[j].segType != TA_SEG_PAUSE)
2319                    {
2320                        nextGoodSeg = j
2321
2322                        if (((*mSegments)[j-1].refPos < endOf1stHalf) ==
    ((*mSegments)[firstBadSeg].refPos < endOf1stHalf) &&
2323                            (*mSegments)[j-1].refPos != endOf1stHalf)
2324
2325                            lastBadSeg  = j-1
2326                    }
2327
2328                    bool doFixPausesAtStart = false, doFixPausesAtEnd = false
2329                    CheckForSurroundingPauses(firstBadSeg, lastBadSeg,
2330                                              doFixPausesAtStart, doFixPausesAtEnd)
2331
2332                    if (lastBadSeg < firstBadSeg)
2333                    {
2334                        prevGoodSeg       = i
2335                        badSegFound       = false
2336                        firstBadSeg       = lastBadSeg = -1
2337                        numPrevBadActSegs = 0
2338                        continue
2339                    }
2340
2341                    int iMaxDegLen, iDegStart, uttStart, uttLen
2342                    GetSignalLengths(prevGoodSeg, firstBadSeg, lastBadSeg, nextGoodSeg,
2343                                     uttStart, uttLen, iDegStart, iMaxDegLen, lDegNorm)
2344
2345                    if (iDegStart < 0 || iMaxDegLen < 1 || uttStart < 0 || uttLen < 1)
2346                    {
2347                        prevGoodSeg       = i
2348                        badSegFound       = false
2349                        firstBadSeg       = lastBadSeg = -1
2350                        numPrevBadActSegs = 0
2351                        continue
2352                    }
2353
2354                    TA_SegList deletedSegments
```

```
                    deletedSegments.insert(deletedSegments.begin(),
                                           mSegments->getIterator(firstBadSeg),
                                           mSegments->getIterator(lastBadSeg+1))

                    //Remove the bad segments.
                    mSegments->erase(mSegments->getIterator(firstBadSeg),
                                     mSegments->getIterator(lastBadSeg+1))

                    int guessedShift, GBS_iMaxDegLen = iMaxDegLen
                    if (iDegStart + iMaxDegLen == lDegNorm)
                        GBS_iMaxDegLen = (((iMaxDegLen) > (uttLen)) ? (iMaxDegLen) : (uttLen))
                    GuessBestShift(uttStart, iDegStart, GBS_iMaxDegLen, fDegNorm, lDegNorm,
                                   uttLen, guessedShift, NULL, endOf1stHalf)

                    bool newShiftIsSameAsOld = true
                    for (int k = 0  k < (int)deletedSegments.size()  k++)
                        if (deletedSegments[k].segType != TA_SEG_MISSING &&
                            deletedSegments[k].segType != TA_SEG_PAUSE &&
                            deletedSegments[k].degPos - deletedSegments[k].refPos != guessedShift)
                            newShiftIsSameAsOld &= false

                    if (newShiftIsSameAsOld ||
                        uttStart+uttLen + guessedShift <=  0 ||
                        uttStart        + guessedShift >= lDegNorm)
                    {
                        mSegments->insert(mSegments->findInsLoc(deletedSegments[0].refPos),
                                          deletedSegments.begin(), deletedSegments.end())
                        prevGoodSeg       = i
                        badSegFound       = false
                        firstBadSeg       = lastBadSeg = -1
                        numPrevBadActSegs = 0
                        continue
                    }

                    //Insert fixed segment.
                    WriteMatchedSegment(uttStart, uttLen, guessedShift,
                                        iDegStart, iMaxDegLen, lDegNorm, false,
                                        NULL, 0, 0, 0.0f)

                    if (uttLen > 0)
                    {
                        mSegments->findInsLoc(uttStart)->segType = TA_SEG_GUESSED
                        mSegments->findInsLoc(uttStart)->dontMergeWithOthers = true
                        mSegments->findInsLoc(uttStart)->reliability = 0.0f
                    }

                    FixPauses(doFixPausesAtStart, doFixPausesAtEnd, lDegNorm)

                    //Move indices after modification of segments list.
                    prevGoodSeg       = mSegments->findInsLocIdx(uttStart)
                    i                 = prevGoodSeg
                    badSegFound       = false
                    firstBadSeg       = lastBadSeg = -1
                    numPrevBadActSegs = 0
                }
                else
                {
                    prevGoodSeg       = i
                    numPrevBadActSegs = 0
                }
            }
        }
        delete SegListCopy
        SegListCopy = NULL
        return
    }
    OPTCATCH((string errorMsg))
    {
```

