```
    typedef double XFLOAT
    typedef double OTA_FLOAT


{
    struct DisturbancePara
    {
        DisturbancePara(int ListeningMode, int Model)
        {
            delayVariationImpact = 20.0
            MinSRR = 0.99
            MaxSRR = 1.05
            loudNoiseP1delta = 2.1
            loudNoiseP2add = 0.16
            loudNoiseP3pow = 0.4
            IntellLevelCorrectionP1max = 0.1
            IntellLevelCorrectionP2max = 0.4
            IntellLevelCorrectionP3max = 0.1
            IntellLevelCorrectionP4max = 0.3
            SilentCriterium = 6.0E6
            DistortedSilencePowerMeanCompensationFac = 1.0
            GlobalScaleCorrectionActiveP1pow = 0.05
            GlobalScaleCorrectionActiveP2pow = 0.14
            PureFreqP1 = 0.2
            LocScaleP1 = 0.45
            LocScaleP2 = 0.35
            LocScaleP3 = 1 - LocScaleP1 - LocScaleP2
            LocScaleP4pow = 0.7
            LocLoudScaleBandLowTimeclip = 0.0
            LocLoudScaleBandHighTimeclip = 99.0
            FreqRespCompPow = 0.4
            PartGlobLoudScaleP1 = 0.9
            NoiseContrSupSilP1 = 0.3
        }

        XFLOAT delayVariationImpact
        XFLOAT MinSRR
        XFLOAT MaxSRR
        XFLOAT loudNoiseP1delta
        XFLOAT loudNoiseP2add
        XFLOAT loudNoiseP3pow
        XFLOAT IntellLevelCorrectionP1max
        XFLOAT IntellLevelCorrectionP2max
        XFLOAT IntellLevelCorrectionP3max
        XFLOAT IntellLevelCorrectionP4max
        XFLOAT      SilentCriterium
        XFLOAT      DistortedSilencePowerMeanCompensationFac
        XFLOAT      GlobalScaleCorrectionActiveP1pow
        XFLOAT      GlobalScaleCorrectionActiveP2pow
        XFLOAT      PureFreqP1
        XFLOAT      LocScaleP1
        XFLOAT      LocScaleP2
        XFLOAT      LocScaleP3
        XFLOAT      LocScaleP4pow
        XFLOAT      LocLoudScaleBandLowTimeclip
        XFLOAT      LocLoudScaleBandHighTimeclip
        XFLOAT      FreqRespCompPow
        XFLOAT      PartGlobLoudScaleP1
        XFLOAT      NoiseContrSupSilP1

    }
}

const int kNarrowBand     = 0
const int kWideBand       = 1
const int kSuperWideBand = 2
```

```


{
    void InternalCorrectedLoudness(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CDoubleArray& aOriginalLoudness, CDoubleArray& aDistortedLoudness,
        CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, XFLOAT* LpLoudness, int bandIdxLow, int bandIdxHigh)
    void FreqResponseCompensLoudnessDomain(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CBarkSpectrum& originalPitchLoudnessDensityMainAvg, CBarkSpectrum& distortedPitchLoudnessDensityMainAvg,
        CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity,
        CIntArray& aActiveFreqresponse, XFLOAT powFact1, XFLOAT powFact2, XFLOAT powFact3, XFLOAT Fact)
    void FreqResponseCompensLoudnessDomainDistorted(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CBarkSpectrum& originalPitchLoudnessDensityMainAvg, CBarkSpectrum& distortedPitchLoudnessDensityMainAvg,
        CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity,
        CIntArray& aActiveFreqresponse, XFLOAT powFact1, XFLOAT powFact2, XFLOAT powFact3, XFLOAT Fact)
    void FreqResponseCompensPowerDomain(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CBarkSpectrum& originalPitchPowerDensityMainAvg, CBarkSpectrum& distortedPitchPowerDensityMainAvg, CBarkSpectrum& originalPitchPowerDensity, CBarkSpectrum& distortedPitchPowerDensity,
        CIntArray& aActiveFreqresponseIntell, XFLOAT powFact1, XFLOAT powFact2, XFLOAT powFact3, XFLOAT Fact, int original)
    void LoudnessScalingTowards20Sone(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CDoubleArray& aDistortedLoudness, CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, XFLOAT* aLoudnessScalingDistorted,
        XFLOAT* aDistortedLoudnessMean, int numberOfSpeechFrames, int bandIdxLow, int bandIdxHigh, XFLOAT bandHighParam, XFLOAT bandLowParam, XFLOAT lpOverBandRangeParam, XFLOAT mulParam1,
        XFLOAT const1, XFLOAT const2, XFLOAT const3, XFLOAT mulParam2)
}

class MatFreeDeleter
{
public:
    void operator()(void* p)
    {
        matFree(p)
    }
}


{

    void SlidingWinMean(CPOLQAData *PolqaHandle, XFLOAT const *in, int oddWinlen, XFLOAT *out, int len)
    void SlidingWinPowNorm(CPOLQAData *POLQAHandle,
        XFLOAT const *In, XFLOAT *Out,
        int len,
        XFLOAT thresh,
        int WINLEN,
        XFLOAT meanOutLevel,
        bool doAvoidShortBursts,
        int const hystLenInSamples)
    void WarpSpectrum(XFLOAT const *OrigSpec, XFLOAT *WarpedSpec, XFLOAT WarpingFac, int NumBands)

    void CreateArrayFromCSignal(double*** pppArray, CSignal* pSignal)
    {
        double** ppArray
        *pppArray = (double**)matMalloc(pSignal->aNumberOfWindows * sizeof(double*))
        ppArray = *pppArray

        for (int f = 0  f < pSignal->aNumberOfWindows  f++)
        {
            ppArray[f] = (double*)matMalloc(pSignal->aNumberOfBands * sizeof(double))
            for (int b = 0  b < pSignal->aNumberOfBands  b++)
                ppArray[f][b] = pSignal->m_pData[f][b]
```

```
121         }
122     }
123
124     XFLOAT CPairParameters::ComputePitchRatio_median()
125     {
126         XFLOAT PitchRatio = 1.0
127
128         std::vector<XFLOAT>PitchRatioVector
129         XFLOAT ratio
130         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++){
131             if (mpPitchVec[frameIndex] > 0 && mpPitchVecDeg[frameIndex] > 0){
132                 ratio = mpPitchVecDeg[frameIndex] / mpPitchVec[frameIndex]
133                 PitchRatioVector.push_back(ratio)
134             }
135         }
136
137         if (PitchRatioVector.size() < 50)
138             return 1.0
139
140         std::nth_element(PitchRatioVector.begin(), PitchRatioVector.begin() +
    PitchRatioVector.size() / 2, PitchRatioVector.end())
141         XFLOAT medianPitchRatio = PitchRatioVector[PitchRatioVector.size() / 2]
142
143         XFLOAT const MIN_PITCH_RATIO_SHIFT = 0.02
144         if ((medianPitchRatio < (1.0 + MIN_PITCH_RATIO_SHIFT)) && (medianPitchRatio >(1.0 -
    MIN_PITCH_RATIO_SHIFT)))
145             return 1.0
146
147         return medianPitchRatio
148
149     }
150
151     bool CPairParameters::ComputeGainCorrectionFactorPerFrame(CDoubleArray&
    compensationFactorGaindBov, const CDoubleArray& distortedEnvelopeBP, const CDoubleArray&
    originalEnvelopeBP, const bool* pActiveFrameFlags, const PolqaStatics *statics)
152     {
153         bool enoughActiveFramesForEstimate = false
154
155         std::vector<XFLOAT>envelopeDifferenceDegVsRefVector
156         XFLOAT const minRefEnvelopeActiveSpeech = -75.0
157         for (int i = statics->startFrameIdx  i <= statics->stopFrameIdx  i++)
158         {
159             if ((pActiveFrameFlags[i]) && (originalEnvelopeBP.m_pData[i] >
    minRefEnvelopeActiveSpeech))
160             {
161                 envelopeDifferenceDegVsRefVector.push_back(distortedEnvelopeBP.m_pData[i] -
    originalEnvelopeBP.m_pData[i])
162             }
163         }
164
165         if (envelopeDifferenceDegVsRefVector.size() > 20)
166         {
167             enoughActiveFramesForEstimate = true
168
169             XFLOAT medianEnvelopeDifferenceDegVsRefVector = 0.0
170             std::vector<XFLOAT>sortedEnvelopeDegVsRefVector = envelopeDifferenceDegVsRefVector
171             std::nth_element(sortedEnvelopeDegVsRefVector.begin(),
    sortedEnvelopeDegVsRefVector.begin() + sortedEnvelopeDegVsRefVector.size() / 2,
    sortedEnvelopeDegVsRefVector.end())
172             medianEnvelopeDifferenceDegVsRefVector =
    sortedEnvelopeDegVsRefVector[sortedEnvelopeDegVsRefVector.size() / 2]
173
174             std::vector<XFLOAT>envelopeSlowVariationDegVsRefVector
175             XFLOAT maxVariation = 15.0
176             int WinLenGainVar = 5
177             XFLOAT gainVarFrame, sumWinGainVariation
178             int WinLenGainVarRange = int(WinLenGainVar / 2.0)
```

```
179             int frameidx
180             XFLOAT curGainVar, prevGainVar, nextGainVar
181             for (int i = statics->startFrameIdx  i <= statics->stopFrameIdx  i++)
182             {
183                 if ((pActiveFrameFlags[i]) && (originalEnvelopeBP.m_pData[i] >
    minRefEnvelopeActiveSpeech))
184                 {
185                     sumWinGainVariation = 0.0
186                     for (int j = 0  j < WinLenGainVar  j++)
187                     {
188                         frameidx = ((((((i + j - WinLenGainVarRange) < (statics->stopFrameIdx)) ? (i
    + j - WinLenGainVarRange) : (statics->stopFrameIdx))) >
    (statics->startFrameIdx)) ? ((((i + j - WinLenGainVarRange) <
    (statics->stopFrameIdx)) ? (i + j - WinLenGainVarRange) :
    (statics->stopFrameIdx))) : (statics->startFrameIdx))
189                         if (!pActiveFrameFlags[frameidx] || (originalEnvelopeBP.m_pData[frameidx] <=
    minRefEnvelopeActiveSpeech)) frameidx = i
190
191                         gainVarFrame = distortedEnvelopeBP.m_pData[frameidx] -
    originalEnvelopeBP.m_pData[frameidx] -
    medianEnvelopeDifferenceDegVsRefVector
192                         if (gainVarFrame < (-1.0)*maxVariation) sumWinGainVariation -= maxVariation
193                         else if (gainVarFrame > maxVariation) sumWinGainVariation += maxVariation
194                         else sumWinGainVariation += gainVarFrame
195                     }
196                     sumWinGainVariation = sumWinGainVariation / XFLOAT(WinLenGainVar)
197                     compensationFactorGaindBov.m_pData[i] = sumWinGainVariation
198
199                 }
200             }
201
202         }
203
204         return enoughActiveFramesForEstimate
205
206     }
207
208     void CreateAlignTimeSeries(CPOLQAData *POLQAHandle, const CTimeSeries &InputTimeSeries, const
    CIntArray &aStartSampleUtterance, const CIntArray &aStopSampleUtterance, const CIntArray
    &aDelayUtterance, const int frameLength, CTimeSeries &AlignedTimeSeries)
209     {
210
211         int BlendLen = frameLength / 32
212         XFLOAT SinTab[(2048/32)]
213         XFLOAT CosTab[(2048/32)]
214         XFLOAT Step = 3.1415926535898 / (2 * (BlendLen - 1))
215         const XFLOAT OverlapCoeff = (1.0 - 0.75)
216         for (int i = 0  i < BlendLen  i++)
217         {
218             SinTab[i] = sin(i*Step)
219             CosTab[i] = cos(i*Step)
220             SinTab[i] = SinTab[i] * SinTab[i]
221             CosTab[i] = CosTab[i] * CosTab[i]
222         }
223
224         int LastStartFrameInputSample = 0
225         int LastStartFrameAlignedSample = 0
226         int LastDelay = 0
227         bool FirstFrame = true
228         int AlignedSequenceLength = AlignedTimeSeries.GetLength()
229         int LastInputFrame = InputTimeSeries.GetLength() - OverlapCoeff*frameLength
230         const int stopFrameIdx = POLQAHandle->statics->stopFrameIdx
231         for (int frameIndex = POLQAHandle->statics->startFrameIdx  frameIndex <= stopFrameIdx
    frameIndex++)
232         {
233             int Utt = GetUtteranceForFrame(aStartSampleUtterance, aStopSampleUtterance,
    aDelayUtterance, frameIndex, frameLength)
234             int Delay = aDelayUtterance.m_pData[Utt]
```

```
            int StartFrameInputSample = frameIndex * (int)(OverlapCoeff*frameLength) + Delay
            int StartFrameAlignedSample = frameIndex * (int)(OverlapCoeff*frameLength)

            if (StartFrameAlignedSample >= 0 && StartFrameAlignedSample < AlignedSequenceLength - OverlapCoeff*frameLength)
            {
                if (StartFrameInputSample >= 0 && StartFrameInputSample < LastInputFrame)
                {
                    matbCopy(InputTimeSeries.m_pData + StartFrameInputSample, AlignedTimeSeries.m_pData + StartFrameAlignedSample, (int)frameLength / 2 )

                    if (!FirstFrame && LastDelay != Delay)
                    {
                        int AlignedStart = StartFrameAlignedSample - BlendLen / 2
                        int SrcStart = StartFrameInputSample - BlendLen / 2
                        int LastStart = LastStartFrameInputSample + OverlapCoeff*frameLength - BlendLen / 2
                        int MaxBlend = BlendLen + (((0) < ((AlignedSequenceLength - SrcStart - BlendLen))) ? (0) : ((AlignedSequenceLength - SrcStart - BlendLen)))
                        for (int i = 0  i < MaxBlend  i++)
                            if (SrcStart + i >= 0)
                                *(AlignedTimeSeries.m_pData + AlignedStart + i) = CosTab[i] * *(InputTimeSeries.m_pData + LastStart + i) + SinTab[i] * *(InputTimeSeries.m_pData + SrcStart + i)
                            else
                                *(AlignedTimeSeries.m_pData + AlignedStart + i) = CosTab[i] * *(InputTimeSeries.m_pData + LastStart + i) + SinTab[i] * 0
                    }

                    LastStartFrameInputSample = StartFrameInputSample
                    LastStartFrameAlignedSample = StartFrameAlignedSample
                    LastDelay = Delay
                    FirstFrame = false
                }
                else
                    for (int i = StartFrameAlignedSample  i < StartFrameAlignedSample + OverlapCoeff*frameLength  i++)
                        AlignedTimeSeries.m_pData[i] = 0.0
            }

        }
    }

    void CPairParameters::ShiftPitch(CHzSpectrum const *spectrumRef, CHzSpectrum const *spectrumDeg, CHzSpectrum *spectrumDegCorrected, bool const *pActiveFrameFlags,
        int* bestShiftPerFrame, XFLOAT *bestWarpingFacPerFrame)
    {
        const XFLOAT freqRes = statics->aFrequencyResolutionHz
        XFLOAT const FRAMES_PER_SEC = statics->sampleRate / (statics->frameLength *(1 - 0.75))

        int    const SHIFT_PITCH_SMOOTHING_LEN =
            (int)((XFLOAT)100.0 / freqRes + 0.5) | 0x1
        int    const SHIFT_PITCH_MAX_PITCH_BIN = (int)(1500.0 / freqRes + 0.5)
        int    const SHIFT_PITCH_MAX_SPEC_BIN = (int)(3000.0 / freqRes + 0.5)
        XFLOAT const SHIFT_PITCH_MIN_CORR = 0.8
        int    const SHIFT_PITCH_FRAME_SEARCH_LEN = (((1) > ((int)((XFLOAT)0.016 * FRAMES_PER_SEC))) ? (1) : ((int)((XFLOAT)0.016 * FRAMES_PER_SEC)))
        XFLOAT const SHIFT_PITCH_MAX_PITCH_RATIO = (XFLOAT)1.1
        XFLOAT const SHIFT_PITCH_CORREL_SPL_THR =
            20.0 / 10.0

        XFLOAT const SHIFT_PITCH_CORREL_DAMPING =
            0.08
        XFLOAT  const SHIFT_PITCH_MAX_LOGFAC_CHANGE =
            fabs(log10((XFLOAT)0.96)) * (XFLOAT)62.5 / (XFLOAT)FRAMES_PER_SEC

        const int NumBands = statics->aNumberOfHzBands

```

```
293         SmartBufferPolqa SB1(this->POLQAHandle, NumBands)
294         SmartBufferPolqa SB2(this->POLQAHandle, NumBands)
295         SmartBufferPolqa SB3(this->POLQAHandle, NumBands)
296         SmartBufferPolqa SB4(this->POLQAHandle, NumBands)
297
298         XFLOAT *SmoothedRefSpec = SB1.Buffer
299         XFLOAT *SmoothedDegSpec = SB2.Buffer
300         XFLOAT *OrigSmoothedDegSpec = SB3.Buffer
301         XFLOAT *WarpedDegSpec = SB4.Buffer
302
303         XFLOAT MaxRefLev, MaxDegLev
304         XFLOAT MaxXCorr, CorrBefore, CorrAfter, BestCorr = -1.0, RefAutoCorr, DegAutoCorr
305         XFLOAT RefPitch, DegPitch, MinPitchRatio, MaxPitchRatio, BestWarpingFac = 1.0
306         int    RefStartBin, RefStopBin, DegStartBin, DegStopBin, StartBin, StopBin, BinLen,
307             WarpedStartBin, WarpedStopBin, WarpedBinLen, MaxShiftLen
308
309         for (int i = statics->startFrameIdx  i <= statics->stopFrameIdx  i++)
310         {
311             bestShiftPerFrame[i] = 0
312             bestWarpingFacPerFrame[i] = 1.0
313
314             MaxRefLev = log10(matMax(spectrumRef->m_pData[i] + 1, NumBands - 1) + 1e-10)
315             MaxDegLev = log10(matMax(spectrumDeg->m_pData[i] + 1, NumBands - 1) + 1e-10)
316             if (!pActiveFrameFlags[i] || MaxRefLev < SHIFT_PITCH_CORREL_SPL_THR || MaxDegLev <
    SHIFT_PITCH_CORREL_SPL_THR)
317             {
318                 continue
319             }
320
321             SlidingWinMean(this->POLQAHandle, spectrumRef->m_pData[i] + 1,
    SHIFT_PITCH_SMOOTHING_LEN, WarpedDegSpec + 1, NumBands - 1)