```
2423         mSegments->swap(*SegListCopy)
2424         delete SegListCopy
2425         SegListCopy = NULL
2426 
2427         return
2428 
2429     }
2430     OPTCATCH((...))
2431     {
2432         mSegments->swap(*SegListCopy)
2433         delete SegListCopy
2434         SegListCopy = NULL
2435 
2436         return
2437 
2438     }
2439 }
2440 
2441 void SQTimeAlignment::EstimateShiftsRange(int const &numMatchedSegs, int lRefNorm, int
    &endOf1stHalf,
2442                                           int &lowerOuterFence_1stHalf, int
    &lowerOuterFence_2ndHalf,
2443                                           int &upperOuterFence_1stHalf, int
    &upperOuterFence_2ndHalf)
2444 {
2445     long *shifts_1stHalf[2] = {NULL, NULL}, *shifts[2] = {NULL, NULL},
2446          *shifts_2ndHalf[2] = {NULL, NULL}
2447     int const ESR_MIN_FENCE_WIDTH
2448         = (((round(0.032f * TA_SAMPLING_RATE)) < (round(FRAME_LEN * TA_SAMPLING_RATE))) ?
    (round(0.032f * TA_SAMPLING_RATE)) : (round(FRAME_LEN * TA_SAMPLING_RATE)))
2449     int const ESR_MIN_MID_PAUSE_LEN
2450         = round(0.33f * TA_SAMPLING_RATE)
2451 
2452     OPTTRY
2453     {
2454         if (numMatchedSegs < 1 || lRefNorm < 1)
2455             OPTTHROW( string("Invalid input arguments."))
2456 
2457         shifts[0] = (long*)matMalloc(numMatchedSegs * sizeof(long))
2458         shifts[1] = (long*)matMalloc(numMatchedSegs * sizeof(long))
2459         int j = 0, i, counter, q[3], qNum, iqr,
2460             totMatchedLen = 0, totMatchedLen_1stHalf = 0, totMatchedLen_2ndHalf = 0,
2461             lastMatchedSegOf1stHalf = -1,
2462             longestMidPauseLen = ESR_MIN_MID_PAUSE_LEN
2463         endOf1stHalf = lRefNorm
2464 
2465         //Copy the shifts of matched segments to a shifts array,
2466         //and try to detect the middle pause between two sentences, if present.
2467         for (i = 0  i < (int)mSegments->size()  i++)
2468         {
2469             if ((*mSegments)[i].segType == TA_SEG_MATCHED)
2470             {
2471                 shifts[0][j]   = (long)((*mSegments)[i].degPos - (*mSegments)[i].refPos)
2472                 shifts[1][j]   = (*mSegments)[i].segLen
2473                 totMatchedLen += (*mSegments)[i].segLen
2474                 j++
2475             }
2476             if (j > 0 && j < numMatchedSegs && (*mSegments)[i].segType == TA_SEG_PAUSE &&
2477                 (*mSegments)[i].segLen > longestMidPauseLen)
2478             {
2479                 longestMidPauseLen      = (*mSegments)[i].segLen
2480                 lastMatchedSegOf1stHalf = j-1
2481                 endOf1stHalf            = (*mSegments)[i].refPos
2482                 totMatchedLen_1stHalf   = totMatchedLen
2483             }
2484         }
2485         if (lastMatchedSegOf1stHalf < 0)
2486         {
```

```
2487            XFLOAT matchedLenNorm = totMatchedLen / (XFLOAT)PI, pauseFac, maxPauseFac = 0.0f
2488            int sumMatchedLen = 0, bestPauseIdx = -1, pauseLen
2489            for (i = 0, j = 0  i < (int)mSegments->size()  i++)
2490            {
2491                if ((*mSegments)[i].segType != TA_SEG_MATCHED &&
2492                    (*mSegments)[i].segType != TA_SEG_PAUSE)
2493                    continue
2494                if ((*mSegments)[i].segType == TA_SEG_MATCHED)
2495                {
2496                    sumMatchedLen += (*mSegments)[i].segLen
2497                    j++
2498                }
2499                else
2500                {
2501                    pauseLen = (*mSegments)[i].segLen
2502                    pauseFac = sin(sumMatchedLen / matchedLenNorm) * sqrt((XFLOAT)pauseLen)
2503                    if (pauseFac > maxPauseFac && sumMatchedLen < totMatchedLen)
2504                    {
2505                        longestMidPauseLen      = pauseLen
2506                        lastMatchedSegOf1stHalf = j-1
2507                        endOf1stHalf            = (*mSegments)[i].refPos
2508                        totMatchedLen_1stHalf   = sumMatchedLen
2509                        maxPauseFac             = pauseFac
2510                    }
2511                }
2512            }
2513        }
2514        totMatchedLen_2ndHalf = totMatchedLen - totMatchedLen_1stHalf
2515
2516        //Determine the fences for extreme outliers by computing the quartiles
2517        int q2_1stHalf, q2_2ndHalf
2518        if (lastMatchedSegOf1stHalf == -1)
2519        {
2520            sortTwoVectors(shifts[0], shifts[1], 1, numMatchedSegs)
2521            for (i = 0, counter = 0, qNum = 0  i < numMatchedSegs && qNum < 3  i++)
2522            {
2523                counter += shifts[1][i]
2524                if (counter >= round((qNum+1)*0.25f*totMatchedLen))
2525                    q[qNum++] = shifts[0][i]
2526            }
2527            if (qNum != 3)
2528            {
2529                if (qNum < 1)
2530                    OPTTHROW( string("Could not compute quartiles of shifts of matched segments."))
2531                else
2532                    for (i = 3-1  i >= qNum  i--)
2533                        q[i] = q[qNum-1]
2534            }
2535
2536            iqr = q[3-1] - q[1-1]
2537            q2_1stHalf = q2_2ndHalf = q[2-1]
2538            lowerOuterFence_1stHalf = lowerOuterFence_2ndHalf = q[1-1] - 3*iqr
2539            upperOuterFence_1stHalf = upperOuterFence_2ndHalf = q[3-1] + 3*iqr
2540        }
2541        else
2542        {
2543            shifts_1stHalf[0] = (long*)matMalloc((((lastMatchedSegOf1stHalf+1) > (1)) ?
    (lastMatchedSegOf1stHalf+1) : (1)) * sizeof(long))
2544            shifts_1stHalf[1] = (long*)matMalloc((((lastMatchedSegOf1stHalf+1) > (1)) ?
    (lastMatchedSegOf1stHalf+1) : (1)) * sizeof(long))
2545            shifts_2ndHalf[0] = (long*)matMalloc((((numMatchedSegs - (lastMatchedSegOf1stHalf+1)) >
    (1)) ? (numMatchedSegs - (lastMatchedSegOf1stHalf+1)) : (1)) * sizeof(long))
2546            shifts_2ndHalf[1] = (long*)matMalloc((((numMatchedSegs - (lastMatchedSegOf1stHalf+1)) >
    (1)) ? (numMatchedSegs - (lastMatchedSegOf1stHalf+1)) : (1)) * sizeof(long))
2547            ivmov(shifts[0], shifts_1stHalf[0], (((lastMatchedSegOf1stHalf+1) > (1)) ?
    (lastMatchedSegOf1stHalf+1) : (1)))
2548            ivmov(shifts[1], shifts_1stHalf[1], (((lastMatchedSegOf1stHalf+1) > (1)) ?
    (lastMatchedSegOf1stHalf+1) : (1)))
```

```
2549            ivmov(shifts[0]+lastMatchedSegOf1stHalf+1, shifts_2ndHalf[0], (((numMatchedSegs -
    (lastMatchedSegOf1stHalf+1)) > (1)) ? (numMatchedSegs - (lastMatchedSegOf1stHalf+1)) :
    (1)))
2550            ivmov(shifts[1]+lastMatchedSegOf1stHalf+1, shifts_2ndHalf[1], (((numMatchedSegs -
    (lastMatchedSegOf1stHalf+1)) > (1)) ? (numMatchedSegs - (lastMatchedSegOf1stHalf+1)) :
    (1)))
2551
2552            sortTwoVectors(shifts_1stHalf[0], shifts_1stHalf[1], 1, lastMatchedSegOf1stHalf+1)
2553            for (i = 0, counter = 0, qNum = 0  i < (((lastMatchedSegOf1stHalf+1) > (1)) ?
    (lastMatchedSegOf1stHalf+1) : (1)) && qNum < 3  i++)
2554            {
2555                counter += shifts_1stHalf[1][i]
2556                if (counter >= round((qNum+1)*0.25f*totMatchedLen_1stHalf))
2557                    q[qNum++] = shifts_1stHalf[0][i]
2558            }
2559            if (qNum != 3)
2560            {
2561                if (qNum < 1)
2562                    OPTTHROW( string("Could not compute quartiles of shifts of matched segments."))
2563                else
2564                    for (i = 3-1  i >= qNum  i--)
2565                        q[i] = q[qNum-1]
2566            }
2567
2568            iqr = q[3-1] - q[1-1]
2569            q2_1stHalf = q[2-1]
2570            lowerOuterFence_1stHalf = q[1-1] - 3*iqr,
2571            upperOuterFence_1stHalf = q[3-1] + 3*iqr
2572
2573            sortTwoVectors(shifts_2ndHalf[0], shifts_2ndHalf[1], 1,
    numMatchedSegs-(lastMatchedSegOf1stHalf+1))
2574            for (i = 0, counter = 0, qNum = 0  i < (((numMatchedSegs - (lastMatchedSegOf1stHalf+1))
    > (1)) ? (numMatchedSegs - (lastMatchedSegOf1stHalf+1)) : (1)) && qNum < 3  i++)
2575            {
2576                counter += shifts_2ndHalf[1][i]
2577                if (counter >= round((qNum+1)*0.25f*totMatchedLen_2ndHalf))
2578                    q[qNum++] = shifts_2ndHalf[0][i]
2579            }
2580            if (qNum != 3)
2581            {
2582                if (qNum < 1)
2583                    OPTTHROW( string("Could not compute quartiles of shifts of matched segments."))
2584                else
2585                    for (i = 3-1  i >= qNum  i--)
2586                        q[i] = q[qNum-1]
2587            }
2588
2589            iqr = q[3-1] - q[1-1]
2590            q2_2ndHalf = q[2-1]
2591            lowerOuterFence_2ndHalf = q[1-1] - 3*iqr,
2592            upperOuterFence_2ndHalf = q[3-1] + 3*iqr
2593        }
2594
2595        if (upperOuterFence_1stHalf - lowerOuterFence_1stHalf < ESR_MIN_FENCE_WIDTH)
2596        {
2597            lowerOuterFence_1stHalf = q2_1stHalf - ESR_MIN_FENCE_WIDTH/2
2598            upperOuterFence_1stHalf = q2_1stHalf + ESR_MIN_FENCE_WIDTH/2
2599        }
2600        if (upperOuterFence_2ndHalf - lowerOuterFence_2ndHalf < ESR_MIN_FENCE_WIDTH)
2601        {
2602            lowerOuterFence_2ndHalf = q2_2ndHalf - ESR_MIN_FENCE_WIDTH/2
2603            upperOuterFence_2ndHalf = q2_2ndHalf + ESR_MIN_FENCE_WIDTH/2
2604        }
2605
2606        matFree(shifts[0])
2607        matFree(shifts[1])
2608        matFree(shifts_1stHalf[0])
2609        matFree(shifts_1stHalf[1])
```