322             matbThresh1(WarpedDegSpec + 1, NumBands - 1, (XFLOAT)0.0, MAT_LT)
323             matbLog2(WarpedDegSpec + 1, SmoothedRefSpec + 1, NumBands - 1)
324             SlidingWinMean(this->POLQAHandle, spectrumDeg->m_pData[i] + 1,
    SHIFT_PITCH_SMOOTHING_LEN, WarpedDegSpec + 1, NumBands - 1)
325             matbThresh1(WarpedDegSpec + 1, NumBands - 1, (XFLOAT)0.0, MAT_LT)
326             matbLog2(WarpedDegSpec + 1, SmoothedDegSpec + 1, NumBands - 1)
327
328             CorrBefore = CorrAfter = 0.0
329             matbAdd1(-SHIFT_PITCH_CORREL_SPL_THR, SmoothedRefSpec + 1, NumBands - 1)
330             matbAdd1(-SHIFT_PITCH_CORREL_SPL_THR, SmoothedDegSpec + 1, NumBands - 1)
331             matbThresh1(SmoothedRefSpec + 1, NumBands - 1, 0.0, MAT_LT)
332             matbThresh1(SmoothedDegSpec + 1, NumBands - 1, 0.0, MAT_LT)
333
334             for (RefStartBin = 1  RefStartBin < NumBands - 1 && SmoothedRefSpec[RefStartBin] == 0.0
     RefStartBin++)
335             for (DegStartBin = 1  DegStartBin < NumBands - 1 && SmoothedDegSpec[DegStartBin] == 0.0
     DegStartBin++)
336             for (RefStopBin = NumBands - 1  RefStopBin >= 0 && SmoothedRefSpec[RefStopBin] == 0.0
    RefStopBin--)
337             for (DegStopBin = NumBands - 1  DegStopBin >= 0 && SmoothedDegSpec[DegStopBin] == 0.0
    DegStopBin--)
338             StartBin = (((RefStartBin) > (DegStartBin)) ? (RefStartBin) : (DegStartBin))
339             StopBin = (((RefStopBin) < ((((DegStopBin) < (SHIFT_PITCH_MAX_SPEC_BIN)) ? (DegStopBin)
    : (SHIFT_PITCH_MAX_SPEC_BIN)))) ? (RefStopBin) : ((((DegStopBin) <
    (SHIFT_PITCH_MAX_SPEC_BIN)) ? (DegStopBin) : (SHIFT_PITCH_MAX_SPEC_BIN))))
340             BinLen = StopBin - StartBin + 1
341             if (BinLen < (int)(1000.0 / freqRes + 0.5))
342             {
343                 continue
344             }
345
346             SlidingWinPowNorm(POLQAHandle, SmoothedRefSpec + 1, WarpedDegSpec + 1, NumBands - 1,
    0.0,
347                 2 * SHIFT_PITCH_SMOOTHING_LEN + 1, 10.0*SHIFT_PITCH_CORREL_SPL_THR, false, 2)
348             matbCopy(WarpedDegSpec + 1, SmoothedRefSpec + 1, NumBands - 1)
349             SlidingWinPowNorm(POLQAHandle, SmoothedDegSpec + 1, WarpedDegSpec + 1, NumBands - 1,
    0.0,
```

```
350                 2 * SHIFT_PITCH_SMOOTHING_LEN + 1, 10.0*SHIFT_PITCH_CORREL_SPL_THR, false, 2)
351             matbCopy(WarpedDegSpec + 1, SmoothedDegSpec + 1, NumBands - 1)
352
353             matbCopy(SmoothedDegSpec + 1, OrigSmoothedDegSpec + 1, NumBands - 1)
354
355             MaxShiftLen = ((((int)ceil((SHIFT_PITCH_MAX_PITCH_RATIO - 1.0)*BinLen)) < (NumBands / 2
    - 1)) ? ((int)ceil((SHIFT_PITCH_MAX_PITCH_RATIO - 1.0)*BinLen)) : (NumBands / 2 - 1))
356             RefAutoCorr = matbNormL2(SmoothedRefSpec + StartBin, BinLen)
357             DegAutoCorr = matbNormL2(SmoothedDegSpec + StartBin, BinLen)
358             matCrossCorr(POLQAHandle->mh, SmoothedRefSpec + StartBin, BinLen, SmoothedDegSpec +
    StartBin, BinLen,
359                 WarpedDegSpec, 2 * MaxShiftLen + 1, -MaxShiftLen)
360             MaxXCorr = matMax(WarpedDegSpec, 2 * MaxShiftLen + 1)
361             MaxXCorr /= RefAutoCorr*DegAutoCorr
362
363             if (MaxXCorr < SHIFT_PITCH_MIN_CORR)
364             {
365                 continue
366             }
367
368             MinPitchRatio = MaxPitchRatio = 1.0f
369             for (int j = -SHIFT_PITCH_FRAME_SEARCH_LEN  j <= SHIFT_PITCH_FRAME_SEARCH_LEN  j++)
370             {
371                 if (i + j < statics->startFrameIdx ||
372                     i + j > statics->stopFrameIdx ||
373                     (RefPitch = CPairParameters::mpPitchVecDeg[i + j]) <= 0.0 ||
374                     (DegPitch = CPairParameters::mpPitchVec[i + j]) <= 0.0)
375                     continue
376
377                 if (RefPitch / DegPitch >= 2.0)
378                     RefPitch /= 2.0
379                 if (DegPitch / RefPitch >= 2.0)
380                     DegPitch /= 2.0
381                 if (RefPitch > 0 && DegPitch > 0)
382                 {
383                     MinPitchRatio = (((DegPitch / RefPitch) < (MinPitchRatio)) ? (DegPitch /
    RefPitch) : (MinPitchRatio))
384                     MaxPitchRatio = (((DegPitch / RefPitch) > (MaxPitchRatio)) ? (DegPitch /
    RefPitch) : (MaxPitchRatio))
385                 }
386             }
387             if (AlmostEqualUlpsFinal((float)MinPitchRatio, 1.0f, 4) &&
    AlmostEqualUlpsFinal((float)MaxPitchRatio, 1.0f, 4))
388             {
389                 continue
390             }
391
392             MinPitchRatio = ((int)(((100 * (((1 / SHIFT_PITCH_MAX_PITCH_RATIO) > (MinPitchRatio -
    0.01)) ? (1 / SHIFT_PITCH_MAX_PITCH_RATIO) : (MinPitchRatio - 0.01))) > 0) ? (100 * (((1
    / SHIFT_PITCH_MAX_PITCH_RATIO) > (MinPitchRatio - 0.01)) ? (1 /
    SHIFT_PITCH_MAX_PITCH_RATIO) : (MinPitchRatio - 0.01)))+0.5f : (100 * (((1 /
    SHIFT_PITCH_MAX_PITCH_RATIO) > (MinPitchRatio - 0.01)) ? (1 /
    SHIFT_PITCH_MAX_PITCH_RATIO) : (MinPitchRatio - 0.01)))-0.5f)) / 100.0
393             MaxPitchRatio = ((int)(((100 * (((SHIFT_PITCH_MAX_PITCH_RATIO) < (MaxPitchRatio + 0.01))
    ? (SHIFT_PITCH_MAX_PITCH_RATIO) : (MaxPitchRatio + 0.01))) > 0) ? (100 *
    (((SHIFT_PITCH_MAX_PITCH_RATIO) < (MaxPitchRatio + 0.01)) ?
    (SHIFT_PITCH_MAX_PITCH_RATIO) : (MaxPitchRatio + 0.01)))+0.5f : (100 *
    (((SHIFT_PITCH_MAX_PITCH_RATIO) < (MaxPitchRatio + 0.01)) ?
    (SHIFT_PITCH_MAX_PITCH_RATIO) : (MaxPitchRatio + 0.01)))-0.5f)) / 100.0
394
395             BestCorr = -1.0
396             BestWarpingFac = 1.0
397             for (XFLOAT pitchRatio = MinPitchRatio  pitchRatio <= MaxPitchRatio  pitchRatio += 0.01)
398             {
399                 WarpSpectrum(spectrumDeg->m_pData[i], WarpedDegSpec, pitchRatio, NumBands)
400
401                 SlidingWinMean(this->POLQAHandle, WarpedDegSpec + 1, SHIFT_PITCH_SMOOTHING_LEN,
    SmoothedDegSpec + 1, NumBands - 1)
```

```
402                 matbThresh1(SmoothedDegSpec + 1, NumBands - 1, (XFLOAT)0.0, MAT_LT)
403                 matbLog2(SmoothedDegSpec + 1, WarpedDegSpec + 1, NumBands - 1)
404                 matbAdd1(-SHIFT_PITCH_CORREL_SPL_THR, WarpedDegSpec + 1, NumBands - 1)
405                 matbThresh1(WarpedDegSpec + 1, NumBands - 1, 0.0, MAT_LT)
406
407                 SlidingWinPowNorm(POLQAHandle, WarpedDegSpec + 1, SmoothedDegSpec + 1, NumBands - 1,
    0.0,
408                     2 * SHIFT_PITCH_SMOOTHING_LEN + 1, 10.0*SHIFT_PITCH_CORREL_SPL_THR, false, 2)
409
410                 WarpedStartBin = ((((int)ceil(StartBin*pitchRatio)) > (StartBin)) ?
    ((int)ceil(StartBin*pitchRatio)) : (StartBin))
411                 WarpedStopBin = ((((int)(StopBin *pitchRatio)) < ((((StopBin) <
    (SHIFT_PITCH_MAX_PITCH_BIN)) ? (StopBin) : (SHIFT_PITCH_MAX_PITCH_BIN)))) ?
    ((int)(StopBin *pitchRatio)) : ((((StopBin) < (SHIFT_PITCH_MAX_PITCH_BIN)) ?
    (StopBin) : (SHIFT_PITCH_MAX_PITCH_BIN))))
412                 WarpedBinLen = WarpedStopBin - WarpedStartBin + 1
413
414                 CorrAfter = matPearsonCorrelation(SmoothedRefSpec + WarpedStartBin, SmoothedDegSpec
    + WarpedStartBin, WarpedBinLen)
415
416                 if (CorrAfter > BestCorr)
417                 {
418                     BestCorr = CorrAfter
419                     BestWarpingFac = pitchRatio
420                 }
421             }
422
423             WarpSpectrum(spectrumDeg->m_pData[i], WarpedDegSpec, BestWarpingFac, NumBands)
424
425             SlidingWinMean(this->POLQAHandle, WarpedDegSpec + 1, SHIFT_PITCH_SMOOTHING_LEN,
    SmoothedDegSpec + 1, NumBands - 1)
426             matbThresh1(SmoothedDegSpec + 1, NumBands - 1, (XFLOAT)0.0, MAT_LT)
427             matbLog2(SmoothedDegSpec + 1, WarpedDegSpec + 1, NumBands - 1)
428             matbAdd1(-SHIFT_PITCH_CORREL_SPL_THR, WarpedDegSpec + 1, NumBands - 1)
429             matbThresh1(WarpedDegSpec + 1, NumBands - 1, 0.0, MAT_LT)
430
431             SlidingWinPowNorm(POLQAHandle, WarpedDegSpec + 1, SmoothedDegSpec + 1, NumBands - 1,
    0.0,
432                 2 * SHIFT_PITCH_SMOOTHING_LEN + 1, 10.0*SHIFT_PITCH_CORREL_SPL_THR, false, 2)
433
434             WarpedStartBin = ((((int)ceil(StartBin*BestWarpingFac)) > (StartBin)) ?
    ((int)ceil(StartBin*BestWarpingFac)) : (StartBin))
435             WarpedStopBin = ((((int)(StopBin *BestWarpingFac)) < ((((StopBin) <
    (SHIFT_PITCH_MAX_SPEC_BIN)) ? (StopBin) : (SHIFT_PITCH_MAX_SPEC_BIN)))) ? ((int)(StopBin
    *BestWarpingFac)) : ((((StopBin) < (SHIFT_PITCH_MAX_SPEC_BIN)) ? (StopBin) :
    (SHIFT_PITCH_MAX_SPEC_BIN))))
436             WarpedBinLen = WarpedStopBin - WarpedStartBin + 1
437
438             CorrBefore = matPearsonCorrelation(SmoothedRefSpec + WarpedStartBin, OrigSmoothedDegSpec
    + WarpedStartBin, WarpedBinLen)
439             CorrAfter = matPearsonCorrelation(SmoothedRefSpec + WarpedStartBin, SmoothedDegSpec +
    WarpedStartBin, WarpedBinLen)
440
441             if (CorrAfter - (1.0 / CorrBefore - 1.0)*SHIFT_PITCH_CORREL_DAMPING <= CorrBefore)
442             {
443                 continue
444             }
445             else
446             {
447                 bestShiftPerFrame[i] = (((1) > ((int)((int)(((fabs(1.0 -
    BestWarpingFac)*CPairParameters::mPitchFreqRef) > 0) ? (fabs(1.0 -
    BestWarpingFac)*CPairParameters::mPitchFreqRef)+0.5f : (fabs(1.0 -
    BestWarpingFac)*CPairParameters::mPitchFreqRef)-0.5f)))) ? (1) :
    ((int)((int)(((fabs(1.0 - BestWarpingFac)*CPairParameters::mPitchFreqRef) > 0) ?
    (fabs(1.0 - BestWarpingFac)*CPairParameters::mPitchFreqRef)+0.5f : (fabs(1.0 -
    BestWarpingFac)*CPairParameters::mPitchFreqRef)-0.5f))))
448                 bestWarpingFacPerFrame[i] = BestWarpingFac
449             }
```

```
450         }
451
452         XFLOAT PrevWarpingFacLog10 = (XFLOAT)0.0, WarpingFacLog10
453         for (int i = statics->startFrameIdx  i <= statics->stopFrameIdx  i++)
454         {
455             WarpingFacLog10 = log10((XFLOAT)bestWarpingFacPerFrame[i])
456
457             if (WarpingFacLog10 > PrevWarpingFacLog10)
458                 WarpingFacLog10 = (((WarpingFacLog10) < (PrevWarpingFacLog10 +
    SHIFT_PITCH_MAX_LOGFAC_CHANGE)) ? (WarpingFacLog10) : (PrevWarpingFacLog10 +
    SHIFT_PITCH_MAX_LOGFAC_CHANGE))
459             else
460                 WarpingFacLog10 = (((WarpingFacLog10) > (PrevWarpingFacLog10 -
    SHIFT_PITCH_MAX_LOGFAC_CHANGE)) ? (WarpingFacLog10) : (PrevWarpingFacLog10 -
    SHIFT_PITCH_MAX_LOGFAC_CHANGE))
461
462             WarpSpectrum(spectrumDeg->m_pData[i], spectrumDegCorrected->m_pData[i],
463                 (XFLOAT)pow((XFLOAT)10.0, WarpingFacLog10), NumBands)
464
465             PrevWarpingFacLog10 = WarpingFacLog10
466         }
467     }
468
469 void PerFrameDistortionCompensation(XFLOAT const *OrigSpec, XFLOAT const *DegSpec, XFLOAT*
    CorrectedSpec, CPOLQAData *POLQAHandle,int FrameInd)
470 {
471     int numOfBands = POLQAHandle->statics->aNumberOfHzBands
472     std::unique_ptr<XFLOAT[], MatFreeDeleter> OriginalSmoothed((XFLOAT*)matMalloc(numOfBands *
    sizeof(XFLOAT)))
473     matbCopy(OrigSpec,OriginalSmoothed.get(),numOfBands)
474     std::unique_ptr<XFLOAT[], MatFreeDeleter> DegSmoothed((XFLOAT*)matMalloc(numOfBands *
    sizeof(XFLOAT)))
475     matbCopy(DegSpec,DegSmoothed.get(),numOfBands)
476     XFLOAT meanEnergy = 0.0
477     float alpha = 0.9
478     float maxDeg = 0
479
480     for (int i=0 i<numOfBands i++)
481     {
482         meanEnergy = meanEnergy*alpha+OriginalSmoothed[i]*(1-alpha)
483         OriginalSmoothed[i]=meanEnergy
484     }
485             meanEnergy = 0.0
486     for (int i=0 i<numOfBands i++)
487     {
488         meanEnergy = meanEnergy*alpha+DegSmoothed[i]*(1-alpha)
489         DegSmoothed[i]=meanEnergy
490         maxDeg = ((DegSmoothed[i]>maxDeg)?DegSmoothed[i]:maxDeg)
491     }
492
493 float delta = 100
494
495       int compensationUpperBoundaryHz=numOfBands
496       int Akku = 0
497       float prevPoint = 0, currPoint = 0
498       for (int i=numOfBands-1 i>0 i--)
499       {
500           if ((DegSmoothed[i]>0.05*maxDeg) &&
    ((OriginalSmoothed[i]-DegSmoothed[i])/(OriginalSmoothed[i]+0.0)<0.9) ||
    (DegSmoothed[i]>5.0e4))
501               Akku++
502           else Akku = 0
503           if (Akku>5)
504           {
505               compensationUpperBoundaryHz = (int) (i+Akku-6)
506               i = 0
507           }
508
```

```
509       }
510       if ( compensationUpperBoundaryHz>numOfBands)
511           compensationUpperBoundaryHz =numOfBands
512       if (compensationUpperBoundaryHz<0)
513           compensationUpperBoundaryHz = 0
514
515       float ratio = 0
516       for (int j = 0  j<numOfBands  j++)
517       {
518           ratio = (OrigSpec[j]+delta)/(DegSpec[j]+delta)
519           if (ratio>1.2)
520               ratio = 1.2
521           if (ratio<0.8)
522               ratio = 0.8
523           CorrectedSpec[j] = ratio*DegSpec[j]
524     }
525
526 }
527
528     void WarpSpectrum(XFLOAT const *OrigSpec, XFLOAT *WarpedSpec, XFLOAT WarpingFac, int NumBands)
529     {
530         if (WarpingFac <= 0.0)
531             return
532         if (AlmostEqualUlpsFinal((float)WarpingFac, 1.0f, 4))
533         {
534             matbCopy(OrigSpec, WarpedSpec, NumBands)
535             return
536         }
537
538         WarpedSpec[0] = OrigSpec[0]
539         XFLOAT exactIdx, lowerRatio, upperRatio
540
541         WarpingFac = 1.0 / WarpingFac
542
543         int i
544         for (i = 1  i < NumBands && (int)ceil(WarpingFac*i) < NumBands  i++)
545         {
546             exactIdx   = (((WarpingFac*i) > (1.0)) ? (WarpingFac*i) : (1.0))
547             upperRatio = exactIdx - (int)(exactIdx)
548             lowerRatio = 1.0 - upperRatio
549
550             WarpedSpec[i] = lowerRatio*OrigSpec[(int)exactIdx] +
    upperRatio*OrigSpec[(int)ceil(exactIdx)]
551         }
552
553         if (i < NumBands && (int)(WarpingFac*i) < NumBands)
554         {
555             exactIdx = WarpingFac*i
556             lowerRatio = (int)ceil(exactIdx) - exactIdx
557
558             WarpedSpec[i] = lowerRatio*OrigSpec[(int)exactIdx]
559         }
560         for (  i < NumBands  i++)
561             WarpedSpec[i] = 1e-10
562     }
563
564     void SlidingWinMean(CPOLQAData *PolqaHandle, XFLOAT const *in, int oddWinlen, XFLOAT *out, int
    len)
565     {
566         if (oddWinlen > len)
567             oddWinlen = (len - 1) | 0x1
568         int i = 0, j, halfwinlen = oddWinlen / 2
569         if (in == NULL || out == NULL || oddWinlen <= 1)
570             return
571
572         SmartBufferPolqa SB1(PolqaHandle, oddWinlen)
573         XFLOAT *temp = SB1.Buffer
574
```

```
575         for (i = 0  i <= halfwinlen  i++)
576             temp[i % (halfwinlen + 1)] = matMean(in, 2 * i + 1)
577 
578         for (  i < len - halfwinlen  i++)
579         {
580             out[i - (halfwinlen + 1)] = temp[i % (halfwinlen + 1)]
581             temp[i % (halfwinlen + 1)] = matMean(in + i - halfwinlen, oddWinlen)
582         }
583 
584         for (  i < len  i++)
585         {
586             out[i - (halfwinlen + 1)] = temp[i % (halfwinlen + 1)]
587             temp[i % (halfwinlen + 1)] = matMean(in + len - 2 * (len - i - 1) - 1, 2 * (len - i - 1)
    + 1)
588         }
589 
590         for (j = 0  j < halfwinlen + 1  i++, j++)
591             out[i - (halfwinlen + 1)] = temp[i % (halfwinlen + 1)]
592 
593         return
594     }
595 
596     void SlidingWinPowNorm(CPOLQAData *POLQAHandle,
597         XFLOAT const *fIn, XFLOAT *fOut,
598         int len,
599         XFLOAT thresh,
600         int WINLEN,
601         XFLOAT meanOutLevel,
602         bool doAvoidShortBursts,
603         int hystLenInSamples)
604     {
605 
606         const double ROUNDFACTOR = 1e8
607 
608         if ((WINLEN | 0x1) != WINLEN)
609             WINLEN--
610 
611         if (fIn == NULL || fOut == NULL || len < WINLEN+3 ||
612             fIn == fOut || WINLEN < 2)
613             throw std::string("Invalid input parameters. Note: In-place operation is not
    supported.")