```
2610         matFree(shifts_2ndHalf[0])
2611         matFree(shifts_2ndHalf[1])
2612         shifts[0] = shifts[1] = shifts_1stHalf[0] = shifts_1stHalf[1] = shifts_2ndHalf[0] =
2613             shifts_2ndHalf[1] = NULL
2614     }
2615     OPTCATCH((string errorMsg))
2616     {
2617         matFree(shifts[0])
2618         matFree(shifts[1])
2619         matFree(shifts_1stHalf[0])
2620         matFree(shifts_1stHalf[1])
2621         matFree(shifts_2ndHalf[0])
2622         matFree(shifts_2ndHalf[1])
2623         shifts[0] = shifts[1] = shifts_1stHalf[0] = shifts_1stHalf[1] = shifts_2ndHalf[0] =
2624             shifts_2ndHalf[1] = NULL
2625         OPTTHROW( string("ERROR in EstimateShiftsRange: " + errorMsg + "\n"))
2626     }
2627     OPTCATCH((...))
2628     {
2629         matFree(shifts[0])
2630         matFree(shifts[1])
2631         matFree(shifts_1stHalf[0])
2632         matFree(shifts_1stHalf[1])
2633         matFree(shifts_2ndHalf[0])
2634         matFree(shifts_2ndHalf[1])
2635         shifts[0] = shifts[1] = shifts_1stHalf[0] = shifts_1stHalf[1] = shifts_2ndHalf[0] =
2636             shifts_2ndHalf[1] = NULL
2637         OPTTHROW( string("ERROR in EstimateShiftsRange: Unknown error.\n"))
2638     }
2639 }
2640
2641 bool SQTimeAlignment::CorrCheck(TA_segStruct const *curSeg,
2642                                 XFLOAT const *fRefNorm, int lRefNorm,
2643                                 XFLOAT const *fDegNorm, int lDegNorm,
2644                                 int &numPrevBadActSegs)
2645 {
2646     if (curSeg == NULL || curSeg->refPos < 0 || curSeg->degPos < 0 ||
2647         curSeg->segLen < 1 ||
2648         (curSeg->segType != TA_SEG_MATCHED && curSeg->segType != TA_SEG_GUESSED) ||
2649         fRefNorm == NULL || fDegNorm == NULL || lRefNorm < curSeg->refPos + curSeg->segLen ||
2650         lDegNorm < curSeg->degPos + curSeg->segLen)
2651         OPTTHROW( string("ERROR in CorrCheck: Invalid input arguments.\n"))
2652
2653     XFLOAT const *refSpan = fRefNorm + curSeg->refPos
2654     XFLOAT const *degSpan = fDegNorm + curSeg->degPos
2655     int    spanLen = curSeg->segLen
2656
2657     XFLOAT fRefAutocorr, fDegAutocorr, fDivisor
2658     svesq(refSpan, &fRefAutocorr, spanLen)
2659     svesq(degSpan, &fDegAutocorr, spanLen)
2660     fDivisor = sqrt(fRefAutocorr * fDegAutocorr)
2661
2662     if (AlmostEqualUlpsFinal((float)fRefAutocorr, (float)0.0) &&
    AlmostEqualUlpsFinal((float)fDegAutocorr, (float)0.0))
2663         return false
2664     else if (AlmostEqualUlpsFinal((float)fDivisor, (float)0.0))
2665         return true
2666
2667     XFLOAT crossCorr
2668     dotpr(refSpan, degSpan, &crossCorr, spanLen)
2669     crossCorr = fabs(crossCorr / fDivisor)
2670
2671     XFLOAT corrThr = pow(0.6, 1.0 / (XFLOAT)limit(numPrevBadActSegs, 1, 4))
2672
2673     if (crossCorr < corrThr)
2674     {
2675         XFLOAT refPow, degPow
2676         rmvesq(mRef->Data() + curSeg->refPos, &refPow, spanLen)
```

```
2677         refPow /= mRef->MaxAmplitude()
2678         rmvesq(mDeg->Data() + curSeg->degPos, &degPow, spanLen)
2679         degPow /= mDeg->MaxAmplitude()
2680 
2681         if (rmsTodB(refPow) > mRef->CurrentASL() - 24.0 &&
2682             rmsTodB(degPow) > mDeg->CurrentASL() - 24.0)
2683             numPrevBadActSegs++
2684     }
2685 
2686     return crossCorr < corrThr
2687 }
2688 
2689 void SQTimeAlignment::CheckForSurroundingPauses(int& firstBadSeg, int& lastBadSeg,
2690                                                 bool& doFixPausesAtStart,
2691                                                 bool& doFixPausesAtEnd)
2692 {
2693     if (firstBadSeg < 0 || lastBadSeg >= (int)mSegments->size() || firstBadSeg > lastBadSeg)
2694         OPTTHROW( string("ERROR in CheckForSurroundingPauses: Invalid input segment indices.\n"))
2695 
2696     int j = lastBadSeg+1
2697     for (  j < (int)mSegments->size() &&
2698          ((*mSegments)[j].segType == TA_SEG_PAUSE ||
2699           (*mSegments)[j].segType == TA_SEG_MISSING)
2700          j++)
2701 
2702     //Only pause/missing segments following, the first following segment being a pause or missing?
2703     if (j == (int)mSegments->size() && lastBadSeg+1 < (int)mSegments->size() &&
2704         ((*mSegments)[lastBadSeg+1].segType == TA_SEG_PAUSE ||
2705          (*mSegments)[lastBadSeg+1].segType == TA_SEG_MISSING))
2706     {
2707         for (j--  j >= firstBadSeg &&
2708              ((*mSegments)[j].segType == TA_SEG_PAUSE ||
2709               (*mSegments)[j].segType == TA_SEG_MISSING)
2710              j--)
2711         lastBadSeg = j
2712         doFixPausesAtEnd = true
2713     }
2714 
2715     for (j = firstBadSeg-1  j >= 0 &&
2716          ((*mSegments)[j].segType == TA_SEG_PAUSE ||
2717           (*mSegments)[j].segType == TA_SEG_MISSING)
2718          j--)
2719 
2720     //Only pause/missing segments preceding, the first preceding segment being a pause or missing?
2721     if (j < 0 && firstBadSeg-1 >= 0 &&
2722         ((*mSegments)[firstBadSeg-1].segType == TA_SEG_PAUSE ||
2723          (*mSegments)[firstBadSeg-1].segType == TA_SEG_MISSING))
2724     {
2725         for (j++  j < (int)mSegments->size() &&
2726              ((*mSegments)[j].segType == TA_SEG_PAUSE ||
2727               (*mSegments)[j].segType == TA_SEG_MISSING)
2728              j++)
2729         firstBadSeg = j
2730         doFixPausesAtStart = true
2731     }
2732 }
2733 
2734 void SQTimeAlignment::GetSignalLengths(int prevGoodSeg, int firstBadSeg,
2735                                        int lastBadSeg, int nextGoodSeg,
2736                                        int& uttStart, int& uttLen, int& iDegStart,
2737                                        int& iMaxDegLen, int lDegNorm)
2738 {
2739     if (prevGoodSeg > nextGoodSeg || firstBadSeg < 0 || firstBadSeg > lastBadSeg ||
2740         lastBadSeg >= (int)mSegments->size())
2741         OPTTHROW( string("ERROR in GetSignalLengths: Invalid input segment indices.\n"))
2742 
2743     if (nextGoodSeg == (int)mSegments->size() ||
2744         (*mSegments)[nextGoodSeg].segType == TA_SEG_PAUSE ||
```

```
2745        (*mSegments)[nextGoodSeg].segType == TA_SEG_MISSING)
2746    {
2747        if (prevGoodSeg < 0)
2748
2749        return
2750
2751        iDegStart  = (*mSegments)[prevGoodSeg].degPos + (*mSegments)[prevGoodSeg].segLen
2752        iMaxDegLen = lDegNorm - iDegStart
2753        uttStart   = ((((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) >
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen)) ?
    ((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) :
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen))
2754        uttLen     = ((*mSegments)[lastBadSeg].refPos + (*mSegments)[lastBadSeg].segLen) - uttStart
2755    }
2756    else if (prevGoodSeg < nextGoodSeg && prevGoodSeg >= 0)
2757    {
2758        if ((*mSegments)[prevGoodSeg].segType != TA_SEG_MISSING)
2759        {
2760            iMaxDegLen = (*mSegments)[nextGoodSeg].degPos - ((*mSegments)[prevGoodSeg].degPos +
    (*mSegments)[prevGoodSeg].segLen)
2761            iDegStart  = (*mSegments)[prevGoodSeg].degPos + (*mSegments)[prevGoodSeg].segLen
2762            uttStart   = ((((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) >
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen)) ?
    ((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) :
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen))
2763            uttLen     = ((*mSegments)[lastBadSeg].refPos + (*mSegments)[lastBadSeg].segLen) -
    uttStart
2764        }
2765        else
2766        {
2767            iDegStart  = prevGoodSeg == 0 ? 0 : ((*mSegments)[prevGoodSeg-1].degPos +
    (*mSegments)[prevGoodSeg-1].segLen)
2768            iMaxDegLen = (*mSegments)[nextGoodSeg].degPos - iDegStart
2769            uttStart   = ((((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) >
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen)) ?
    ((*mSegments)[prevGoodSeg].refPos + (*mSegments)[prevGoodSeg].segLen) :
    ((*mSegments)[firstBadSeg-1].refPos + (*mSegments)[firstBadSeg-1].segLen))
2770            uttLen     = ((*mSegments)[lastBadSeg].refPos + (*mSegments)[lastBadSeg].segLen) -
    uttStart
2771        }
2772    }
2773    else
2774    {
2775        iMaxDegLen = (*mSegments)[nextGoodSeg].degPos
2776        iDegStart  = 0
2777        uttStart   = 0
2778        uttLen     = ((*mSegments)[lastBadSeg].refPos + (*mSegments)[lastBadSeg].segLen)
2779    }
2780 }
2781
2782 void SQTimeAlignment::FixPauses(bool doFixPausesAtStart, bool doFixPausesAtEnd, int lDegNorm)
2783 {
2784    if (doFixPausesAtEnd)
2785    {
2786        int  pauseStart      = (int)mSegments->size()-1
2787        int  totPauseLen     = 0
2788        int  minDegPausePos  = lDegNorm
2789        bool onlyMissingSegs = true
2790        while (pauseStart >= 0 &&
2791               ((*mSegments)[pauseStart].segType == TA_SEG_PAUSE ||
2792                (*mSegments)[pauseStart].segType == TA_SEG_MISSING))
2793        {
2794            totPauseLen     += (*mSegments)[pauseStart].segLen
2795            onlyMissingSegs &= (*mSegments)[pauseStart].segType == TA_SEG_MISSING
2796            minDegPausePos   = (((minDegPausePos) < ((*mSegments)[pauseStart].degPos)) ?
    (minDegPausePos) : ((*mSegments)[pauseStart].degPos))
2797            pauseStart--
2798        }
```