614 
615         double dWinEnergy  = 0.0
616         double dEnergThr   = thresh * WINLEN
617         int    hystCounter = 0
618         XFLOAT fTemp
619         fTemp = matbNormL2(fIn, WINLEN/2)
620 
621         dWinEnergy = floor(ROUNDFACTOR*fTemp*fTemp)/ROUNDFACTOR
622 
623         SmartBufferPolqa SB(POLQAHandle, len)
624         XFLOAT *fInSquared = SB.Buffer
625 
626         matbSqr2(fIn, fInSquared, len)
627 
628         //Process first WINLEN/2 data points
629         for (int i = 0  i < WINLEN/2+1  i++)
630         {
631             if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
632             {
633                 fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
634                 hystCounter = dWinEnergy > dEnergThr ?
635                     (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
    :
636                     (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
637             }
638             else
639             {
```

```
                fOut[i]     = (XFLOAT)0.0
                hystCounter = 0
            }

            dWinEnergy += floor(ROUNDFACTOR * fIn[i+WINLEN/2]*fIn[i+WINLEN/2])/ROUNDFACTOR

        }

        //Main loop
        for (int i = WINLEN/2+1  i < len - WINLEN/2  i++)
        {
            if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
            {
                fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
                hystCounter = dWinEnergy > dEnergThr ?
                    (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
:
                    (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
            }
            else
            {
                fOut[i]     = (XFLOAT)0.0
                hystCounter = 0
            }

            dWinEnergy = floor(matSum(&fInSquared[i - WINLEN/2], WINLEN) * ROUNDFACTOR)/ROUNDFACTOR

        }

        //Process last WINLEN/2 data points
        for (int i = len - WINLEN/2  i < len  i++)
        {
            if (dWinEnergy > dEnergThr || (hystCounter > 0 && dWinEnergy > 0.0))
            {
                fOut[i]     = (XFLOAT)(fIn[i] / sqrt(dWinEnergy / WINLEN))
                hystCounter = dWinEnergy > dEnergThr ?
                    (((hystLenInSamples) < (hystCounter+1)) ? (hystLenInSamples) : (hystCounter+1))
:
                    (((0) > (hystCounter-1)) ? (0) : (hystCounter-1))
            }
            else
            {
                fOut[i]     = (XFLOAT)0.0
                hystCounter = 0
            }

            dWinEnergy -= floor(ROUNDFACTOR * fIn[i-WINLEN/2-1]*fIn[i-WINLEN/2-1])/ROUNDFACTOR

        }

        if (doAvoidShortBursts)
        {
            int actStart = 0
            for (int i = 0  i < len  i++)
            {
                while (fOut[i] == (XFLOAT)0.0 && i < len) i++
                actStart = i
                while (fOut[i] != (XFLOAT)0.0 && i < len) i++
                if (i - actStart < 2*WINLEN)
                    for (int j = actStart  j < i  j++)
                        fOut[j] = (XFLOAT)0.0
            }
        }

        //Rescale processed data to desired level
        XFLOAT fScalefac
        fScalefac = matbNormL2(fOut, len) / sqrt((XFLOAT)len)
        if (fScalefac > (XFLOAT)0.0)
```

```
706         {
707             fScalefac = pow((XFLOAT)10.0, (XFLOAT)meanOutLevel/(XFLOAT)20.0) / fScalefac
708             matbMpy1(fScalefac, fOut, len)
709         }
710     }
711
712     //Local loudness scaling distorted,  predominantly for modelling the local impact of additive noise and pulses during silent intervals
713     void LoudnessScalingDistortedAdditiveNoise1(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, bool *UseThisFrame,
714         XFLOAT* originalLoudnessHulp, XFLOAT* distortedLoudnessHulp, XFLOAT* loudnessScaleLow, XFLOAT* oldLoudnessScaleLow, XFLOAT* oldOldLoudnessScaleLow, int bandIdxLow, int bandIdxHigh,
715         XFLOAT Blow, XFLOAT Bhigh, XFLOAT LpOverBandRange, XFLOAT ScaleDelta, XFLOAT OldOld, XFLOAT Old, XFLOAT noiseContrastMax1, XFLOAT pow1, XFLOAT pow2, XFLOAT pow3, XFLOAT pow4)
716     {
717         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(Blow)
718         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(Bhigh)
719         XFLOAT hulp
720         for (int frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx) frameIndex++)
721         {
722             if (UseThisFrame[frameIndex])
723             {
724                 *distortedLoudnessHulp = distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, LpOverBandRange, bandIdxLow, bandIdxHigh)
725                 *originalLoudnessHulp = originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, LpOverBandRange, bandIdxLow, bandIdxHigh)
726                 hulp = *originalLoudnessHulp
727                 if (hulp > 0.9) hulp = 0.9
728                 *loudnessScaleLow = (*originalLoudnessHulp + ScaleDelta - hulp) / (*distortedLoudnessHulp + ScaleDelta - hulp)
729                 *loudnessScaleLow = OldOld*(*oldOldLoudnessScaleLow) + Old*(*oldLoudnessScaleLow) + (1.0-Old-OldOld)*(*loudnessScaleLow)
730                     if (*loudnessScaleLow > 1.0) *loudnessScaleLow = 1.0
731                     if ((*oldOldLoudnessScaleLow < *oldLoudnessScaleLow) && (*oldLoudnessScaleLow < *loudnessScaleLow))
732                         distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, pow(*loudnessScaleLow, pow2), 0.0, 99.0)
733                     else
734                     {
735                         if ((*oldOldLoudnessScaleLow > *oldLoudnessScaleLow) && (*oldLoudnessScaleLow > *loudnessScaleLow))
736                             distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, pow(*loudnessScaleLow, pow3), 0.0, 99.0)
737                         else distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, pow(*loudnessScaleLow, pow4), 0.0, 99.0)
738                     }
739                 *oldOldLoudnessScaleLow = *oldLoudnessScaleLow
740                 *oldLoudnessScaleLow = *loudnessScaleLow
741             }
742             else {
743                 originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
744                 distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
745             }
746         }
747     }
748     void LoudnessScalingDistortedAdditiveNoise2(const PolqaStatics *statics, CPOLQAData *POLQAHandle, CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, bool *UseThisFrame,
749         XFLOAT* originalLoudnessHulp, XFLOAT* distortedLoudnessHulp, XFLOAT* loudnessScaleLow, XFLOAT* oldLoudnessScaleLow, XFLOAT* oldOldLoudnessScaleLow, int bandIdxLow, int bandIdxHigh,
750         XFLOAT Blow, XFLOAT Bhigh, XFLOAT LpOverBandRange, XFLOAT ScaleDelta, XFLOAT OldOld, XFLOAT Old, XFLOAT noiseContrastMax1, XFLOAT pow1)
751     {
```

```
752         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(Blow)
753         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(Bhigh)
754         XFLOAT hulp
755         for (int frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
756         {
757             if (UseThisFrame[frameIndex])
758             {
759                 *distortedLoudnessHulp =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    LpOverBandRange, bandIdxLow, bandIdxHigh)
760                 *originalLoudnessHulp = originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle,
    frameIndex, LpOverBandRange, bandIdxLow, bandIdxHigh)
761                 hulp = *originalLoudnessHulp
762                 if (hulp > 1.0) hulp = 1.0
763                 *loudnessScaleLow = (*originalLoudnessHulp + ScaleDelta*noiseContrastMax1) /
    (*distortedLoudnessHulp + ScaleDelta*noiseContrastMax1)
764                 *loudnessScaleLow = OldOld*(*oldOldLoudnessScaleLow) + Old*(*oldLoudnessScaleLow) +
    (1.0-Old-OldOld)*(*loudnessScaleLow)
765                 *oldOldLoudnessScaleLow = *oldLoudnessScaleLow
766                 *oldLoudnessScaleLow = *loudnessScaleLow
767                 if (*loudnessScaleLow > 1.0) *loudnessScaleLow = 1.0
768                     distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex,
    pow(*loudnessScaleLow, pow1), 0.0, 99.0)
769             }
770             else {
771                 originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
772                 distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
773             }
774         }
775     }
776     //Local loudness scaling original, predominantly for modelling the impact of time clip effects
    during active intervals
777     void LoudnessScalingOriginalClipEffect1(const PolqaStatics *statics, CPOLQAData *POLQAHandle,
    CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, bool
    *UseThisFrame,
778         XFLOAT* originalLoudnessHulp, XFLOAT* distortedLoudnessHulp, XFLOAT* loudnessScaleLow,
    XFLOAT* oldLoudnessScaleLow, XFLOAT* oldOldLoudnessScaleLow, int bandIdxLow, int
    bandIdxHigh,
779         XFLOAT LpOverBandRange, XFLOAT scaleDelta, XFLOAT pow1, XFLOAT pow2, XFLOAT pow3, XFLOAT
    noiseIndConst)
780     {
781         for (int frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
782         {
783             if (UseThisFrame[frameIndex])
784             {
785                 *distortedLoudnessHulp =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, 1.2,
    bandIdxLow, bandIdxHigh)
786                 *originalLoudnessHulp = originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle,
    frameIndex, LpOverBandRange, bandIdxLow, bandIdxHigh)
787                 *loudnessScaleLow = (*distortedLoudnessHulp + 8.0) / (*originalLoudnessHulp +
    scaleDelta)
788                 *loudnessScaleLow = 0.11*noiseIndConst*(*oldLoudnessScaleLow) + (1.0 -
    0.11*noiseIndConst)*(*loudnessScaleLow)
789                 if (*loudnessScaleLow<0.02) *loudnessScaleLow = 0.02
790                 if (*loudnessScaleLow>1.0) *loudnessScaleLow = 1.0
791                 if ((*oldOldLoudnessScaleLow < *oldLoudnessScaleLow) && (*oldLoudnessScaleLow <
    *loudnessScaleLow))
792                     originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex,
    pow(*loudnessScaleLow, pow1), 0.0, 99.0)
793
794                 else
795                 {
796                     if ((*oldOldLoudnessScaleLow > *oldLoudnessScaleLow) && (*oldLoudnessScaleLow >
    *loudnessScaleLow))
797                         originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex,
```

```
    pow(*loudnessScaleLow, pow2), 0.0, 99.0)
798                     else originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex,
    pow(*loudnessScaleLow, pow3), 0.0, 99.0)
799                 }
800                 *oldOldLoudnessScaleLow = *oldLoudnessScaleLow
801                 *oldLoudnessScaleLow = *loudnessScaleLow
802             }
803             else
804             {
805                 originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
806                 distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
807             }
808         }
809     }
810
811     void LoudnessScalingOriginalClipEffect2(const PolqaStatics *statics, CPOLQAData *POLQAHandle,
    CBarkSpectrum& originalLoudnessDensity, CBarkSpectrum& distortedLoudnessDensity, bool
    *UseThisFrame,
812         XFLOAT* originalLoudnessHulp, XFLOAT* distortedLoudnessHulp, XFLOAT* loudnessScaleLow,
    XFLOAT* oldLoudnessScaleLow, XFLOAT* oldOldLoudnessScaleLow, int bandIdxLow, int
    bandIdxHigh,
813         XFLOAT BLconst, XFLOAT BHconst, XFLOAT LpOverBandRange, XFLOAT scaleDelta, XFLOAT pow1)
814     {
815         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(BLconst)
816         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(BHconst)
817         for (int frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
818         {
819             if (UseThisFrame[frameIndex])
820             {
821                 *distortedLoudnessHulp =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    LpOverBandRange, bandIdxLow, bandIdxHigh)
822                 *originalLoudnessHulp = originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle,
    frameIndex, LpOverBandRange, bandIdxLow, bandIdxHigh)
823                 *loudnessScaleLow = (*distortedLoudnessHulp + scaleDelta) / (*originalLoudnessHulp +
    scaleDelta)
824                 *oldOldLoudnessScaleLow = *oldLoudnessScaleLow
825                 *oldLoudnessScaleLow = *loudnessScaleLow
826                 if (*loudnessScaleLow > 1.25) *loudnessScaleLow = 1.25
827                 originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, pow(*loudnessScaleLow,
    pow1), 0.0, 99.0)
828             }
829             else
830             {
831                 originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
832                 distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 0.0, 99.0)
833             }
834         }
835     }
836
837     void LoudnessScalingOriginalTowardsDistorted1(const PolqaStatics *statics, CPOLQAData
    *POLQAHandle, CDoubleArray& aOriginalLoudnessCompletelyScaled,
838         CDoubleArray& aDistortedLoudnessCompletelyScaled, CDoubleArray& aOriginalLoudness,
    CDoubleArray& aDistortedLoudness, CBarkSpectrum& originalLoudnessDensity,
839         CBarkSpectrum& distortedLoudnessDensity, XFLOAT *aLoudnessScalingOriginal, int
    numberOfSpeechFrames, XFLOAT* LpLoudnessMeanComplete, XFLOAT* LpBandRangeComplete,
840         XFLOAT* hulp1, XFLOAT* hulp2, int bandIdxLow, int bandIdxHigh, XFLOAT*
    aOriginalLoudnessMean, XFLOAT* aDistortedLoudnessMean, XFLOAT const1, XFLOAT const2)
841     {
842         XFLOAT mulConst1, mulConst2
843         *aOriginalLoudnessMean = 0.0
844         *aDistortedLoudnessMean = 0.0
845         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(0.0)
846         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(99.0)
847         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
848         {
```

```
849             aOriginalLoudness.m_pData[frameIndex] =
    originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    *LpBandRangeComplete, bandIdxLow, bandIdxHigh)
850             aDistortedLoudness.m_pData[frameIndex] =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    *LpBandRangeComplete, bandIdxLow, bandIdxHigh)
851             mulConst1 = aOriginalLoudness.m_pData[frameIndex]
852             mulConst2 = aDistortedLoudness.m_pData[frameIndex]
853             *aOriginalLoudnessMean += pow(mulConst1, *LpLoudnessMeanComplete)
854             *aDistortedLoudnessMean += pow(mulConst2, *LpLoudnessMeanComplete)
855         }
856     }
857
858 void LoudnessScalingOriginalTowardsDistorted2(const PolqaStatics *statics, CPOLQAData *POLQAHandle,
    CDoubleArray& aOriginalLoudnessCompletelyScaled,
859         CDoubleArray& aDistortedLoudnessCompletelyScaled, CDoubleArray& aOriginalLoudness,
    CDoubleArray& aDistortedLoudness, CBarkSpectrum& originalLoudnessDensity,
860         CBarkSpectrum& distortedLoudnessDensity, XFLOAT *aLoudnessScalingOriginal, int
    numberOfSpeechFrames, XFLOAT* LpLoudnessMeanComplete, XFLOAT* LpBandRangeComplete,
861         XFLOAT* hulp1, XFLOAT* hulp2, int bandIdxLow, int bandIdxHigh, XFLOAT*
    aOriginalLoudnessMean, XFLOAT* aDistortedLoudnessMean, XFLOAT const1, XFLOAT const2)
862     {
863         XFLOAT mulConst1, mulConst2
864         *aDistortedLoudnessMean = 0.0
865         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(0.0)
866         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(99.0)
867         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
868         {
869                 aOriginalLoudness.m_pData[frameIndex] =
    originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    *LpBandRangeComplete, bandIdxLow, bandIdxHigh)
870                 aDistortedLoudness.m_pData[frameIndex] =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
    *LpBandRangeComplete, bandIdxLow, bandIdxHigh)
871                 mulConst1 = aOriginalLoudness.m_pData[frameIndex]
872                 mulConst2 = aDistortedLoudness.m_pData[frameIndex]
873             *aOriginalLoudnessMean += pow(mulConst1, *LpLoudnessMeanComplete)
874             *aDistortedLoudnessMean += pow(mulConst2, *LpLoudnessMeanComplete)
875         }
876             *aOriginalLoudnessMean /= (numberOfSpeechFrames + 0.5)
877             *aDistortedLoudnessMean /= (numberOfSpeechFrames + 0.5)
878             *aLoudnessScalingOriginal = (pow(*aDistortedLoudnessMean, (1.0 /
    *LpLoudnessMeanComplete)) + const1) / (pow(*aOriginalLoudnessMean, (1.0 /
    *LpLoudnessMeanComplete)) + const1)
879             for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
880                 originalLoudnessDensity.MultiplyWith(frameIndex, const2)
881     }
882
883 void LoudnessScalingOriginalTowardsDistorted3(const PolqaStatics *statics, CPOLQAData *POLQAHandle,
    CDoubleArray& aOriginalLoudnessCompletelyScaled,
884         CDoubleArray& aDistortedLoudnessCompletelyScaled, CDoubleArray& aOriginalLoudness,
    CDoubleArray& aDistortedLoudness, CBarkSpectrum& originalLoudnessDensity,
885         CBarkSpectrum& distortedLoudnessDensity, XFLOAT *aLoudnessScalingOriginal, int
    numberOfSpeechFrames, XFLOAT* LpLoudnessMeanComplete, XFLOAT* LpBandRangeComplete,
886         XFLOAT* hulp1, XFLOAT* hulp2, int bandIdxLow, int bandIdxHigh, XFLOAT*
    aOriginalLoudnessMean, XFLOAT* aDistortedLoudnessMean, XFLOAT const1, XFLOAT const2)
887     {
888         XFLOAT mulConst1, mulConst2
889         *aOriginalLoudnessMean = 0.0
890         *aDistortedLoudnessMean = 0.0
891         bandIdxLow = originalLoudnessDensity.GetBandLowIdx(0.0)
892         bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(99.0)
893         for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
894         {
895             *hulp1 = originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
```

```
*LpBandRangeComplete, bandIdxLow, bandIdxHigh)
            *hulp2 = distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex,
*LpBandRangeComplete, bandIdxLow, bandIdxHigh)
            mulConst1 = *hulp1
            mulConst2 = *hulp2
            *aOriginalLoudnessMean += pow(mulConst1, *LpLoudnessMeanComplete)