```
2799
2800        int lastUttSeg = pauseStart
2801        pauseStart++
2802        int pauseEndPos = (*mSegments)[pauseStart].degPos + totPauseLen - 1
2803
2804        if (!onlyMissingSegs && lastUttSeg >= 0 && totPauseLen > 0 && pauseEndPos < lDegNorm-1 &&
2805            (*mSegments)[lastUttSeg].degPos + (*mSegments)[lastUttSeg].segLen >
    (*mSegments)[pauseStart].degPos)
2806        {
2807            int lastUttEndPos = (*mSegments)[lastUttSeg].degPos + (*mSegments)[lastUttSeg].segLen -
    1
2808            int shiftLen      =
2809
2810            mSegments->erase(mSegments->getIterator(pauseStart+1),
2811                             mSegments->end())
2812
2813            (*mSegments)[pauseStart].degPos += shiftLen
2814            (*mSegments)[pauseStart].segLen  = totPauseLen
2815        }
2816        else if (onlyMissingSegs && lastUttSeg >= 0 && totPauseLen > 0)
2817        {
2818            (*mSegments)[pauseStart].degPos = minDegPausePos
2819            (*mSegments)[pauseStart].segLen = totPauseLen
2820
2821            mSegments->erase(mSegments->getIterator(pauseStart+1),
2822                             mSegments->end())
2823        }
2824    }
2825
2826    if (doFixPausesAtStart)
2827    {
2828        int  pauseEnd        = 0
2829        int  totPauseLen     = 0
2830        int  minDegPausePos  = lDegNorm
2831        bool onlyMissingSegs = true
2832        while (pauseEnd < (int)mSegments->size() &&
2833               ((*mSegments)[pauseEnd].segType == TA_SEG_PAUSE ||
2834                (*mSegments)[pauseEnd].segType == TA_SEG_MISSING))
2835        {
2836            totPauseLen     += (*mSegments)[pauseEnd].segLen
2837            onlyMissingSegs &= (*mSegments)[pauseEnd].segType == TA_SEG_MISSING
2838            minDegPausePos   = (((minDegPausePos) < ((*mSegments)[pauseEnd].degPos)) ?
    (minDegPausePos) : ((*mSegments)[pauseEnd].degPos))
2839            pauseEnd++
2840        }
2841
2842        int firstUttSeg = pauseEnd
2843        pauseEnd--
2844        int pauseStartPos = totPauseLen - ((*mSegments)[pauseEnd].degPos +
    (*mSegments)[pauseEnd].segLen)
2845
2846        if (!onlyMissingSegs && firstUttSeg < (int)mSegments->size() && totPauseLen > 0 &&
    pauseStartPos > 0 &&
2847            (*mSegments)[pauseEnd].degPos + (*mSegments)[pauseEnd].segLen >
    (*mSegments)[firstUttSeg].degPos)
2848        {
2849            int pauseEndPos = (*mSegments)[pauseEnd].degPos + (*mSegments)[pauseEnd].segLen - 1
2850            int shiftLen    =
2851
2852            (*mSegments)[pauseEnd].degPos -= shiftLen
2853            (*mSegments)[pauseEnd].segLen  = totPauseLen
2854
2855            mSegments->erase(mSegments->begin(),
2856                             mSegments->getIterator(pauseEnd))
2857        }
2858        else if (onlyMissingSegs && firstUttSeg < (int)mSegments->size() && totPauseLen > 0)
2859        {
2860            (*mSegments)[pauseEnd].refPos = (*mSegments)[0].refPos
```

```
2861            (*mSegments)[pauseEnd].degPos = minDegPausePos
2862            (*mSegments)[pauseEnd].segLen = totPauseLen
2863
2864            mSegments->erase(mSegments->begin(),
2865                             mSegments->getIterator(pauseEnd))
2866        }
2867    }
2868 }
2869
2870 void SQTimeAlignment::FixIncorrectMatches()
2871 {
2872     OPTTRY
2873     {
2874         if (mSegments == NULL || mSegments->size() == 0)
2875             OPTTHROW( string("Invalid input parameters."))
2876         if (mSegments->size() <= 4)
2877             return
2878
2879         int const FIM_MIN_FIXABLE_GAP_LEN =
2880             round(10e-3f * mCurSegmentRate)
2881         int const FIM_MAX_FIXABLE_GAP_LEN =
2882             round(50e-3f * mCurSegmentRate)
2883
2884         TA_SegList::iterator seg = mSegments->end(), matchedSeg = mSegments->end()
2885         int listLen = (int)mSegments->size(), degGapLen = -1
2886
2887         for (int i = 0  i < listLen  i++)
2888         {
2889             if ((seg = mSegments->getIterator(i))->segType != TA_SEG_MISSING ||
2890                 seg->segLen < FIM_MIN_FIXABLE_GAP_LEN || seg->segLen > FIM_MAX_FIXABLE_GAP_LEN)
2891                 continue
2892
2893             if (i+2 < listLen && (seg+1)->segType == TA_SEG_MATCHED &&
2894                 (seg+2)->degPos > (seg+1)->degPos + (seg+1)->segLen &&
2895                 (seg+2)->segType != TA_SEG_MISSING)
2896             {
2897                 if ((i+3 >= listLen || (seg+3)->segType != TA_SEG_MISSING) &&
2898                     (i-2 <  0       || (seg-2)->degPos + (seg-2)->segLen == (seg-1)->degPos))
2899                 {
2900                     matchedSeg = seg+1
2901                     degGapLen  = (seg+2)->degPos - (matchedSeg->degPos + matchedSeg->segLen)
2902                 }
2903             }
2904
2905             else if (i > 2 && (seg-1)->segType == TA_SEG_MATCHED &&
2906                      (seg-2)->degPos + (seg-2)->segLen < (seg-1)->degPos &&
2907                      (seg-2)->segType != TA_SEG_MISSING)
2908             {
2909                 if ((i-3 <  0       || (seg-3)->segType != TA_SEG_MISSING) &&
2910                     (i+2 >= listLen || (seg+2)->degPos == (seg+1)->degPos + (seg+1)->segLen))
2911                 {
2912                     matchedSeg = seg-1
2913                     degGapLen  = matchedSeg->degPos - ((seg-2)->degPos + (seg-2)->segLen)
2914                 }
2915             }
2916
2917             if (matchedSeg != mSegments->end() &&
2918                 degGapLen >= FIM_MIN_FIXABLE_GAP_LEN &&
2919                 degGapLen <= FIM_MAX_FIXABLE_GAP_LEN)
2920             {
2921                 matchedSeg->segType     = TA_SEG_GUESSED
2922                 matchedSeg->reliability = 0.0f
2923             }
2924
2925             matchedSeg = mSegments->end()
2926             degGapLen  = -1
2927         }
2928     }
```

```
2929      OPTCATCH((string errorMsg))
2930      {
2931          OPTTHROW( string("ERROR in FixIncorrectMatches: " + errorMsg + "\n"))
2932      }
2933      OPTCATCH((...))
2934      {
2935          OPTTHROW( string("ERROR in FixIncorrectMatches: Unknown error.\n"))
2936      }
2937  }
2938
2939  void SQTimeAlignment::MergeConsecutiveSegments()
2940  {
2941      if (mSegments == NULL || mSegments->size() == 0)
2942          OPTTHROW( string("ERROR in MergeConsecutiveSegments: No segments list.\n"))
2943
2944      //Merge consecutive segments of same type with identical shift
2945      for (int i = 0  i < (int)mSegments->size()-1  i++)
2946      {
2947          int   lastConsecSeg
2948          int   totConsecSegLen = 0,
2949                curShift        = (*mSegments)[i].degPos - (*mSegments)[i].refPos
2950          XFLOAT avgReliability  = (XFLOAT)0.0
2951
2952          for (lastConsecSeg  = i+1
2953               lastConsecSeg < (int)mSegments->size() &&
2954               (*mSegments)[lastConsecSeg].segType == (*mSegments)[i].segType &&
2955               ((*mSegments)[i].segType == TA_SEG_MISSING ||
2956                (*mSegments)[lastConsecSeg].degPos - (*mSegments)[lastConsecSeg].refPos == curShift)
2957               lastConsecSeg++)
2958          {
2959              totConsecSegLen += (*mSegments)[lastConsecSeg].segLen
2960              avgReliability  += (*mSegments)[lastConsecSeg].reliability
2961          }
2962
2963          //Found 1 or more consecutive segs w/ same shift and same type?
2964          if (totConsecSegLen > 0)
2965          {
2966              mSegments->erase(mSegments->getIterator(i+1),
2967                               mSegments->getIterator(lastConsecSeg))
2968
2969              //Calculate new shift value for consecutive TA_SEG_MISSING segments
2970              if ((*mSegments)[i].segType == TA_SEG_MISSING)
2971              {
2972                  int degStart = (*mSegments)[i].degPos + (*mSegments)[i].segLen/2
2973                  (*mSegments)[i].degPos = (((0) > (degStart - ((*mSegments)[i].segLen +
totConsecSegLen)/2)) ? (0) : (degStart - ((*mSegments)[i].segLen +
totConsecSegLen)/2))
2974              }
2975
2976              //Expand the first segment of the series
2977              (*mSegments)[i].segLen += totConsecSegLen
2978
2979              if ((*mSegments)[i].segType == TA_SEG_PAUSE || (*mSegments)[i].segType ==
TA_SEG_GUESSED)
2980                  (*mSegments)[i].reliability = avgReliability / (XFLOAT)(lastConsecSeg-i)
2981          }
2982      }
2983  }
2984
2985  void SQTimeAlignment::FineDelayPostProc(XFLOAT const *fDegNorm, int const &lDegNorm,
2986                                          int const &lRefNorm)
2987  {
2988      OPTTRY
2989      {
2990          if (lDegNorm <= 0 || lRefNorm <= 0 || fDegNorm == NULL ||
2991              mSegments == NULL || mSegments->size() == 0)
2992              OPTTHROW( string("Invalid input parameters."))
2993
```

```
2994        for (TA_SegList::iterator seg = mSegments->begin()  seg != mSegments->end()  seg++)
2995        {
2996            if (seg->segType == TA_SEG_MISSING)
2997            {
2998                if (seg != mSegments->begin() && (seg-1)->segType == TA_SEG_GUESSED)
2999                    (seg-1)->dontMergeWithOthers = true
3000                if (seg != mSegments->end()-1 && (seg+1)->segType == TA_SEG_GUESSED)
3001                    (seg+1)->dontMergeWithOthers = true
3002            }
3003        }
3004
3005        delete mMergedSegments
3006        mMergedSegments = new TA_SegList
3007
3008        for (int i = 0  i < (int)mSegments->size()  i++)
3009        {
3010            if ((*mSegments)[i].segType == TA_SEG_MISSING || (*mSegments)[i].dontMergeWithOthers)
3011            {
3012                mMergedSegments->insert(mMergedSegments->getIterator(i), (*mSegments)[i])
3013                continue
3014            }
3015
3016            TA_SEG_TYPE curSegType   = (*mSegments)[i].segType,
3017                        prevSegType  = TA_SEG_PAUSE,
3018                        firstSegType = (*mSegments)[i].segType,
3019                        finalSegType = curSegType
3020            int lastConsecSeg,
3021                lastMatchedSeg,
3022                lenTillLastMatchedSeg= 0,
3023                curSegLen            = (*mSegments)[i].segLen,
3024                totConsecSegLen      = curSegLen,
3025                curShift             = (*mSegments)[i].degPos - (*mSegments)[i].refPos,
3026                prevMatchedSegLen    = curSegType == TA_SEG_MATCHED? curSegLen : 0,
3027                prevGuessedSegLen    = curSegType != TA_SEG_MATCHED? curSegLen : 0
3028            XFLOAT avgWeightedRlblt   = (*mSegments)[i].reliability * curSegLen
3029
3030            //Find last consecutive seg w/ same or similar shift
3031            for (lastConsecSeg = i+1, lastMatchedSeg = -1
3032                 lastConsecSeg < (int)mSegments->size() &&
3033
3034                 (*mSegments)[lastConsecSeg].segType != TA_SEG_MISSING &&
3035                 !(*mSegments)[lastConsecSeg].dontMergeWithOthers &&
3036
3037                 (((*mSegments)[lastConsecSeg].segType != TA_SEG_PAUSE && firstSegType !=
    TA_SEG_PAUSE) ||
3038                  ((*mSegments)[lastConsecSeg].segType == TA_SEG_PAUSE && firstSegType ==
    TA_SEG_PAUSE))
3039                 &&
3040
3041                 abs( (*mSegments)[lastConsecSeg].degPos - (*mSegments)[lastConsecSeg].refPos -
    curShift ) <= TA_SHIFT_TOLERANCE
3042                 &&
3043
3044                 (prevSegType != TA_SEG_PAUSE || firstSegType == TA_SEG_PAUSE ||
3045                  firstSegType == TA_SEG_MATCHED || (*mSegments)[lastConsecSeg].segType ==
    TA_SEG_GUESSED) &&
3046
3047                 (*mSegments)[lastConsecSeg].refPos + curShift + (*mSegments)[lastConsecSeg].segLen
    <= mDegLen
3048
3049                 lastConsecSeg++)
3050            {
3051                curSegLen        = (*mSegments)[lastConsecSeg].segLen
3052                curSegType       = (*mSegments)[lastConsecSeg].segType
3053                totConsecSegLen += curSegLen
3054                avgWeightedRlblt+= (*mSegments)[lastConsecSeg].reliability * curSegLen
3055
3056                if (curSegType == TA_SEG_MATCHED)
```