            *aDistortedLoudnessMean += pow(mulConst2, *LpLoudnessMeanComplete)
        }
            *aOriginalLoudnessMean /= (numberOfSpeechFrames + 0.5)
            *aDistortedLoudnessMean /= (numberOfSpeechFrames + 0.5)
            *aLoudnessScalingOriginal = (pow(*aDistortedLoudnessMean, (1.0 /
*LpLoudnessMeanComplete)) + const1) / (pow(*aOriginalLoudnessMean, (1.0 /
*LpLoudnessMeanComplete)) + const1)
            for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++)
                originalLoudnessDensity.MultiplyWith(frameIndex, const2)
            for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++)
                {
                    aOriginalLoudnessCompletelyScaled.m_pData[frameIndex] =
originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, 2.0,
bandIdxLow, bandIdxHigh)
                    aDistortedLoudnessCompletelyScaled.m_pData[frameIndex] =
distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, 2.0,
bandIdxLow, bandIdxHigh)
                }
    }

    //START OF THE MAIN DisturbanceProcess ROUTINE

BOOL CPairParameters::DisturbanceProcess(POLQA_RESULT_DATA* pOverviewHolder)
{

    XFLOAT MeasuredSampleRate = pOverviewHolder->m_MeasuredSamplerate
    CBarkSpectrum   smearedOriginalPitchPowerDensity, smearedDistortedPitchPowerDensity
    CBarkSpectrum   originalLoudnessDensityNoInhibition, distortedLoudnessDensityNoInhibition
    CBarkSpectrum   originalPitchPowerDensityMainAvg, distortedPitchPowerDensityMainAvg
    CBarkSpectrum   originalPitchLoudnessDensityMainAvg, distortedPitchLoudnessDensityMainAvg
    CBarkSpectrum   disturbanceDensityAsymAdd
    CBarkSpectrum   mask

    CHzSpectrum     originalHzPowerSpectrum, distortedHzPowerSpectrum

    CHzSpectrum     originalHzPowerSpectrum_intact, distortedHzPowerSpectrum_intact

    CBarkSpectrum   originalPitchPowerDensity, distortedPitchPowerDensity
    CBarkSpectrum   originalPitchPowerDensity_intact, distortedPitchPowerDensity_intact
    CBarkSpectrum   originalLoudnessDensity, distortedLoudnessDensity
    CBarkSpectrum   hulp1DistortedLoudnessDensity, hulp2DistortedLoudnessDensity
    CBarkSpectrum   hulp1DistortedPitchLoudnessDensityMainAvg,
hulp2DistortedPitchLoudnessDensityMainAvg
    CBarkSpectrum   disturbanceDensity

    XFLOAT          ratioAvgCorrection, ratioUpDownAvg = 1.0, globalScaleCorrection,
globalScaleCorrectionActiveAdded
    int             numberCVCsoft, numberCVCactive
    int             numberOfSpeechFrames, numberOfFrames, numberOfSilentFrames,
numberOfNotSilentFrames, numberOfActiveFrames = 0, numberOfSuperSilentFrames,
numberOfRatioFrames
    int             numberOfaActiveFreqresponse, numberOfaSuperLoud, numberOfExtremeTimeclipFrames,
numberOfPowerRatioTimeClipFrames, numberSilentRatioOk
    int             numberActiveOk, numberActiveRatioOk, numberActiveRatioOkCorrection
    int             frameIndex, bandIdxLow, bandIdxHigh, hulpCount
    int             SNRloudnessCountTotal, SNRloudnessCountExcellent, SNRloudnessCountGood,
SNRloudnessCountFair
    int             SNRloudnessCountPoor, SNRloudnessCountBad

    int THRESHOLD_BAD_FRAMES
```

```
948     XFLOAT          powFac
949     XFLOAT          oldScale = 1, oldOldScale = 1
950     XFLOAT          scale, MaxScale, MinScale, MinMinScale, scaleOriginalFactor,
    scaleCorrectionQuality, scaleCorrectionQualityPlus, scaleCorrectionQualityPlusAdded
951     XFLOAT          scaleCorrectionQualityPlusOld, scaleCorrectionQualityPlusAddedOld
952     XFLOAT          scaleCorrectionIntell, scaleCorrectionMusic, scaleCorrectionIntellOld,
    scaleCorrectionMusicOld
953     XFLOAT          minimumOriginalFramePower,  maxDisturbance
954     XFLOAT          aPowerRatioaAvg, aPowerRatioaAvgProduct, aOriginalSilencePowerMean,
    aDistortedSilencePowerMean
955     XFLOAT          aOriginalCVCsoftPowerMean, aDistortedCVCsoftPowerMean,
    aOriginalCVCactivePowerMean, aDistortedCVCactivePowerMean
956     XFLOAT          aOriginalLoudnessMean, aDistortedLoudnessMean
957     XFLOAT          LpBandRangeLocal, aLoudnessScalingOriginal, aLoudnessScalingDistorted
958     XFLOAT          LpBandRangePartial, LpLoudnessMeanPartial, LpBandRangeComplete,
    LpLoudnessMeanComplete, LpLoudness
959     XFLOAT          fixedGlobalInternalLevel, fixedGlobalInternalLevelAdded
960     CNewStdString      s
961     int             count, count0, count1,  i
962     XFLOAT          hulp, hulp0, hulp1, hulp2, hulp3, hulp4, hulpRatio
963     XFLOAT          hulpLowOld, hulpLow, hulpHighOld, hulpHigh
964     XFLOAT          hulpLowOldNarrowband, hulpLowNarrowband, hulpHighOldNarrowband,
    hulpHighNarrowband
965
966     XFLOAT          maxDisturbanceOverFile
967     int             maxDisturbanceFrame
968
969     XFLOAT          loudnessScaleLow
970     XFLOAT          oldOldLoudnessScaleLow, oldLoudnessScaleLow
971     XFLOAT          originalLoudnessHulp, distortedLoudnessHulp
972
973     float            LTLCoeffArray1[10], LTLCoeffArray2[10]
974
975     XFLOAT          frameCorrelationTimeOriginal, frameCorrelationTimeDistorted,
    frameCorrelationTimeDisturbance
976     XFLOAT          frameCorrelationTimeOriginalOld, frameCorrelationTimeDistortedOld
977     XFLOAT          frameCorrelationTimeCompensationOriginal,
    frameCorrelationTimeCompensationDistorted, frameCorrelationTimeCompensationDifference
978     XFLOAT          frameFlatnessTimeOriginal, frameFlatnessTimeDistorted, frameFlatnessDisturbance,
    frameFlatnessDisturbanceAvg
979     XFLOAT          frameFlatnessDistortedAvgCompensationSilent,
    frameFlatnessDistortedAvgCompensationAddedSilent
980     XFLOAT          frameFlatnessDisturbanceAvgCompensationSilent,
    frameFlatnessDisturbanceAvgCompensationAddedSilent
981     XFLOAT          frameFlatnessDisturbanceAvgCompensationActive,
    frameFlatnessDisturbanceAvgCompensationAddedActive,
    frameFlatnessDisturbanceAvgCompensationActiveFrq
982     XFLOAT          frameFlatnessDisturbanceAdded
983     XFLOAT          overallDisturbance, overallMovingAvgDisturbance, overallMovingAvgDisturbanceOld,
    overallMovingAvgDisturbanceOldOld
984     XFLOAT          overallAvgDisturbance, overallAvgAddedDisturbance
985     XFLOAT          originalLoudnessTimbrePerFrame, originalLoudnessTimbrePerFrameDifferenceOld,
    originalLoudnessTimbrePerFrameDifference, originalLoudnessTimbrePerFrameDifferenceCompensation,
    originalLoudnessTimbrePerFrameDifferenceCompensationAdd
986     XFLOAT          distortedLoudnessTimbrePerFrame, distortedLoudnessTimbreHighPerFrame
987     XFLOAT          distortedLoudnessTimbrePerFrameNarrowband,
    distortedLoudnessTimbrePerFrameNarrowbandAvg, distortedLoudnessTimbrePerFrameLoudAvg,
    differenceInLoudnessTimbrePerFrame
988     XFLOAT          distortedLoudnessTimbreHighPerFrameAvg,
    distortedLoudnessTimbreHighPerFrameAvgSilent
989     XFLOAT          distortedLoudnessTimbrePerFrameDifferenceOld,
    distortedLoudnessTimbrePerFrameDifference
990     XFLOAT          avgPitchLoudFramesCompensation
991     XFLOAT          PitchRatio = 1.0
992     XFLOAT          globalScaleCorrectionIntellLevelCorrectionForMaximumD,
    globalScaleCorrectionIntellLevelCorrectionForMaximumA
993     XFLOAT          delayReliabilityPerFrameWeight, delayReliabilityPerFrameWeightOld
994     XFLOAT          number_of_sections_inserted
```

```
    XFLOAT          number_of_sections_critical
    XFLOAT          number_of_sections_invalid
    const XFLOAT OverlapCoeff = (1.0 - 0.75)

    originalHzPowerSpectrum.Initialize("originalHzPowerSpectrum", POLQAHandle)
    distortedHzPowerSpectrum.Initialize("distortedHzPowerSpectrum", POLQAHandle)

    originalHzPowerSpectrum_intact.Initialize("originalHzPowerSpectrum_intact", POLQAHandle)
    distortedHzPowerSpectrum_intact.Initialize("distortedHzPowerSpectrum_intact", POLQAHandle)

    originalPitchPowerDensity.Initialize("originalPitchPowerDensity", POLQAHandle)
    distortedPitchPowerDensity.Initialize("distortedPitchPowerDensity", POLQAHandle)
    originalPitchPowerDensity_intact.Initialize("originalPitchPowerDensity_intact", POLQAHandle)
    distortedPitchPowerDensity_intact.Initialize("distortedPitchPowerDensity_intact", POLQAHandle)
    smearedOriginalPitchPowerDensity.Initialize("smearedOriginalPitchPowerDensity", POLQAHandle)
    smearedDistortedPitchPowerDensity.Initialize("smearedDistortedPitchPowerDensity", POLQAHandle)
    originalLoudnessDensityNoInhibition.Initialize("originalLoudnessDensityNoInhibition",
POLQAHandle)
    distortedLoudnessDensityNoInhibition.Initialize("distortedLoudnessDensityNoInhibition",
POLQAHandle)

    originalLoudnessDensity.Initialize("originalLoudnessDensity", POLQAHandle)
    distortedLoudnessDensity.Initialize("distortedLoudnessDensity", POLQAHandle)
    originalPitchPowerDensityMainAvg.Initialize("originalPitchPowerDensityMainAvg", POLQAHandle)
    distortedPitchPowerDensityMainAvg.Initialize("distortedPitchPowerDensityMainAvg", POLQAHandle)
    originalPitchLoudnessDensityMainAvg.Initialize("originalPitchLoudnessDensityMainAvg",
POLQAHandle)
    distortedPitchLoudnessDensityMainAvg.Initialize("distortedPitchLoudnessDensityMainAvg",
POLQAHandle)
    disturbanceDensity.Initialize("disturbanceDensity", POLQAHandle)
    disturbanceDensityAsymAdd.Initialize("disturbanceDensityAsymAdd", POLQAHandle)
    mask.Initialize("mask", POLQAHandle)

    hulp1DistortedLoudnessDensity.Initialize("hulp1DistortedLoudnessDensity", POLQAHandle)
    hulp2DistortedLoudnessDensity.Initialize("hulp2DistortedLoudnessDensity", POLQAHandle)
    hulp1DistortedPitchLoudnessDensityMainAvg.Initialize("hulp1DistortedPitchLoudnessDensityMainAvg"
, POLQAHandle)
    hulp2DistortedPitchLoudnessDensityMainAvg.Initialize("hulp2DistortedPitchLoudnessDensityMainAvg"
, POLQAHandle)

    DisturbancePara Para(aListeningCondition, 0)

    XFLOAT maxFreqBarkLowLimit = 18
    XFLOAT maxFreqBarkHighLimit = 22


    XFLOAT delayThrMs = 3.0
    XFLOAT delayThrMsMem = 3.0
    XFLOAT hulpDelay, hulpDelayMem, hulpDelayMemOld, delayVariationCompensation,
delayVariationCompensationAdded
    hulpDelay = 9999.0
    hulpDelayMem = 9999.0
    hulpDelayMemOld = 9999.0
    pOverviewHolder->m_ConstantDelayIndicator = 0
    for (int i = (statics->startFrameIdx + 2)  i <= statics->stopFrameIdx  i++)
    {
        hulpDelayMemOld = hulpDelayMem
        hulpDelayMem = hulpDelay
        hulpDelay = 1.0*abs(pOverviewHolder->m_DelayPerFrame[i] - pOverviewHolder->m_DelayPerFrame[i
- 2])
        if (( (XFLOAT) (hulpDelay)*1000.0/ (XFLOAT) pOverviewHolder->m_SampleFrequencyHz) < 10.0)
pOverviewHolder->m_ConstantDelayIndicator += (float)( (XFLOAT) (hulpDelay)*1000.0/ (XFLOAT)
pOverviewHolder->m_SampleFrequencyHz)
    }
    numberOfSpeechFrames = statics->stopFrameIdx - statics->startFrameIdx
    pOverviewHolder->m_ConstantDelayIndicator /= ((float)numberOfSpeechFrames + 0.1F)
    constantDelayIndicator = pOverviewHolder->m_ConstantDelayIndicator
    delayVariationCompensation = 1.0 + constantDelayIndicator / 20.0
```

```
1053     delayVariationCompensationAdded = 1.0 + constantDelayIndicator / Para.delayVariationImpact
1054
1055     delayVariationCompensation = 1.0
1056     delayVariationCompensationAdded = 1.0
1057
1058
1059
1060     bool* UseThisFrame = new bool[statics->stopFrameIdx + 1]
1061
1062     int numberOf_FFLAG_SHORT_DELAY_CHANGE = 0
1063
1064     long* FrameFlags = new long[statics->stopFrameIdx + 1]
1065     {
1066         int NumDelayChanges = 0
1067         long LastDelay = aDelayUtterance.m_pData[0]
1068         int numberOfUtterances = aDelayUtterance.GetSize()
1069
1070         for (frameIndex = 0  frameIndex <= statics->stopFrameIdx  frameIndex++)
1071         {
1072             int utt = GetUtteranceForFrame(aStartSampleUtterance, aStopSampleUtterance,
     aDelayUtterance, frameIndex, aOriginalTimeSeries.GetFrameLength())
1073             UseThisFrame[frameIndex] = (utt >= 0)
1074             FrameFlags[frameIndex] = 0
1075             if (utt >= 0 && aDelayUtterance.m_pData[utt] != LastDelay)
1076             {
1077                 NumDelayChanges++
1078                 FrameFlags[frameIndex] |= 0x0000001
1079                 if (abs(aDelayUtterance.m_pData[utt] - LastDelay) < 0.001*statics->sampleRate)  {
1080                     FrameFlags[frameIndex] |= 0x0000002
1081                     numberOf_FFLAG_SHORT_DELAY_CHANGE++
1082                 }
1083
1084             }
1085             if (utt >= 0) LastDelay = aDelayUtterance.m_pData[utt]
1086         }
1087         pOverviewHolder->m_NumDelayChangesPerFrame = (XFLOAT)NumDelayChanges /
     (XFLOAT)statics->stopFrameIdx
1088     }
1089     XFLOAT compensation_FFLAG_SHORT_DELAY_CHANGE = pow((1.0 + (numberOf_FFLAG_SHORT_DELAY_CHANGE +
     1.0) / (numberOfSpeechFrames + 1.0)), 0.1)
1090
1091     PitchRatio = ComputePitchRatio_median()
1092
1093     pOverviewHolder->m_PitchRatio = PitchRatio
1094     pOverviewHolder->m_VoicedFramesRef = mNumVoicedFramesRef
1095     pOverviewHolder->m_VoicedFramesDeg = mNumVoicedFramesDeg
1096     pOverviewHolder->m_VoicedFrames = mNumVoicedFrames
1097
1098
1099
1100     originalHzPowerSpectrum.STFTPowerAndPhaseSpectrumOf(POLQAHandle, aOriginalTimeSeries,
     aStartSampleUtterance, aStopSampleUtterance, aDelayUtterance, false, false)
1101
1102     globalScaleDistortedToFixedlevelHulp = globalScaleDistortedToFixedlevel
1103     if (globalScaleDistortedToFixedlevelHulp > 9.0) globalScaleDistortedToFixedlevelHulp = 9.0
1104     if (globalScaleDistortedToFixedlevelHulp < 1.0) globalScaleDistortedToFixedlevelHulp = 1.0
1105     globalScaleDistortedToFixedlevelHulp = pow(globalScaleDistortedToFixedlevelHulp, 0.01)
1106
1107
1108
1109     distortedHzPowerSpectrum.STFTPowerAndPhaseSpectrumOf(POLQAHandle, aDistortedTimeSeries,
     aStartSampleUtterance, aStopSampleUtterance, aDelayUtterance, true, true)
1110
1111     originalHzPowerSpectrum_intact.CopySpectraFrom(originalHzPowerSpectrum)
1112     distortedHzPowerSpectrum_intact.CopySpectraFrom(distortedHzPowerSpectrum)
1113
1114     int minHzBand = int(300.0 / statics->aFrequencyResolutionHz)
1115     int maxHzBand = int(3400.0 / statics->aFrequencyResolutionHz)
```

```
1116     XFLOAT totalPowerRefBP, totalPowerDegBP
1117     const XFLOAT normFactor16bit = 32768.0 * 32768.0
1118
1119     matdbSet(-80.0, aOriginalEnvelopeBP.m_pData, statics->nrFrames)
1120     matdbSet(-80.0, aDistortedEnvelopeBP.m_pData, statics->nrFrames)
1121
1122     matdbSet(1.0, aFactorRefVsDegPowerShortTerm.m_pData, statics->nrFrames)
1123
1124     for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1125     {
1126         totalPowerRefBP = matSum(originalHzPowerSpectrum.m_pData[frameIndex] + minHzBand, maxHzBand
    - minHzBand) / XFLOAT(statics->aNumberOfHzBands)
1127         totalPowerDegBP = matSum(distortedHzPowerSpectrum.m_pData[frameIndex] + minHzBand, maxHzBand
    - minHzBand) / XFLOAT(statics->aNumberOfHzBands)
1128
1129         aOriginalEnvelopeBP.m_pData[frameIndex] = 10.0*log10(totalPowerRefBP / (normFactor16bit *
    XFLOAT(statics->frameLength)) + 0.00000001)
1130         aDistortedEnvelopeBP.m_pData[frameIndex] = 10.0*log10(totalPowerDegBP / (normFactor16bit *
    XFLOAT(statics->frameLength)) + 0.00000001)
1131         XFLOAT const minRefEnvelopeActiveSpeech = -75.0
1132         if (aOriginalEnvelopeBP.m_pData[frameIndex] <= minRefEnvelopeActiveSpeech)
1133         {
1134             aDistortedEnvelopeBP.m_pData[frameIndex] = aOriginalEnvelopeBP.m_pData[frameIndex]
1135             totalPowerDegBP = totalPowerRefBP
1136         }
1137
1138         if (pActiveFrameFlags[frameIndex] && (aOriginalEnvelopeBP.m_pData[frameIndex] >
    minRefEnvelopeActiveSpeech))
1139             aFactorRefVsDegPowerShortTerm.m_pData[frameIndex] = (totalPowerRefBP + 0.00000001) /
    (totalPowerDegBP + 0.00000001)
1140
1141     }
1142
1143     bool enoughActiveFramesForEstimate = false
1144
1145     if (pOverviewHolder->m_P56ActiveSpeechLevelDegdB - pOverviewHolder->m_P56PauseLevelDegdB > 35.0)
1146     {
1147
1148         enoughActiveFramesForEstimate =
    ComputeGainCorrectionFactorPerFrame(compensationFactorGaindBov, aDistortedEnvelopeBP,
    aOriginalEnvelopeBP, pActiveFrameFlags, statics)
1149
1150     }
1151
1152     if (enoughActiveFramesForEstimate)
1153     {
1154         XFLOAT const maxCompensationFactor = 1.3
1155         XFLOAT const minGainDiffInDBToCompensate = 0.2
1156         matdbSet(1.0, aFactorRefVsDegPowerLongerTermCorrected.m_pData, statics->nrFrames)
1157         for (int frameIndex = (statics->startFrameIdx + 2)  frameIndex <= statics->stopFrameIdx
    frameIndex++)
1158         {
1159             if (pActiveFrameFlags[frameIndex] &&
    fabs(compensationFactorGaindBov.m_pData[frameIndex]) > minGainDiffInDBToCompensate)
1160             {
1161                 aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] = 1.0 / pow(10.0,
    compensationFactorGaindBov.m_pData[frameIndex] / 10.0)
1162                 if (aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] >
    maxCompensationFactor) aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] =
    maxCompensationFactor
1163                 if (aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] < (1.0 /
    maxCompensationFactor)) aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex]
    = (1.0 / maxCompensationFactor)
1164                 hulp = (aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] + 0.5) /
    (aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex - 2] + 0.5)
1165                 if (hulp>1.0) hulp = 1 / hulp
1166                 hulp = pow(hulp, 10.0)
1167                 aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex] =
```

```
    pow(aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex], hulp)
1168            }
1169        }
1170
1171        for (int frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
1172        {
1173            if (pActiveFrameFlags[frameIndex])
1174            {
1175                for (int hzBandIndex = 0  hzBandIndex < statics->aNumberOfHzBands  hzBandIndex++)
1176                {
1177                    distortedHzPowerSpectrum.m_pData[frameIndex][hzBandIndex] *=
    aFactorRefVsDegPowerLongerTermCorrected.m_pData[frameIndex]
1178                }
1179            }
1180        }
1181
1182    }
1183
1184    CHzSpectrum *distortedHzPowerSpectrumCorrected = new CHzSpectrum
1185    distortedHzPowerSpectrumCorrected->Initialize("distortedHzPowerSpectrumCorrected", POLQAHandle)
1186    bestSpectrumShift = 0
1187    bestSpectrumShift = (int*)matMalloc((statics->stopFrameIdx + 1) * sizeof(int))
1188    XFLOAT *bestWarpingFacPerFrame = (XFLOAT*)matMalloc((statics->stopFrameIdx + 1) *
    sizeof(XFLOAT))
1189
1190    CheckTimeMatInit(POLQAHandle->mh, 3)
1191
1192
1193
1194    for (int i = statics->startFrameIdx  i <= statics->stopFrameIdx  i++)
1195    {
1196        bestSpectrumShift[i] = 0
1197        bestWarpingFacPerFrame[i] = 1.0
1198        WarpSpectrum(distortedHzPowerSpectrum.m_pData[i],
    distortedHzPowerSpectrumCorrected->m_pData[i], 1.0 / PitchRatio, statics->aNumberOfHzBands)
1199
1200        mpPitchVecDeg[i] /= PitchRatio
1201
1202        matbCopy(distortedHzPowerSpectrumCorrected->m_pData[i], distortedHzPowerSpectrum.m_pData[i],
    statics->aNumberOfHzBands)
1203
1204    }
1205
1206    if (aListeningCondition == WIDE_H) {
1207        for (int fr = POLQAHandle->statics->startFrameIdx  fr <= POLQAHandle->statics->stopFrameIdx
     fr++)
1208        PerFrameDistortionCompensation(originalHzPowerSpectrum.m_pData[fr],
    distortedHzPowerSpectrum.m_pData[fr], distortedHzPowerSpectrum.m_pData[fr], POLQAHandle, fr)
1209    }
1210
1211        PitchRatio = 1.0
1212        ShiftPitch(&originalHzPowerSpectrum, &distortedHzPowerSpectrum,
    distortedHzPowerSpectrumCorrected, pActiveFrameFlags, bestSpectrumShift,
    bestWarpingFacPerFrame)
1213
1214        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++)
1215            matbCopy(distortedHzPowerSpectrumCorrected->m_pData[fr],
    distortedHzPowerSpectrum.m_pData[fr], statics->aNumberOfHzBands)
1216        delete distortedHzPowerSpectrumCorrected
1217        distortedHzPowerSpectrumCorrected = NULL
1218
1219        CheckTimeMatEval(POLQAHandle->mh, 3, &ClockCycles, &TimeDiff)
1220        AddProcessingTime(pOverviewHolder, "Spectrum correction", TimeDiff, ClockCycles)
1221
1222
1223
1224        XFLOAT SampleRateRatio = pOverviewHolder->m_MeasuredSamplerate / statics->sampleRate
```

```
1225        if (SampleRateRatio < 0.0) SampleRateRatio = 1.0
1226
1227        if (SampleRateRatio<Para.MinSRR) SampleRateRatio = Para.MinSRR
1228        if (SampleRateRatio>Para.MaxSRR) SampleRateRatio = Para.MaxSRR
1229
1230        originalPitchPowerDensity.FrequencyWarpingOf(POLQAHandle, originalHzPowerSpectrum, 1.0)
1231        distortedPitchPowerDensity.FrequencyWarpingOf(POLQAHandle, distortedHzPowerSpectrum,
    PitchRatio)
1232
1233        if (aSampleFrequencyHzSource < (2650.0*2.0))
1234        {
1235            maxFreqBarkSource = -1.8585e-6*aSampleFrequencyHzSource*aSampleFrequencyHzSource / 4.0 +
    10.263*aSampleFrequencyHzSource / 2.0 + 0.1203
1236        }
1237        else
1238        {
1239            maxFreqBarkSource = 4.9289*log(aSampleFrequencyHzSource / 2.0) / log(2.718281828) -
    23.463
1240        }
1241        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1242        {
1243            originalPitchPowerDensity.MultiplyWithOverBandRange(frameIndex, 0.0, maxFreqBarkSource,
    99.0)
1244            distortedPitchPowerDensity.MultiplyWithOverBandRange(frameIndex, 0.0, maxFreqBarkSource,
    99.0)
1245        }
1246        if (aListeningCondition == STANDARD_IRS) maxFreqBarkSource = 17
1247
1248    ShortLevelVariations(&LevelVariation,statics, originalPitchPowerDensity,
    distortedPitchPowerDensity,pOverviewHolder)
1249
1250        if (maxFreqBarkSource < maxFreqBarkHighLimit) {
1251            if (maxFreqBarkSource < maxFreqBarkLowLimit) {
1252                LpBandRangeLocal = 1.2
1253                LpBandRangePartial = 2.3
1254                LpLoudnessMeanPartial = 0.7
1255                LpBandRangeComplete = 1.6
1256                LpLoudnessMeanComplete = 1.45
1257                LpLoudness = 2.2
1258                fixedGlobalInternalLevel = 20.0
1259                fixedGlobalInternalLevelAdded = 15.0
1260            }
1261            else {
1262                LpBandRangeLocal = 1.2
1263                LpBandRangePartial = 2.1
1264                LpLoudnessMeanPartial = 0.65
1265                LpBandRangeComplete = 1.9
1266                LpLoudnessMeanComplete = 1.45
1267                LpLoudness = 2.1
1268                fixedGlobalInternalLevel = 22.0
1269                fixedGlobalInternalLevelAdded = 14.0
1270            }
1271        } else {
1272            LpBandRangeLocal = 1.2
1273            LpBandRangePartial = 2.1
1274            LpLoudnessMeanPartial = 0.5
1275            LpBandRangeComplete = 1.9
1276            LpLoudnessMeanComplete = 1.5
1277            LpLoudness = 2.2
1278            fixedGlobalInternalLevel = 23.0
1279            fixedGlobalInternalLevelAdded = 14.0
1280        }
1281        aAvgOriginalPower = 0.0
1282        aAvgDistortedPower = 0.0
1283        numberOfFrames = 0
1284        numberCVCsoft = 0
1285        numberCVCactive = 0
1286        aOriginalCVCsoftPowerMean = 0.0
```

```
1287        aDistortedCVCsoftPowerMean = 0.0
1288        aOriginalCVCactivePowerMean = 0.0
1289        aDistortedCVCactivePowerMean = 0.0
1290        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1291        {
1292            aOriginalTotalPower.m_pData[frameIndex] = originalPitchPowerDensity.Total(frameIndex,
    300.0, 3500.0)
1293            aDistortedTotalPower.m_pData[frameIndex] = distortedPitchPowerDensity.Total(frameIndex,
    300.0, 3500.0)
1294            numberOfFrames++
1295            aAvgOriginalPower += aOriginalTotalPower.m_pData[frameIndex]
1296            aAvgDistortedPower += aDistortedTotalPower.m_pData[frameIndex]
1297            hulp1 = originalPitchPowerDensity.Total(frameIndex, 300.0, 3500.0)
1298            hulp2 = distortedPitchPowerDensity.Total(frameIndex, 300.0, 3500.0)
1299            if ((hulp1<7.0E7) && (hulp1>2.0E7))
1300            {
1301                numberCVCsoft++
1302                aOriginalCVCsoftPowerMean += hulp1
1303                aDistortedCVCsoftPowerMean += hulp2
1304            }
1305            if ((hulp1<9.0E7) && (hulp1>2.0E5))
1306            {
1307                numberCVCactive++
1308                aOriginalCVCactivePowerMean += hulp1
1309                aDistortedCVCactivePowerMean += hulp2
1310            }
1311        }
1312        aOriginalCVCsoftPowerMean /= (numberCVCsoft + 0.01)
1313        aDistortedCVCsoftPowerMean /= (numberCVCsoft + 0.01)
1314        aOriginalCVCactivePowerMean /= (numberCVCactive + 0.01)
1315        aDistortedCVCactivePowerMean /= (numberCVCactive + 0.01)
1316        CVCratioSNRlevelRangecompensation0001 = ((aDistortedCVCsoftPowerMean + 1.0) /
    (aDistortedCVCactivePowerMean + 1.0) + Para.loudNoiseP1delta) / ((aOriginalCVCsoftPowerMean
    + 1.0) / (aOriginalCVCactivePowerMean + 1.0) + Para.loudNoiseP1delta)
1317        if (CVCratioSNRlevelRangecompensation0001<1.0)
1318        {
1319            CVCratioSNRlevelRangecompensation0001 += Para.loudNoiseP2add
1320            if (CVCratioSNRlevelRangecompensation0001>1.0) CVCratioSNRlevelRangecompensation0001 =
    1.0
1321            CVCratioSNRlevelRangecompensation0001 = pow(CVCratioSNRlevelRangecompensation0001,
    Para.loudNoiseP3pow)
1322        }
1323        else
1324        {
1325            CVCratioSNRlevelRangecompensation0001 = 1.0
1326        }
1327
1328        aAvgOriginalPower /= (numberOfFrames + 0.01)
1329        aAvgDistortedPower /= (numberOfFrames + 0.01)
1330
1331        aAvgActiveOriginalPower = 0.0
1332        numberOfFrames = 0
1333        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1334        {
1335            if (aOriginalTotalPower.m_pData[frameIndex] > aAvgOriginalPower / 100.0)
1336            {
1337                aAvgActiveOriginalPower += aOriginalTotalPower.m_pData[frameIndex]
1338                numberOfFrames++
1339            }
1340        }
1341        aAvgActiveOriginalPower /= (numberOfFrames + 0.01)
1342
1343        aAvgActiveDistortedPower = 0.0
1344        numberOfFrames = 0
1345        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1346        {
1347            if (aDistortedTotalPower.m_pData[frameIndex] > aAvgDistortedPower / 100.0)
1348            {
```

```
1349                aAvgActiveDistortedPower += aDistortedTotalPower.m_pData[frameIndex]
1350                numberOfFrames++
1351            }
1352        }
1353        aAvgActiveDistortedPower /= (numberOfFrames + 0.01)
1354        globalScaleCorrection = (aAvgActiveDistortedPower + 0.01) / (aAvgActiveOriginalPower + 0.01)
1355        globalScaleCorrectionIntellLevelCorrection = (aAvgActiveDistortedPower + 1.0) /
    (aAvgActiveOriginalPower + 1.0)
1356
1357        XFLOAT Power1
1358        XFLOAT Power2
1359
1360        if (globalScaleCorrectionIntellLevelCorrection < 0.1)
    globalScaleCorrectionIntellLevelCorrection = 0.1
1361        if (globalScaleCorrectionIntellLevelCorrection < 1.0)
1362        {
1363            Power1 = Para.IntellLevelCorrectionP1max
1364            Power2 = Para.IntellLevelCorrectionP2max
1365        }
1366
1367        else
1368        {
1369            Power1 = Para.IntellLevelCorrectionP3max
1370            Power2 = Para.IntellLevelCorrectionP4max
1371        }
1372
1373            globalScaleCorrectionIntellLevelCorrectionForMaximumD =
    pow(globalScaleCorrectionIntellLevelCorrection, Power1)
1374            globalScaleCorrectionIntellLevelCorrectionForMaximumA =
    pow(globalScaleCorrectionIntellLevelCorrection, Power2)
1375
1376        //Determine silent interactive etc.... intervals
1377        aPowerRatioaAvg = 0.0
1378        aPowerRatioaAvgProduct = 1.0
1379        aOriginalSilencePowerMean = 0.0
1380        aDistortedSilencePowerMean = 0.0
1381
1382        numberOfaActiveFreqresponse = 0
1383        numberOfaSuperLoud = 0
1384        extremeTimeclipFramesCompensation000 = 0
1385        numberOfPowerRatioTimeClipFrames = 0
1386        numberSilentRatioOk = 0
1387        numberActiveOk = 0
1388        numberActiveRatioOk = 0
1389        numberActiveRatioOkCorrection = 0
1390        numberOfSilentFrames = 0
1391        numberOfNotSilentFrames = 0
1392        numberOfSuperSilentFrames = 0
1393        ratioAvgCorrection = 0.0
1394        envelopeContinuityCompensation000 = 0.0
1395        numberOfRatioFrames = 0
1396        numberOfExtremeTimeclipFrames = 0
1397
1398        count0 = 0
1399        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1400        {
1401
1402            if (aOriginalTotalPower.m_pData[frameIndex] < Para.SilentCriterium)
1403            {
1404                numberOfSilentFrames++
1405                aSilent.m_pData[frameIndex] = TRUE
1406                aOriginalSilencePowerMean += aOriginalTotalPower.m_pData[frameIndex]
1407                aDistortedSilencePowerMean += aDistortedTotalPower.m_pData[frameIndex]
1408            }
1409            else
1410            {
1411                numberOfNotSilentFrames++
1412                aSilent.m_pData[frameIndex] = FALSE
```

```
1413            }
1414
1415            aPowerRatio.m_pData[frameIndex] = 1.0  {
1416            if (aOriginalTotalPower.m_pData[frameIndex] > 1.0E7)
1417                aPowerRatio.m_pData[frameIndex] = (aOriginalTotalPower.m_pData[frameIndex] + 100.0)
    / (aDistortedTotalPower.m_pData[frameIndex] + 100.0)
1418                if (aPowerRatio.m_pData[frameIndex] < 1.0) aPowerRatio.m_pData[frameIndex] = 1.0
1419                aPowerRatioaAvgProduct /= pow(aPowerRatio.m_pData[frameIndex], 0.0008)
1420                if ( (aPowerRatio.m_pData[frameIndex] > 10000.0) && (frameIndex >
    (statics->startFrameIdx + 9)) ) {
1421                    if (  (aPowerRatio.m_pData[frameIndex - 9] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 8] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 7] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 6] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 5] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 4] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 3] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 2] > 10000.0) &&
    (aPowerRatio.m_pData[frameIndex - 1] > 1000.0)  ) {
1422                        aPowerRatioaAvg += 1 / aPowerRatio.m_pData[frameIndex]
1423                        numberOfExtremeTimeclipFrames++
1424                    }
1425                }
1426                if ((aOriginalTotalPower.m_pData[frameIndex] > 5.0e7) &&
    ((aOriginalTotalPower.m_pData[frameIndex] + 3000.0) /
    (aDistortedTotalPower.m_pData[frameIndex] + 3000.0) > 50.0))
    numberOfPowerRatioTimeClipFrames++
1427                numberOfRatioFrames++
1428            }
1429            if (frameIndex > (statics->startFrameIdx + 2)) hulp = (aPowerRatio.m_pData[frameIndex] +
    0.1) / (aPowerRatio.m_pData[frameIndex - 2] + 0.1)  else hulp = 1.0
1430            if (hulp>1.0) ratioUpDownAvg *= hulp  else ratioUpDownAvg /= hulp
1431
1432            if (aOriginalTotalPower.m_pData[frameIndex] < 2.0E5)
1433            {
1434                numberOfSuperSilentFrames++
1435                aSuperSilent.m_pData[frameIndex] = TRUE
1436            }
1437            else
1438            {
1439                aSuperSilent.m_pData[frameIndex] = FALSE
1440            }
1441
1442            if (aOriginalTotalPower.m_pData[frameIndex] > 9.0E6)
1443            {
1444                aActiveFreqresponse.m_pData[frameIndex] = TRUE
1445                numberOfaActiveFreqresponse++
1446            }
1447            else
1448            {
1449                aActiveFreqresponse.m_pData[frameIndex] = FALSE
1450            }
1451
1452            if (aOriginalTotalPower.m_pData[frameIndex] > 7.0E8)
1453            {
1454                aSuperLoud.m_pData[frameIndex] = TRUE
1455                numberOfaSuperLoud++
1456            }
1457            else
1458            {
1459                aSuperLoud.m_pData[frameIndex] = FALSE
1460            }
1461
1462            if (aOriginalTotalPower.m_pData[frameIndex] > 2.0E7)
1463            {
1464                aActiveFreqresponseIntell.m_pData[frameIndex] = TRUE
1465            }
1466            else
```

```
            {
                aActiveFreqresponseIntell.m_pData[frameIndex] = FALSE
            }

            aSilentRatioOk.m_pData[frameIndex] = FALSE
            aActiveRatioOk.m_pData[frameIndex] = FALSE
            if (aOriginalTotalPower.m_pData[frameIndex] < 3.0E6)
            {
                hulp = (aDistortedTotalPower.m_pData[frameIndex] + 1.0) /
(globalScaleCorrection*aOriginalTotalPower.m_pData[frameIndex] + 1.0)
                if (hulp < 1.0e4)
                {
                    numberSilentRatioOk++
                    aSilentRatioOk.m_pData[frameIndex] = TRUE
                }
            }
            else
            {
                numberActiveOk++
                hulp = (aDistortedTotalPower.m_pData[frameIndex] + 10.0) /
(globalScaleCorrection*aOriginalTotalPower.m_pData[frameIndex] + 10.0)
                if ((0.12 < hulp) && (hulp < 8.0))
                {
                    numberActiveRatioOk++
                    aActiveRatioOk.m_pData[frameIndex] = TRUE
                }
                hulp = (aDistortedTotalPower.m_pData[frameIndex] + 10.0) /
(globalScaleCorrection*aOriginalTotalPower.m_pData[frameIndex] + 10.0)
                if ((0.3<hulp) && (hulp<3.3))
                {
                    numberActiveRatioOkCorrection++
                }
            }

            if (frameIndex>(statics->startFrameIdx + 31))
            {
                count1 = 0
                for (i = 0  i <= 28  i++) if (aOriginalTotalPower.m_pData[frameIndex - i]>2.0e7)
count1++
                if (count1 > 25.0)
                {
                    count0++
                    hulp1 = 0.0
                    hulp2 = 0.0
                    hulp3 = 0.0
                    hulp4 = 0.0
                    for (i = 0  i <= 12  i++)
                    {
                        hulp1 += aOriginalTotalPower.m_pData[frameIndex - i - 12]
                        hulp3 += aDistortedTotalPower.m_pData[frameIndex - i - 12]
                        hulp2 += aOriginalTotalPower.m_pData[frameIndex - i]
                        hulp4 += aDistortedTotalPower.m_pData[frameIndex - i]
                    }
                    envelopeContinuityCompensation000 += fabs((hulp1 + 5.0e9) / (hulp2 + 5.0e9) -
(hulp3 + 5.0e9) / (hulp4 + 5.0e9))
                }
            }
        }
        ratioUpDownAvg = pow(ratioUpDownAvg, 1 / (numberOfRatioFrames + 1.0))

        aOriginalSilencePowerMean /= (numberOfSilentFrames + 0.01)
        aDistortedSilencePowerMean /= (numberOfSilentFrames + 0.01)
        hulp = (numberOfNotSilentFrames - 20.0)
        if (hulp<0.0) hulp = 0.0
        hulp /= 50000.0
        if (numberOfNotSilentFrames>100)
        {
            aPowerRatioaAvgTimeClipCompensation000 = pow(((100 + 1.0) /
```

```
    (numberOfPowerRatioTimeClipFrames + 1.0)), hulp)
1530        }
1531        else
1532        {
1533            aPowerRatioaAvgTimeClipCompensation000 = pow(((numberOfNotSilentFrames + 1.0) /
    (numberOfPowerRatioTimeClipFrames + 1.0)), hulp)
1534        }
1535        if (numberOfPowerRatioTimeClipFrames < 5) aPowerRatioaAvgTimeClipCompensation000 = 1.0
1536        aPowerRatioaAvg /= (numberOfRatioFrames + 0.01)
1537        envelopeContinuityCompensation000 /= (count0 + 1.0)
1538        envelopeContinuityCompensation000 /= 5.0
1539        extremeTimeclipFramesCompensation000 = numberOfExtremeTimeclipFrames / (numberOfRatioFrames
    + 1.0)
1540
1541        XFLOAT  aDistortedSilencePowerMeanCompensation = pow((aDistortedSilencePowerMean + 1.0),
    0.01)
1542        aDistortedSilencePowerMeanCompensation *= Para.DistortedSilencePowerMeanCompensationFac
1543
1544        //END Determine silent interactive etc.... intervals
1545
1546        //GLOBAL SCALE original based on all active frames
1547        hulp1 = 0.0