```
3057                {
3058                    if (curSegLen + prevMatchedSegLen > 5*prevGuessedSegLen)
3059                        finalSegType = curSegType
3060
3061                    lastMatchedSeg        = lastConsecSeg
3062                    lenTillLastMatchedSeg = totConsecSegLen
3063                    prevMatchedSegLen    += curSegLen
3064                }
3065                else
3066                {
3067                    if (curSegLen + prevGuessedSegLen > prevMatchedSegLen/5)
3068                        finalSegType = TA_SEG_GUESSED
3069
3070                    prevGuessedSegLen += curSegLen
3071                }
3072            }
3073
3074            if (totConsecSegLen > (*mSegments)[i].segLen)
3075            {   //found 1 or more consecutive segs w/ same or similar shift
3076
3077                if (firstSegType == TA_SEG_PAUSE)
3078                    finalSegType = TA_SEG_PAUSE
3079
3080                if (finalSegType == TA_SEG_GUESSED && lastMatchedSeg > i &&
3081                    lenTillLastMatchedSeg >= 0.75f * totConsecSegLen &&
3082                    prevMatchedSegLen > lenTillLastMatchedSeg - prevMatchedSegLen)
3083                {
3084                    mMergedSegments->insert(mMergedSegments->getIterator(i),
3085                                            TA_segStruct((*mSegments)[i].refPos,
    (*mSegments)[i].degPos,
3086                                                         lenTillLastMatchedSeg, TA_SEG_MATCHED,
3087                                                         false, avgWeightedRlblt/totConsecSegLen))
3088                    prevSegType = TA_SEG_MATCHED
3089                    i = lastMatchedSeg - 1
3090                }
3091                else
3092                {
3093                    mMergedSegments->insert(mMergedSegments->getIterator(i),
3094                                            TA_segStruct((*mSegments)[i].refPos,
    (*mSegments)[i].degPos,
3095                                                         totConsecSegLen, finalSegType,
3096                                                         false, avgWeightedRlblt/totConsecSegLen))
3097
3098                    prevSegType = curSegType
3099                    i = lastConsecSeg - 1
3100                }
3101            }
3102            else
3103            {
3104                mMergedSegments->insert(mMergedSegments->getIterator(i), (*mSegments)[i])
3105                prevSegType = curSegType
3106            }
3107        }
3108
3109        //Fill mUnusedDegSegments with, well, unused deg signal parts!
3110        delete mUnusedDegSegments
3111        mUnusedDegSegments = new TA_SegList()
3112        int unusedDegSegStart = 0, unusedDegSegLen = 0, suggestedShift = 0
3113        while (mSegments->findNextGapInDegSegList(unusedDegSegStart, unusedDegSegLen,
    suggestedShift,
3114                                                  fDegNorm, lDegNorm, lRefNorm)
3115               == 0)
3116        {
3117            int suggestedRefPos = unusedDegSegStart - suggestedShift
3118            mUnusedDegSegments->insert(mUnusedDegSegments->findInsLocDeg(unusedDegSegStart),
3119                                       TA_segStruct(suggestedRefPos, unusedDegSegStart,
3120                                                    unusedDegSegLen, TA_SEG_MISSING,
3121                                                    false, 0.0f))
```

```
3122            unusedDegSegStart += unusedDegSegLen
3123        }
3124
3125        //Scale all segment lists to mTargetRate
3126        for (int i = 0  i < (int)mSegments->size()  i++)
3127        {
3128            (*mSegments)[i].refPos *= mTargetRate/TA_SAMPLING_RATE
3129            (*mSegments)[i].degPos *= mTargetRate/TA_SAMPLING_RATE
3130            (*mSegments)[i].segLen *= mTargetRate/TA_SAMPLING_RATE
3131        }
3132        for (int i = 0  i < (int)mMergedSegments->size()  i++)
3133        {
3134            (*mMergedSegments)[i].refPos *= mTargetRate/TA_SAMPLING_RATE
3135            (*mMergedSegments)[i].degPos *= mTargetRate/TA_SAMPLING_RATE
3136            (*mMergedSegments)[i].segLen *= mTargetRate/TA_SAMPLING_RATE
3137        }
3138        for (int i = 0  i < (int)mUnusedDegSegments->size()  i++)
3139        {
3140            (*mUnusedDegSegments)[i].refPos *= mTargetRate/TA_SAMPLING_RATE
3141            (*mUnusedDegSegments)[i].degPos *= mTargetRate/TA_SAMPLING_RATE
3142            (*mUnusedDegSegments)[i].segLen *= mTargetRate/TA_SAMPLING_RATE
3143        }
3144        mCurSegmentRate = mTargetRate
3145    }
3146    OPTCATCH((string errorMsg))
3147    {
3148        OPTTHROW( string("ERROR in FineDelayPostProc: " + errorMsg + "\n"))
3149    }
3150 }
3151
3152 }
3153
3154 }
3155
3156
```