1548        hulp2 = 0.0
1549        hulpCount = 0
1550        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1551        {
1552            if (!aSilent.m_pData[frameIndex])
1553            {
1554                hulp1 += originalPitchPowerDensity.Total(frameIndex, 350.0, 3500.0)
1555                hulp2 += distortedPitchPowerDensity.Total(frameIndex, 350.0, 3500.0)
1556                hulpCount++
1557            }
1558        }
1559        hulp1 /= (hulpCount + 1.0)
1560        hulp2 /= (hulpCount + 1.0)
1561        scaleOriginalFactor = (hulp2 + 1.0e4) / (hulp1 + 1.0e4)
1562        if (scaleOriginalFactor < 0.03) scaleOriginalFactor = scaleOriginalFactor + pow((0.03 -
    scaleOriginalFactor), 1.5)
1563
1564        frameFlatnessDistortedAvgCompensationSilent = 0.01
1565        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1566        {
1567            originalPitchPowerDensity.MultiplyWith(frameIndex, scaleOriginalFactor)
1568            if (aSuperSilent.m_pData[frameIndex]) frameFlatnessDistortedAvgCompensationSilent +=
    distortedPitchPowerDensity.SpectralFlatnessPower(frameIndex)
1569        }
1570        frameFlatnessDistortedAvgCompensationSilent /= (numberOfSuperSilentFrames + 0.01)
1571        frameFlatnessDistortedAvgCompensationSilent000 = frameFlatnessDistortedAvgCompensationSilent
1572        if (frameFlatnessDistortedAvgCompensationSilent000<0.0004)
    frameFlatnessDistortedAvgCompensationSilent000 = 0.0004
1573        if (frameFlatnessDistortedAvgCompensationSilent000>0.007)
    frameFlatnessDistortedAvgCompensationSilent000 = 0.007
1574        frameFlatnessDistortedAvgCompensationSilent000 = 1.0 /
    frameFlatnessDistortedAvgCompensationSilent000 - 1.0 / 0.007
1575        frameFlatnessDistortedAvgCompensationSilent000 =
    frameFlatnessDistortedAvgCompensationSilent000 / 1700.0
1576        frameFlatnessDistortedAvgCompensationAddedSilent =
    pow((frameFlatnessDistortedAvgCompensationSilent + 0.3), 0.01)
1577        frameFlatnessDistortedAvgCompensationSilent =
    pow((frameFlatnessDistortedAvgCompensationSilent + 0.3), 0.01)
1578
1579        globalScaleCorrectionActive = (hulp2 + 1.0e4) / (hulp1 + 1.0e4)
1580        globalScaleCorrectionActiveAdded = pow(globalScaleCorrectionActive, 0.07)
1581        if (globalScaleCorrectionActive > 1.0)
1582        {
1583            globalScaleCorrectionActive = pow(globalScaleCorrectionActive,
    Para.GlobalScaleCorrectionActiveP1pow)
1584        }
```

```
1585        else
1586        {
1587            globalScaleCorrectionActive = pow(globalScaleCorrectionActive,
    Para.GlobalScaleCorrectionActiveP2pow)
1588        }
1589
1590        //END GLOBAL SCALE original
1591
1592
1593
1594        //FREQ Indicator
1595
1596
1597
1598        aPureFrqLoudnessMean = CalculateFrequencyIndicator_SpeechLQ(POLQAHandle,
    originalPitchPowerDensity_intact, distortedPitchPowerDensity_intact, aOriginalTotalPower,
    aDistortedTotalPower, aActiveRatioOk.m_pData, aGlobalCompensation1, aPowerRatioaAvgProduct,
    aSilent, numberOfSilentFrames, Para.PureFreqP1, statics, aResultsFile)
1599        XFLOAT aPureFrqLoudnessMeanCompensation = pow((aPureFrqLoudnessMean + 1.0), 0.02)
1600
1601        //END FREQ Indicator
1602
1603
1604        //POLQAMAIN PART 0
1605        CheckTimeMatInit(POLQAHandle->mh, 3)
1606
1607        superLowbandCorrection = 0.0
1608        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1609        {
1610            hulp1 = originalPitchPowerDensity.Total(frameIndex, 0.0, 50.0)
1611            hulp2 = distortedPitchPowerDensity.Total(frameIndex, 0.0, 50.0)
1612            hulp = abs(pow(hulp2, 0.22) - pow(hulp1, 0.22))
1613            superLowbandCorrection += hulp
1614        }
1615        superLowbandCorrection /= (numberOfSpeechFrames + 0.01)
1616        superLowbandCorrection /= 900.0
1617
1618        superWidebandCorrection = 0.0
1619        hulp = 0.0
1620        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1621        {
1622            hulp1 = originalPitchPowerDensity_intact.Total(frameIndex, 8000.0, 14000.0)
1623            hulp2 = distortedPitchPowerDensity_intact.Total(frameIndex, 8000.0, 14000.0)
1624            hulp = abs(pow(hulp1, 0.05) - pow(hulp2, 0.05))
1625            superWidebandCorrection += hulp2
1626        }
1627        superWidebandCorrection /= (numberOfSpeechFrames + 0.01)
1628
1629        fullbandCorrection = 0.0
1630
1631        if (aSampleFrequencyHzSource>35000) {
1632        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1633        {
1634            hulp1 = originalPitchPowerDensity.Total(frameIndex, 14000.0, aSampleFrequencyHzSource /
    2.0)
1635            hulp2 = distortedPitchPowerDensity.Total(frameIndex, 14000.0, aSampleFrequencyHzSource /
    2.0)
1636
1637            if (hulp2>1.0) hulp = pow(hulp2, 0.1)
1638            fullbandCorrection += hulp2
1639        }
1640        }
1641        fullbandCorrection /= (numberOfSpeechFrames + 0.01)
1642
1643        aScale.Initialize("aScale", mMaxModelFrames)
1644        int numberOfUsedFrames = 0
1645        count = 0
1646        scaleDistortion = 0.0
```

```
        oldScale = 1.0
        minimumOriginalFramePower = 10000000.0
        MaxScale = 1.0
        MinScale = 1.0
        MinMinScale = 0.5
        for (frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
frameIndex++)
        {
            if (UseThisFrame[frameIndex])
            {
                aOriginalTotalPower.m_pData[frameIndex] =
originalPitchPowerDensity.Total(frameIndex, 0.0, 10000.0)
                aDistortedTotalPower.m_pData[frameIndex] =
distortedPitchPowerDensity.Total(frameIndex, 0.0, 10000.0)
                if (aOriginalTotalPower.m_pData[frameIndex] < minimumOriginalFramePower) {
minimumOriginalFramePower = aOriginalTotalPower.m_pData[frameIndex]  }
                scale = (aDistortedTotalPower.m_pData[frameIndex] + (XFLOAT) 4.0e4) /
(aOriginalTotalPower.m_pData[frameIndex] + (XFLOAT) 4.0e4)
                if (frameIndex>10 && aActiveFreqresponse.m_pData[frameIndex] && oldScale > scale)
                {
                    scaleDistortion += fabs(oldScale - scale)
                    count++
                }

                if (scale > MaxScale) scale = MaxScale
                if (scale < MinMinScale) scale = MinMinScale

                aScale.m_pData[frameIndex] = scale
                scale = Para.LocScaleP1 * oldOldScale + Para.LocScaleP2 * oldScale + Para.LocScaleP3
* scale
                oldOldScale = oldScale
                oldScale = scale
                originalPitchPowerDensity.MultiplyWith(frameIndex, pow(scale,
Para.LocScaleP4pow*globalScaleDistortedToFixedlevelHulp))
                numberOfUsedFrames++
            }
            else
            {
                originalPitchPowerDensity.MultiplyWith(frameIndex, 0.0)
                distortedPitchPowerDensity.MultiplyWith(frameIndex, 0.0)
            }
        }
        scaleDistortion /= (count + 0.1)
        XFLOAT fractionOfUsedFrames
        if (numberOfUsedFrames > statics->startFrameIdx)
        {
            fractionOfUsedFrames = numberOfUsedFrames / (numberOfSpeechFrames + 1.0)
        }
        else
        {
            fractionOfUsedFrames = 1.0
        }
        XFLOAT fractionOfSilentFrames = (numberOfSilentFrames + 1.0) / (numberOfUsedFrames + 1.0)
        if (fractionOfSilentFrames > 0.5) fractionOfSilentFrames = 0.5

        XFLOAT FBconst1, FBconst2, FBconst3
        if (maxFreqBarkSource < maxFreqBarkHighLimit) {
            if (maxFreqBarkSource<maxFreqBarkLowLimit) {
                FBconst1 = 0.5
                FBconst2 = 2.0E4
                FBconst3 = 0.65
            } else {
                FBconst1 = 0.5
                FBconst2 = 2.0E4
                FBconst3 = 0.65
            }
        } else {
            FBconst1 = 0.5
```

```
            FBconst2 = 7.0E3
            FBconst3 = 0.65

        }

        FreqResponseCompensPowerDomain(statics, POLQAHandle, originalPitchPowerDensityMainAvg,
distortedPitchPowerDensityMainAvg, originalPitchPowerDensity,
            distortedPitchPowerDensity, aActiveFreqresponseIntell, FBconst1, FBconst1, FBconst3,
FBconst2, 1)

        smearedOriginalPitchPowerDensity.ExcitationOf(POLQAHandle, originalPitchPowerDensity,
UseThisFrame, statics->listeningCondition)
        smearedDistortedPitchPowerDensity.ExcitationOf(POLQAHandle, distortedPitchPowerDensity,
UseThisFrame, statics->listeningCondition)

        originalLoudnessDensityNoInhibition.IntensityWarpingOfSpeechLQ(POLQAHandle,
smearedOriginalPitchPowerDensity, maxFreqBarkSource)
        distortedLoudnessDensityNoInhibition.IntensityWarpingOfSpeechLQ(POLQAHandle,
smearedDistortedPitchPowerDensity, maxFreqBarkSource)

        originalLoudnessDensity.InhibitionSpeechLQ(POLQAHandle, originalLoudnessDensityNoInhibition,
maxFreqBarkSource)
        distortedLoudnessDensity.InhibitionSpeechLQ(POLQAHandle,
distortedLoudnessDensityNoInhibition, maxFreqBarkSource)

        //END POLQAMAIN PART 0


        //NOISE Indicator



        XFLOAT noiseIndicatorTimbre, noiseIndicatorTimbreAdd
        XFLOAT noiseIndicator, noiseIndicatorFreqImpact, noiseIndicatorScalingImpact
        XFLOAT delayJumpCompNB, delayJumpCompWB
        int noiseIndicatorAlignJumpsIntWB

        noiseIndicatorAlignJumpsIntWB = 5

        noiseIndicator = CalculateNoiseIndicator_SpeechLQ(statics, POLQAHandle,
originalPitchPowerDensity_intact, distortedPitchPowerDensity_intact, aActiveRatioOk.m_pData,
                numberOfSilentFrames, numberOfNotSilentFrames, aActiveRatioOk, aSilent,
aAddedSilentDisturbance, &noiseIndicatorTimbre, &noiseIndicatorTimbreAdd,
                &noiseIndicatorFreqImpact, &noiseIndicatorScalingImpact,  &delayJumpCompNB,
&delayJumpCompWB, aActive,
                originalLoudnessDensity, distortedLoudnessDensity,
originalPitchLoudnessDensityMainAvg, distortedPitchLoudnessDensityMainAvg,
&noiseIndicatorHighBandsCompensation000)

        //END NOISE Indicator



        CheckTimeMatInit(POLQAHandle->mh, 3)
        //POLQAMAIN PART 1
        disturbanceDensity.DifferenceOf(distortedLoudnessDensity, originalLoudnessDensity)
        mask.MinimumOf(distortedLoudnessDensity, originalLoudnessDensity)
        mask *= (XFLOAT) 0.9
        disturbanceDensity.MaskWith(POLQAHandle, mask)

        XFLOAT noiseTransparencyWeightIndicatorOrg = 0.0
        XFLOAT noiseTransparencyWeightIndicatorDis = 0.0
        XFLOAT disturbanceTransparencyWeightIndicator = 0.0
        XFLOAT digitalSilenceCompensationFactor = 0.0
        int    digitalSilenceCount = 0
        bandIdxLow = originalLoudnessDensity.GetBandLowIdx(0.0)
        bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(99.0)
```

```
1763        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1764        {
1765            hulp = (originalLoudnessDensity.IntegralLpOverBandRange<2>(POLQAHandle, frameIndex,
    bandIdxLow, bandIdxHigh) + 0.01)
1766            if (hulp < 0.2) hulp = 0.2
1767            disturbanceTransparencyWeightIndicator +=
    fabs(disturbanceDensity.IntegralLpOverBandRange<2>(POLQAHandle, frameIndex, bandIdxLow,
    bandIdxHigh))/hulp
1768            if (aSuperSilent.m_pData[frameIndex])
1769            {
1770                noiseTransparencyWeightIndicatorOrg +=
    originalLoudnessDensity.IntegralLpOverBandRange<2>(POLQAHandle, frameIndex,
    bandIdxLow, bandIdxHigh)
1771                noiseTransparencyWeightIndicatorDis +=
    distortedLoudnessDensity.IntegralLpOverBandRange<2>(POLQAHandle, frameIndex,
    bandIdxLow, bandIdxHigh)
1772                if (aOriginalTotalPower.m_pData[frameIndex] < 0.2) {
1773                    digitalSilenceCompensationFactor +=
    originalLoudnessDensity.IntegralLpOverBandRange<2>(POLQAHandle, frameIndex,
    bandIdxLow, bandIdxHigh)
1774                    digitalSilenceCount++
1775                }
1776            } else {
1777            }
1778        }
1779        noiseTransparencyWeightIndicatorOrg /= (numberOfSuperSilentFrames + 0.01)
1780        noiseTransparencyWeightIndicatorDis /= (numberOfSuperSilentFrames + 0.01)
1781        disturbanceTransparencyWeightIndicator /= ((numberOfSpeechFrames) + 0.01)
1782        digitalSilenceCompensationFactor = digitalSilenceCount
1783        if (digitalSilenceCompensationFactor > 70.0) digitalSilenceCompensationFactor = 70.0
1784        digitalSilenceCompensationFactor /= 500.0
1785        if (disturbanceTransparencyWeightIndicator < 0.0) disturbanceTransparencyWeightIndicator =
    0.0
1786        XFLOAT alpha_noise
1787        alpha_noise = pow(disturbanceTransparencyWeightIndicator,0.08)
1788        if (alpha_noise > 1.0) alpha_noise = 1.0
1789        if (alpha_noise < 0.0) alpha_noise = 0.0
1790
1791        aDistortedLoudnessMeanIndicator1 = 0.0
1792        distortedLoudnessTimbrePerFrameLoudAvg = 0.0
1793        bandIdxLow = originalLoudnessDensity.GetBandLowIdx(0.0)
1794        bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(99.0)
1795        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1796        {
1797
1798            aOriginalLoudness.m_pData[frameIndex] =
    originalLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, 2.0,
    bandIdxLow, bandIdxHigh)
1799            aDistortedLoudness.m_pData[frameIndex] =
    distortedLoudnessDensity.IntegralLpOverBandRange(POLQAHandle, frameIndex, 2.0,
    bandIdxLow, bandIdxHigh)
1800
1801            aDistortedLoudnessMeanIndicator1 += pow(aDistortedLoudness.m_pData[frameIndex], 0.3)
1802            if (aSuperLoud.m_pData[frameIndex])
1803            {
1804                hulp1 = (distortedLoudnessDensity.IntegralLowFrameLoud(frameIndex,
    statics->listeningCondition) -
    distortedLoudnessDensity.IntegralHighFrameLoud(frameIndex,
    statics->listeningCondition))
1805                distortedLoudnessTimbrePerFrameLoudAvg += hulp1
1806            }
1807
1808        }
1809        aDistortedLoudnessMeanIndicator1 /= ( numberOfSpeechFrames + 0.01)
1810        aDistortedLoudnessMeanIndicator1 = pow(aDistortedLoudnessMeanIndicator1,(1.0/0.3))
1811        distortedLoudnessTimbrePerFrameLoudAvg /= (numberOfaSuperLoud+0.1)
1812        distortedLoudnessTimbrePerFrameLoudAvg000 = (distortedLoudnessTimbrePerFrameLoudAvg / 2300.0
    + 0.01)
```

```
1813
1814        originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity,
    aSuperSilent, 4.0, numberOfSuperSilentFrames)
1815        distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
    aSuperSilent, 4.0, numberOfSuperSilentFrames)
1816
1817        originalLoudnessDensity.AudibleNoiseRespCompensationOfPartly(POLQAHandle,
    originalPitchLoudnessDensityMainAvg, (0.31 - digitalSilenceCompensationFactor) /
    noiseIndicatorFreqImpact)
1818
1819        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
1820            matbCopy(distortedLoudnessDensity.m_pData[fr],
    hulp1DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.aNumberOfBands)
1821            matbCopy(distortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.aNumberOfBands)
1822        }
1823        hulp1DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly(POLQAHandle,
    distortedPitchLoudnessDensityMainAvg, (0.31 - digitalSilenceCompensationFactor) /
    noiseIndicatorFreqImpact)
1824        hulp2DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly2(distortedPitchLoudnessDe
    nsityMainAvg, (0.34 - digitalSilenceCompensationFactor)*delayVariationCompensation, 1.0)
1825        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
1826            matbMpy1(1 - alpha_noise, hulp1DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
1827            matbMpy1(alpha_noise, hulp2DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
1828        }
1829        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++)
1830            matbAdd3(hulp1DistortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
1831
1832        //Local loudness scaling original, predominantly for modelling the impact of time clip
    effects during active intervals
1833        oldOldLoudnessScaleLow = 1.0
1834        oldLoudnessScaleLow = 1.0
1835        bandIdxLow = originalLoudnessDensity.GetBandLowIdx(Para.LocLoudScaleBandLowTimeclip)
1836        bandIdxHigh = originalLoudnessDensity.GetBandHighIdx(Para.LocLoudScaleBandHighTimeclip)
1837
1838        LoudnessScalingOriginalClipEffect1(statics, POLQAHandle, originalLoudnessDensity,
    distortedLoudnessDensity, UseThisFrame, &originalLoudnessHulp, &distortedLoudnessHulp,
1839                                           &loudnessScaleLow, &oldLoudnessScaleLow,
    &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh, 1.24,
    7.5, 0.6, 0.6, 0.8, noiseIndicatorScalingImpact)
1840
1841        //Local loudness scaling distorted, predominantly for modelling the local impact of additive
    noise and pulses during silent intervals
1842        oldOldLoudnessScaleLow = 1.0
1843        oldLoudnessScaleLow = 1.0
1844        if (maxFreqBarkSource < maxFreqBarkHighLimit)
1845            if (maxFreqBarkSource < maxFreqBarkLowLimit)
1846                LoudnessScalingDistortedAdditiveNoise1(statics, POLQAHandle,
    originalLoudnessDensity, distortedLoudnessDensity, UseThisFrame,
1847                    &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
    &oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
1848                    4.0, 12.0, 1.0, 2.7, 0.0, 0.0, 1.0, 0, 0.6, 0.55, 0.6)
1849            else LoudnessScalingDistortedAdditiveNoise1(statics, POLQAHandle,
    originalLoudnessDensity, distortedLoudnessDensity, UseThisFrame,
1850                    &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
    &oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
1851                    3.0, 16.0, 1.0, 2.4, 0.0, 0.0, 1.0, 0, 0.6, 0.55, 0.6)
1852        else LoudnessScalingDistortedAdditiveNoise1(statics, POLQAHandle, originalLoudnessDensity,
    distortedLoudnessDensity, UseThisFrame,
1853                &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
    &oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
1854                3.0, 17.0, 1.2, 1.9, 0.00, 0.00, 1.0, 0, 0.6, 0.55, 0.6)
1855
1856        //Frequency response compensation in loudness domain
```

```
1857        FreqResponseCompensLoudnessDomain(statics, POLQAHandle, originalPitchLoudnessDensityMainAvg,
    distortedPitchLoudnessDensityMainAvg,
1858            originalLoudnessDensity, distortedLoudnessDensity, aActiveFreqresponse, 2.0, 2.0, 10.0,
    Para.FreqRespCompPow)
1859
1860        //set highest bands to zero
1861
1862        XFLOAT HBconst1, HBconst2
1863        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
1864        {
1865            if (maxFreqBarkSource < maxFreqBarkHighLimit)
1866            {
1867                if (maxFreqBarkSource < maxFreqBarkLowLimit)
1868                {
1869                    HBconst1 = 16.5
1870                    HBconst2 = 17.5
1871                }
1872                else
1873                {
1874                    HBconst1 = 21.0
1875                    HBconst2 = 21.0
1876                }
1877            }
1878            else
1879            {
1880                HBconst1 = 21.0
1881                HBconst2 = 21.0
1882            }
1883            originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, HBconst1, 99.0)
1884            distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, HBconst2, 99.0)
1885        }
1886
1887        LoudnessScalingTowards20Sone(statics, POLQAHandle, aDistortedLoudness,
    originalLoudnessDensity, distortedLoudnessDensity, &aLoudnessScalingDistorted,
1888            &aDistortedLoudnessMean, numberOfSpeechFrames, bandIdxLow, bandIdxHigh, 17.0, 0.0,
    LpBandRangePartial, Para.PartGlobLoudScaleP1, fixedGlobalInternalLevel,
1889                Para.PartGlobLoudScaleP1, 0.7, 0.4)
1890
1891        // noise contrast calculation in the super silent frames
1892        XFLOAT noiseContrastMax1 = 1.0
1893        XFLOAT noiseContrastMax2 = 1.0
1894        {
1895            noiseContrastParameter = 1.0
1896            XFLOAT noiseContrastMax3 = 1.0
1897            int frameIndexMax1 = 0
1898            int frameIndexMax2 = 0
1899            int frameIndexMax3 = 0
1900            for (frameIndex = statics->startFrameIdx  (frameIndex <= statics->stopFrameIdx - 7)
    frameIndex++)
1901            {
1902                if (aSuperSilent.m_pData[frameIndex] && aSuperSilent.m_pData[frameIndex + 1] &&
    aSuperSilent.m_pData[frameIndex + 2] && aSuperSilent.m_pData[frameIndex + 3] &&
    aSuperSilent.m_pData[frameIndex + 4] && aSuperSilent.m_pData[frameIndex + 5] &&
    aSuperSilent.m_pData[frameIndex + 6] && aSuperSilent.m_pData[frameIndex + 7])
1903                {
1904                    hulp1 = aDistortedLoudness.m_pData[frameIndex] + 0.2
1905                    hulp2 = aDistortedLoudness.m_pData[frameIndex + 3] + 0.2
1906                    if (hulp1 > 1.4) hulp1 = 1.4
1907                    if (hulp2 > 1.4) hulp2 = 1.4
1908                    hulpRatio = (hulp2) / (hulp1)
1909                    if (hulpRatio > 7.0) hulpRatio = 7.0
1910                    hulp1 = pow(hulpRatio, Para.NoiseContrSupSilP1)
1911
1912                    if ((hulp1 > noiseContrastMax3))
1913                    {
1914                        if ((hulp1 > noiseContrastMax2))
1915                        {
1916                            if ((hulp1 > noiseContrastMax1))
```

```
                            {
                                noiseContrastMax3 = noiseContrastMax2
                                noiseContrastMax2 = noiseContrastMax1
                                noiseContrastMax1 = hulp1
                                frameIndexMax1 = frameIndex
                            }
                            else
                            {
                                noiseContrastMax3 = noiseContrastMax2
                                noiseContrastMax2 = hulp1
                                frameIndexMax2 = frameIndex
                            }
                        }
                        else
                        {
                            noiseContrastMax3 = hulp1
                            frameIndexMax3 = frameIndex
                        }

                    }
                    if (hulp1 > 1.0) noiseContrastParameter *= pow(hulp1, 0.1)
                }
            }
            noiseContrastParameter *= pow(noiseContrastParameter, 0.1)
            noiseContrastParameter *= pow(noiseContrastParameter, 0.1)
        }

        LoudnessScalingOriginalTowardsDistorted1(statics, POLQAHandle,
aOriginalLoudnessCompletelyScaled,
            aDistortedLoudnessCompletelyScaled, aOriginalLoudness, aDistortedLoudness,
originalLoudnessDensity,
            distortedLoudnessDensity, &aLoudnessScalingOriginal, numberOfSpeechFrames,
&LpLoudnessMeanComplete, &LpBandRangeComplete,
            &hulp1, &hulp2, bandIdxLow, bandIdxHigh, &aOriginalLoudnessMean,
&aDistortedLoudnessMean, 0.0, 0.0)

        //What you should do here:
        //Normalize aOriginalLoudnessMean and aDistortedLoudnessMean to the number of speech frames
plus 0.5

        XFLOAT const1
        XFLOAT const2
        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
        {
            if (maxFreqBarkSource < maxFreqBarkHighLimit)
            {
                const1 = 4.1
                const2 = 0.97
            }
            else
            {
                const1 = 3.9
                const2 = 0.96
            }

            aLoudnessScalingOriginal = (pow(aDistortedLoudnessMean, (1.0 / LpLoudnessMeanComplete))
+ const1) / (pow(aOriginalLoudnessMean, (1.0 / LpLoudnessMeanComplete)) + const1)
            originalLoudnessDensity.MultiplyWith(frameIndex, pow(aLoudnessScalingOriginal, const2))
        }

        //Noise suppression in loudness densities using super silent frames for modelling the global
impact of additive noise during silent intervals

        XFLOAT SSconst1, SSconst2, SSconst3
        if (maxFreqBarkSource < maxFreqBarkHighLimit)
        {
            if (maxFreqBarkSource<maxFreqBarkLowLimit)
            {
```

```
1978
1979                originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity,
    aSuperSilent, 1.0, numberOfSuperSilentFrames)
1980                distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
    aSuperSilent, 1.0, numberOfSuperSilentFrames)
1981
1982                SSconst1 = 1.4
1983                SSconst2 = 0.3
1984                SSconst3 = 0.22
1985            }
1986            else
1987            {
1988
1989                originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity,
    aSuperSilent, 1.0, numberOfSuperSilentFrames)
1990                distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
    aSuperSilent, 1.0, numberOfSuperSilentFrames)
1991
1992                SSconst1 = 1.2
1993                SSconst2 = 0.5
1994                SSconst3 = 0.24
1995            }
1996        }
1997        else
1998        {
1999
2000            originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity,
    aSuperSilent, 2.0, numberOfSuperSilentFrames)
2001            distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
    aSuperSilent, 2.0, numberOfSuperSilentFrames)
2002
2003            SSconst1 = 1.1
2004            SSconst2 = 0.8
2005            SSconst3 = 0.23
2006        }
2007
2008        originalLoudnessDensity.AudibleNoiseRespCompensationOfPartly(POLQAHandle,
    originalPitchLoudnessDensityMainAvg, SSconst1)
2009
2010        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2011            matbCopy(distortedLoudnessDensity.m_pData[fr],
    hulp1DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.aNumberOfBands)
2012            matbCopy(distortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.aNumberOfBands)
2013        }
2014        hulp1DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly(POLQAHandle,
    distortedPitchLoudnessDensityMainAvg, SSconst1)
2015        hulp2DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly2(distortedPitchLoudnessDe
    nsityMainAvg, SSconst2, SSconst3)
2016        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2017            matbMpy1(1 - alpha_noise, hulp1DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2018            matbMpy1(alpha_noise, hulp2DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2019        }
2020        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++)
2021            matbAdd3(hulp1DistortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2022
2023        InternalCorrectedLoudness(statics, POLQAHandle, aOriginalLoudness, aDistortedLoudness,
2024            originalLoudnessDensity, distortedLoudnessDensity, &LpLoudness, bandIdxLow, bandIdxHigh)
2025
2026        //Compute disturbance density for POLQA
2027        disturbanceDensity.DifferenceOf(distortedLoudnessDensity, originalLoudnessDensity)
2028
2029        mask.MinimumOf(distortedLoudnessDensity, originalLoudnessDensity)
2030        if (maxFreqBarkSource < maxFreqBarkHighLimit){
```

```
2031            mask *= (XFLOAT) 0.25
2032            disturbanceDensity.MaskWith(POLQAHandle, mask)
2033        }
2034        else {
2035            mask *= (XFLOAT) 0.26
2036            disturbanceDensity.MaskWith48k(POLQAHandle, mask)
2037        }
2038
2039        CIntArray frameWasSkipped
2040        frameWasSkipped.SetSize(statics->nrFrames)
2041        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2042        {
2043            frameWasSkipped.m_pData[frameIndex] = FALSE
2044        }
2045
2046        for (int utt = 1  utt < aStartSampleUtterance.GetSize()  utt++)
2047        {
2048            int startFrame = (int)floor((aStartSampleUtterance.m_pData[utt] +
    aDelayUtterance.m_pData[utt]) / (OverlapCoeff * aTransformLength))
2049            if (startFrame >(int) floor((aStopSampleUtterance.m_pData[utt - 1] +
    aDelayUtterance.m_pData[utt - 1]) / (OverlapCoeff * aTransformLength)))
2050            {
2051                startFrame = (int)floor((aStopSampleUtterance.m_pData[utt - 1] +
    aDelayUtterance.m_pData[utt - 1]) / (OverlapCoeff * aTransformLength))
2052            }
2053
2054            if (startFrame < 0)
2055            {
2056                startFrame = 0
2057            }
2058
2059            int delayJumpInSamples = aDelayUtterance.m_pData[utt] - aDelayUtterance.m_pData[utt - 1]
2060
2061            if (delayJumpInSamples < -(int)(aTransformLength * OverlapCoeff))
2062            {
2063                int stopFrame = (int)((aStartSampleUtterance.m_pData[utt] + (((0) >
    (fabs((XFLOAT)delayJumpInSamples))) ? (0) : (fabs((XFLOAT)delayJumpInSamples)))) /
    (OverlapCoeff * aTransformLength)) + 1
2064            }
2065        }
2066
2067        frameWasSkipped.SetSize(statics->nrFrames)
2068        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2069        {
2070            frameWasSkipped.m_pData[frameIndex] = FALSE
2071        }
2072
2073        for (int utt = 1  utt < aStartSampleUtterance.GetSize()  utt++)
2074        {
2075            int startFrame = (int)floor((aStartSampleUtterance.m_pData[utt] +
    aDelayUtterance.m_pData[utt]) / (OverlapCoeff * aTransformLength))
2076            if (startFrame >(int) floor((aStopSampleUtterance.m_pData[utt - 1] +
    aDelayUtterance.m_pData[utt - 1]) / (OverlapCoeff * aTransformLength)))
2077            {
2078                startFrame = (int)floor((aStopSampleUtterance.m_pData[utt - 1] +
    aDelayUtterance.m_pData[utt - 1]) / (OverlapCoeff * aTransformLength))
2079            }
2080
2081            if (startFrame < 0)
2082            {
2083                startFrame = 0
2084            }
2085
2086            int delayJumpInSamples = aDelayUtterance.m_pData[utt] - aDelayUtterance.m_pData[utt - 1]
2087
2088            if (delayJumpInSamples < -(int)(aTransformLength * OverlapCoeff))
2089            {
2090                int stopFrame = (int)((aStartSampleUtterance.m_pData[utt] + (((0) >
```

```
    (fabs((XFLOAT)delayJumpInSamples))) ? (0) : (fabs((XFLOAT)delayJumpInSamples)))) /
    (OverlapCoeff * aTransformLength)) + 1
            }
        }

        disturbanceDensity.ComputeLpWeights(POLQAHandle, MINIMUM_POWER_FREQ, STEP_POWER_FREQ,
    NUMBER_OF_POWERS_OVER_FREQ, aDisturbance)

        {
            const int length = statics->stopFrameIdx - statics->startFrameIdx + 1
            SmartBufferPolqa tempBuffer(POLQAHandle, length)
            XFLOAT *temp = tempBuffer.Buffer

            matbPow2(aDisturbance[2].m_pData + statics->startFrameIdx, 3.0, temp, length)
            overallAvgDisturbance = matSum(temp, length)

            matbPow2(aAddedDisturbance[2].m_pData + statics->startFrameIdx, 3.0, temp, length)
            overallAvgAddedDisturbance = matSum(temp, length)
        }
        overallAvgDisturbance /= (numberOfSpeechFrames + 0.01)
        overallAvgAddedDisturbance /= (numberOfSpeechFrames + 0.01)
        overallAvgDisturbance = pow(overallAvgDisturbance, 0.33)
        overallAvgAddedDisturbance = pow(overallAvgAddedDisturbance, 0.33)

        overallDisturbance = 0.0
        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
            if (aSilent.m_pData[frameIndex])
                overallDisturbance += aDisturbance[2].m_pData[frameIndex]
        overallDisturbance /= (numberOfSpeechFrames + 0.01)
        //END POLQAMAIN PART 1s
        CheckTimeMatEval(POLQAHandle->mh, 3, &ClockCycles, &TimeDiff)
        AddProcessingTime(pOverviewHolder, "PESQMAIN PART 2", TimeDiff, ClockCycles)

        frameFlatnessDisturbanceAvg = 0.0
        frameFlatnessDisturbanceAvgCompensationSilent = 0.0
        frameFlatnessDisturbanceAvgCompensationActive = 0.0

        numberOfActiveFrames = 0
        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
        {
            if (aSilent.m_pData[frameIndex])
                frameFlatnessDisturbanceAvgCompensationSilent +=
    disturbanceDensity.SpectralFlatnessIandM(frameIndex)
            else
            {
                numberOfActiveFrames++
                frameFlatnessDisturbanceAvgCompensationActive +=
    disturbanceDensity.SpectralFlatnessIandM(frameIndex)
            }
        }
        XFLOAT FBConst
        if (maxFreqBarkSource < maxFreqBarkHighLimit)
            FBConst = 0.19
        else
            FBConst = 0.15

        frameFlatnessDisturbanceAvgCompensationSilent /= (numberOfSilentFrames + 0.1)
        frameFlatnessDisturbanceAvgCompensation000silent =
    frameFlatnessDisturbanceAvgCompensationSilent
        frameFlatnessDisturbanceAvgCompensationAddedSilent =
    pow((frameFlatnessDisturbanceAvgCompensationSilent + 0.8), 0.12)
        frameFlatnessDisturbanceAvgCompensationSilent =
    pow((frameFlatnessDisturbanceAvgCompensationSilent + 0.9), FBConst)
        if (frameFlatnessDisturbanceAvgCompensation000silent > 0.55)
    frameFlatnessDisturbanceAvgCompensation000silent = 0.55
        if (frameFlatnessDisturbanceAvgCompensation000silent < 0.15)
    frameFlatnessDisturbanceAvgCompensation000silent = 0.15
        frameFlatnessDisturbanceAvgCompensation000silent /= 0.55
```

```
2149        frameFlatnessDisturbanceAvgCompensation000silent =
    pow((frameFlatnessDisturbanceAvgCompensation000silent), 0.2)
2150
2151        frameFlatnessDisturbanceAvgCompensationActive /= (numberOfActiveFrames + 0.1)
2152        frameFlatnessDisturbanceAvgCompensationActiveFrq =
    frameFlatnessDisturbanceAvgCompensationActive
2153        frameFlatnessDisturbanceAvgCompensationActiveFrq = 0.3*(1.0 +
    frameFlatnessDisturbanceAvgCompensationActiveFrq)
2154        if (frameFlatnessDisturbanceAvgCompensationActiveFrq > 0.5)
    frameFlatnessDisturbanceAvgCompensationActiveFrq = 0.5
2155        frameFlatnessDisturbanceAvgCompensation000active =
    frameFlatnessDisturbanceAvgCompensationActive
2156        frameFlatnessDisturbanceAvgCompensationAddedActive =
    pow((frameFlatnessDisturbanceAvgCompensationActive + 0.8), 0.45)
2157        frameFlatnessDisturbanceAvgCompensationActive =
    pow((frameFlatnessDisturbanceAvgCompensationActive + 1.2), 0.02)
2158        if (frameFlatnessDisturbanceAvgCompensation000active > 0.5)
    frameFlatnessDisturbanceAvgCompensation000active = 0.5
2159        if (frameFlatnessDisturbanceAvgCompensation000active < 0.2)
    frameFlatnessDisturbanceAvgCompensation000active = 0.2
2160        frameFlatnessDisturbanceAvgCompensation000active /= 0.5
2161        frameFlatnessDisturbanceAvgCompensation000active =
    1.02*pow(frameFlatnessDisturbanceAvgCompensation000active, 0.2)
2162
2163
2164
2165        //POLQAMAIN PART 0 ADDED
2166        CheckTimeMatInit(POLQAHandle->mh, 3)
2167
2168        oldOldScale = 1.0
2169        oldScale = 1.0
2170        minimumOriginalFramePower = 10000000.0
2171        MaxScale = 1.0
2172        MinScale = 1.0
2173        MinMinScale = 0.5
2174        for (frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
2175        {
2176            if (UseThisFrame[frameIndex])
2177            {
2178                scale = (aDistortedTotalPower.m_pData[frameIndex] + (XFLOAT) 1.0e5) /
    (aOriginalTotalPower.m_pData[frameIndex] + (XFLOAT) 1.0e5)
2179
2180                if (scale > MaxScale) scale = MaxScale
2181                if (scale < MinMinScale) scale = MinMinScale
2182                aScale.m_pData[frameIndex] = scale
2183                scale = (XFLOAT) 0.15 * oldOldScale + (XFLOAT) 0.35 * oldScale + (XFLOAT) 0.5 *
    scale
2184                oldOldScale = oldScale
2185                oldScale = scale
2186                originalPitchPowerDensity.MultiplyWith(frameIndex, pow(scale, 0.6))
2187            }
2188            else
2189            {
2190                originalPitchPowerDensity.MultiplyWith(frameIndex, 0.0)
2191                distortedPitchPowerDensity.MultiplyWith(frameIndex, 0.0)
2192            }
2193        }
2194
2195        FreqResponseCompensPowerDomain(statics, POLQAHandle, originalPitchPowerDensityMainAvg,
    distortedPitchPowerDensityMainAvg, originalPitchPowerDensity,
2196            distortedPitchPowerDensity, aActiveFreqresponseIntell, 0.4, 0.4, 0.8, 3.0E3, 1)
2197
2198        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2199        {
2200        }
2201
2202        smearedOriginalPitchPowerDensity.ExcitationOf(POLQAHandle, originalPitchPowerDensity,
```

```
    UseThisFrame, statics->listeningCondition)
2203        smearedDistortedPitchPowerDensity.ExcitationOf(POLQAHandle, distortedPitchPowerDensity,
    UseThisFrame, statics->listeningCondition)
2204
2205        originalLoudnessDensityNoInhibition.IntensityWarpingOfSpeechLQ(POLQAHandle,
    smearedOriginalPitchPowerDensity, maxFreqBarkSource)
2206        distortedLoudnessDensityNoInhibition.IntensityWarpingOfSpeechLQ(POLQAHandle,
    smearedDistortedPitchPowerDensity, maxFreqBarkSource)
2207
2208        originalLoudnessDensity.InhibitionSpeechLQ(POLQAHandle, originalLoudnessDensityNoInhibition,
    maxFreqBarkSource)
2209        distortedLoudnessDensity.InhibitionSpeechLQ(POLQAHandle,
    distortedLoudnessDensityNoInhibition, maxFreqBarkSource)
2210
2211        //END POLQAMAIN PART 0 ADDED
2212
2213
2214        //POLQAMAIN PART 1 ADDED
2215
2216        FreqResponseCompensLoudnessDomain(statics, POLQAHandle, originalPitchLoudnessDensityMainAvg,
    distortedPitchLoudnessDensityMainAvg,
2217            originalLoudnessDensity, distortedLoudnessDensity, aActiveFreqresponse, 0.3, 0.3, 0.15,
    0.8)
2218
2219        originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity, aSilent,
    4.0, numberOfSilentFrames)
2220        distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
    aSilent, 2.0, numberOfSilentFrames)
2221
2222        originalLoudnessDensity. AudibleNoiseRespCompensationOfPartlyAdded (POLQAHandle,
    originalPitchLoudnessDensityMainAvg, 0.25)
2223
2224        hulp1DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartlyAdded(POLQAHandle,
    hulp1DistortedPitchLoudnessDensityMainAvg, 0.25)
2225        hulp2DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly2Added(hulp2DistortedPitch
    LoudnessDensityMainAvg,
    0.6*delayVariationCompensationAdded*aDistortedSilencePowerMeanCompensation /
    pow(noiseContrastMax1, 0.65), 1.0)
2226        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2227            matbMpy1(1. - alpha_noise, hulp1DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2228            matbMpy1(alpha_noise, hulp2DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2229            matbMpy1(1. - alpha_noise, hulp1DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2230            matbMpy1(alpha_noise, hulp2DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2231        }
2232        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2233            matbAdd3(hulp1DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    hulp2DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2234            matbAdd3(hulp1DistortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2235        }
2236
2237        oldOldLoudnessScaleLow = 1.0
2238        oldLoudnessScaleLow = 1.0
2239        if (maxFreqBarkSource <maxFreqBarkHighLimit)
2240            if (maxFreqBarkSource < maxFreqBarkLowLimit)
2241                LoudnessScalingOriginalClipEffect2(statics, POLQAHandle, originalLoudnessDensity,
    distortedLoudnessDensity, UseThisFrame,
2242                    &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
    &oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2243                    1.0, 17.0, 1.6, 5.0, 0.9)
2244            else LoudnessScalingOriginalClipEffect2(statics, POLQAHandle, originalLoudnessDensity,
```

```
distortedLoudnessDensity, UseThisFrame,
2245                &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
&oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2246                1.0, 17.0, 1.6, 5.0, 0.8)
2247        else LoudnessScalingOriginalClipEffect2(statics, POLQAHandle, originalLoudnessDensity,
distortedLoudnessDensity, UseThisFrame,
2248            &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow, &oldLoudnessScaleLow,
&oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2249            1.0, 99.0, 1.6, 5.0, 0.8)
2250        oldOldLoudnessScaleLow = 1.0
2251        oldLoudnessScaleLow = 1.0
2252        delayReliabilityPerFrameWeightOld = 1.0
2253        if (maxFreqBarkSource <maxFreqBarkHighLimit)
2254            if (maxFreqBarkSource < maxFreqBarkLowLimit)
2255                LoudnessScalingDistortedAdditiveNoise2(statics, POLQAHandle,
originalLoudnessDensity, distortedLoudnessDensity, UseThisFrame,
2256                &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
&oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2257                1.0, 17.0, 0.8, 7.0, 0.4, 0.35, noiseContrastMax1, 0.65)
2258            else LoudnessScalingDistortedAdditiveNoise2(statics, POLQAHandle,
originalLoudnessDensity, distortedLoudnessDensity, UseThisFrame,
2259                    &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
&oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2260                    1.0, 17.0, 0.8, 7.5, 0.4, 0.30, noiseContrastMax1, 0.6)
2261        else LoudnessScalingDistortedAdditiveNoise2(statics, POLQAHandle, originalLoudnessDensity,
distortedLoudnessDensity, UseThisFrame,
2262                &originalLoudnessHulp, &distortedLoudnessHulp, &loudnessScaleLow,
&oldLoudnessScaleLow, &oldOldLoudnessScaleLow, bandIdxLow, bandIdxHigh,
2263                1.0, 17.0, 0.85, 7.5, 0.3, 0.20, noiseContrastMax1, 0.6)
2264
2265        FreqResponseCompensLoudnessDomain(statics, POLQAHandle, originalPitchLoudnessDensityMainAvg,
distortedPitchLoudnessDensityMainAvg,
2266            originalLoudnessDensity, distortedLoudnessDensity, aActiveFreqresponse, 0.9, 0.9, 0.01,
1.0)
2267
2268        if (maxFreqBarkSource < maxFreqBarkLowLimit) {
2269            for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
frameIndex++) {
2270                originalLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 17.0, 99.0)
2271                distortedLoudnessDensity.MultiplyWithOverBandRange(frameIndex, 0.0, 18.0, 99.0)
2272            }
2273        } else {
2274        }
2275
2276        LoudnessScalingTowards20Sone(statics, POLQAHandle, aDistortedLoudness,
originalLoudnessDensity, distortedLoudnessDensity, &aLoudnessScalingDistorted,
2277            &aDistortedLoudnessMean, numberOfSpeechFrames, bandIdxLow, bandIdxHigh, 16.0, 2.0, 2.1,
LpLoudnessMeanPartial, fixedGlobalInternalLevelAdded,
2278            LpLoudnessMeanPartial, 0.9, 0.18)
2279
2280        if (maxFreqBarkSource < maxFreqBarkLowLimit)
2281            LoudnessScalingOriginalTowardsDistorted2(statics, POLQAHandle,
aOriginalLoudnessCompletelyScaled,
2282                aDistortedLoudnessCompletelyScaled, aOriginalLoudness, aDistortedLoudness,
originalLoudnessDensity,
2283                distortedLoudnessDensity, &aLoudnessScalingOriginal, numberOfSpeechFrames,
&LpLoudnessMeanComplete, &LpBandRangeComplete,
2284                &hulp1, &hulp2, bandIdxLow, bandIdxHigh, &aOriginalLoudnessMean,
&aDistortedLoudnessMean, 0.1*pow(globalScaleDistortedToFixedlevel, 0.1),
pow(aLoudnessScalingOriginal, 0.3))
2285
2286        originalPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(originalLoudnessDensity, aSilent,
2.0, numberOfSilentFrames)
2287        distortedPitchLoudnessDensityMainAvg.TimeLpAudibleOfSilent(distortedLoudnessDensity,
aSilent, 10.0, numberOfSilentFrames)
2288
2289        originalLoudnessDensity.AudibleNoiseRespCompensationOfPartlyAdded(POLQAHandle,
originalPitchLoudnessDensityMainAvg, 0.07)
```

```
2290
2291        hulp1DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartlyAdded(POLQAHandle,
    hulp1DistortedPitchLoudnessDensityMainAvg, 0.07)
2292        hulp2DistortedLoudnessDensity.AudibleNoiseRespCompensationOfPartly2Added(hulp2DistortedPitch
    LoudnessDensityMainAvg, 0.15*delayVariationCompensationAdded, 2.2*noiseContrastMax1)
2293        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2294            matbMpy1(1 - alpha_noise, hulp1DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2295            matbMpy1(alpha_noise, hulp2DistortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2296            matbMpy1(1. - alpha_noise, hulp1DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2297            matbMpy1(alpha_noise, hulp2DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2298        }
2299        for (int fr = statics->startFrameIdx  fr <= statics->stopFrameIdx  fr++){
2300            matbAdd3(hulp1DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    hulp2DistortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedPitchLoudnessDensityMainAvg.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2301            matbAdd3(hulp1DistortedLoudnessDensity.m_pData[fr],
    hulp2DistortedLoudnessDensity.m_pData[fr], distortedLoudnessDensity.m_pData[fr],
    distortedLoudnessDensity.aNumberOfBands)
2302        }
2303
2304        InternalCorrectedLoudness(statics, POLQAHandle, aOriginalLoudness, aDistortedLoudness,
2305            originalLoudnessDensity, distortedLoudnessDensity, &LpLoudness, bandIdxLow, bandIdxHigh)
2306
2307        disturbanceDensityAsymAdd.DifferenceOf(distortedLoudnessDensity, originalLoudnessDensity)
2308        mask.MinimumOf(distortedLoudnessDensity, originalLoudnessDensity)
2309        if (maxFreqBarkSource < maxFreqBarkHighLimit) {
2310            mask *= (XFLOAT) 0.85
2311            disturbanceDensityAsymAdd.MaskWith(POLQAHandle, mask)
2312        } else {
2313            mask *= (XFLOAT) 0.85
2314            disturbanceDensityAsymAdd.MaskWith48k(POLQAHandle, mask)
2315        }
2316        disturbanceDensityAsymAdd.MultiplyWithAsymmetryFactorAddOf(originalPitchPowerDensity,
    distortedPitchPowerDensity, statics->listeningCondition, noiseContrastMax1, aSuperSilent,
    aDistortedSilencePowerMeanCompensation, maxFreqBarkSource)
2317
2318        disturbanceDensityAsymAdd.ComputeLpWeights(POLQAHandle, MINIMUM_POWER_FREQ, STEP_POWER_FREQ,
    NUMBER_OF_POWERS_OVER_FREQ, aAddedDisturbance)
2319
2320        //END POLQAMAIN PART 1 ADDED
2321
2322        count = 0
2323        XFLOAT  sumXY = 0.0
2324        XFLOAT  sumX = 0.0
2325        XFLOAT  sumY = 0.0
2326        XFLOAT  sumX2 = 0.0
2327        XFLOAT  sumY2 = 0.0
2328        XFLOAT  correlationOriginalWithDisturbance
2329        for (frameIndex = (statics->startFrameIdx)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
2330        {
2331            if (aActiveFreqresponse.m_pData[frameIndex])
2332            {
2333                XFLOAT X = aOriginalLoudness.m_pData[frameIndex]
2334                XFLOAT Y = aDisturbance[2].m_pData[frameIndex]
2335                sumXY += X*Y
2336                sumX += X
2337                sumY += Y
2338                sumX2 += X*X
2339                sumY2 += Y*Y
2340                count++
2341            }
2342        }
```

```
2343        int originalLen = statics->stopFrameIdx
2344
2345        if (count > 2 && sumX > 0.0 && sumY > 0.0)
2346        {
2347            correlationOriginalWithDisturbance = (count*sumXY - sumX*sumY) / sqrt((count*sumX2 -
    sumX*sumX)*(count*sumY2 - sumY*sumY))
2348        }
2349        else
2350        {
2351            correlationOriginalWithDisturbance = 0.0
2352        }
2353        correlationOriginalWithDisturbanceCompensation000ForReverb =
    correlationOriginalWithDisturbance
2354        if (correlationOriginalWithDisturbanceCompensation000ForReverb < 0.5)
    correlationOriginalWithDisturbanceCompensation000ForReverb = 0.5
2355        correlationOriginalWithDisturbanceCompensation000ForMNRU =
    correlationOriginalWithDisturbance
2356        if (correlationOriginalWithDisturbanceCompensation000ForMNRU<0.0)
    correlationOriginalWithDisturbanceCompensation000ForMNRU = 0.0
2357        if (correlationOriginalWithDisturbanceCompensation000ForMNRU>0.7)
    correlationOriginalWithDisturbanceCompensation000ForMNRU = 0.7
2358        MNRU_indicator = correlationOriginalWithDisturbanceCompensation000ForMNRU
2359        correlationOriginalWithDisturbanceCompensation000ForMNRU /= 15.0
2360
2361        if (correlationOriginalWithDisturbance < 0.3) correlationOriginalWithDisturbance = 0.3
2362
2363        CheckTimeMatEval(POLQAHandle->mh, 3, &ClockCycles, &TimeDiff)
2364
2365
2366
2367        //POLQAMAIN PART 2
2368        CheckTimeMatInit(POLQAHandle->mh, 3)
2369
2370        //Start with calculating the normalized LoudnessDensities for use in timbre and frame
    correlations
2371        LoudnessScalingTowards20Sone(statics, POLQAHandle, aDistortedLoudness,
    originalLoudnessDensity, distortedLoudnessDensity, &aLoudnessScalingDistorted,
2372            &aDistortedLoudnessMean, numberOfSpeechFrames, bandIdxLow, bandIdxHigh, 99.0, 0.0,
    LpBandRangePartial, LpLoudnessMeanPartial, fixedGlobalInternalLevel,
2373            LpLoudnessMeanPartial, 0.9, 1)
2374
2375            LoudnessScalingOriginalTowardsDistorted3(statics, POLQAHandle,
    aOriginalLoudnessCompletelyScaled,
2376                aDistortedLoudnessCompletelyScaled, aOriginalLoudness, aDistortedLoudness,
    originalLoudnessDensity,
2377                distortedLoudnessDensity, &aLoudnessScalingOriginal, numberOfSpeechFrames,
    &LpLoudnessMeanComplete, &LpBandRangeComplete,
2378                &hulp1, &hulp2, bandIdxLow, bandIdxHigh, &aOriginalLoudnessMean,
    &aDistortedLoudnessMean, 0.01, aLoudnessScalingOriginal)
2379
2380        CreateAlignTimeSeries(POLQAHandle, aOriginalTimeSeries, aStartSampleUtterance,
    aStopSampleUtterance, aDelayUtterance, GetTransformLength(), aAlignedOriginalTimeSeries)
2381
2382        if (mpAlignedPairFile)
2383            fclose(mpAlignedPairFile)
2384
2385        XFLOAT quantileDisturbanceOverFile = 1.0
2386        XFLOAT hulpLevel = ((XFLOAT) log10 (10.0 + aAvgDistortedPower/1.0e8))/1.02
2387        pitchFreqReference = mPitchFreqRef
2388        pitchFreqDegraded = mPitchFreqDeg
2389
2390        //compensation and correction factors including maximum used in final disturbance
    calculation
2391        averageScale = 0.0
2392        maxDisturbanceFrame = 0
2393        maxDisturbanceOverFile = 0.0
2394        frameCorrelationTimeOriginalOld = 0.0
2395        frameCorrelationTimeDistortedOld = 0.0
```

```
2396        overallAvgDisturbance = 0.0
2397        overallMovingAvgDisturbance = 0.0
2398        overallMovingAvgDisturbanceOld = 0.0
2399        overallMovingAvgDisturbanceOldOld = 0.0
2400
2401        SNRloudnessCountTotal = 0
2402        SNRloudnessCountExcellent = 0
2403        SNRloudnessCountGood = 0
2404        SNRloudnessCountFair = 0
2405        SNRloudnessCountPoor = 0
2406        SNRloudnessCountBad = 0
2407        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2408        {
2409            if (aOriginalLoudnessCompletelyScaled.m_pData[frameIndex] > 1.0)
2410            {
2411                SNRloudnessCountTotal++
2412                if (aOriginalLoudnessCompletelyScaled.m_pData[frameIndex] >
    aDistortedLoudnessCompletelyScaled.m_pData[frameIndex])
2413                {
2414                    hulp = (aOriginalLoudnessCompletelyScaled.m_pData[frameIndex] -
    aDistortedLoudnessCompletelyScaled.m_pData[frameIndex]) /
    aOriginalLoudnessCompletelyScaled.m_pData[frameIndex]
2415                    if (hulp <= 0.7) SNRloudnessCountExcellent++
2416                    if ((hulp > 1.0) && (hulp <= 2.0)) SNRloudnessCountGood++
2417                    if ((hulp > 2.0) && (hulp <= 3.0)) SNRloudnessCountFair++
2418                    if ((hulp > 3.0) && (hulp <= 4.0)) SNRloudnessCountPoor++
2419                    if (hulp > 4.0) SNRloudnessCountBad++
2420                }
2421                else
2422                {
2423                    hulp = (aDistortedLoudnessCompletelyScaled.m_pData[frameIndex] -
    aOriginalLoudnessCompletelyScaled.m_pData[frameIndex]) /
    aOriginalLoudnessCompletelyScaled.m_pData[frameIndex]
2424                    if (hulp<1.5) SNRloudnessCountExcellent++
2425                    if ((hulp>1.0) && (hulp <= 2.0)) SNRloudnessCountGood++
2426                    if ((hulp > 2.0) && (hulp <= 3.0)) SNRloudnessCountFair++
2427                    if ((hulp > 3.0) && (hulp <= 4.0)) SNRloudnessCountPoor++
2428                    if (hulp > 4.0) SNRloudnessCountBad++
2429                }
2430            }
2431        }
2432        SNRloudnessWeightedRatio1 = (SNRloudnessCountExcellent + 0.1) / (SNRloudnessCountTotal +
    0.1)
2433        SNRloudnessWeightedRatio2 = pow(SNRloudnessWeightedRatio1, 0.1)
2434        SNRloudnessWeightedRatio3 = pow(SNRloudnessWeightedRatio1, 0.2)
2435        SNRloudnessWeightedRatio4 = pow(SNRloudnessWeightedRatio1, 0.33)
2436        SNRloudnessWeightedRatio5 = pow(SNRloudnessWeightedRatio1, 0.4)
2437        SNRloudnessWeightedRatio1 = pow(SNRloudnessWeightedRatio1, 0.03)
2438
2439        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2440        {
2441            overallAvgDisturbance += aDisturbance[1].m_pData[frameIndex]
2442            if (aDisturbance[1].m_pData[frameIndex] > maxDisturbanceOverFile)
2443            {
2444                maxDisturbanceOverFile = aDisturbance[1].m_pData[frameIndex]
2445                maxDisturbanceFrame = frameIndex
2446            }
2447        }
2448        overallAvgDisturbance /= (numberOfSpeechFrames + 0.01)
2449        hulp1 = overallAvgDisturbance - 1.5
2450        if (hulp1 < 3.0) hulp1 = 3.0
2451        maxDisturbance = 130.0 + maxDisturbanceOverFile / hulp1
2452
2453        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2454        {
2455            if (aDisturbance[5].m_pData[frameIndex] > maxDisturbanceOverFile / 6.0)
2456            {
2457                quantileDisturbanceOverFile += 1.0
```

```
2458            }
2459        }
2460        quantileDisturbanceOverFile /= (numberOfSpeechFrames + 1.0)
2461        quantileDisturbanceOverFile = pow(quantileDisturbanceOverFile, 0.011)
2462
2463        XFLOAT varianceDisturbanceDown = 0.0
2464        XFLOAT varianceDisturbanceUp = 0.0
2465        for (frameIndex = (statics->startFrameIdx + 1)  frameIndex <= statics->stopFrameIdx
    frameIndex++)
2466        {
2467            varianceDisturbanceDown += (aDisturbance[2].m_pData[frameIndex] -
    aDisturbance[2].m_pData[frameIndex - 1])
2468            if (varianceDisturbanceDown > 0.0) varianceDisturbanceDown = 0.0
2469            varianceDisturbanceUp += (aDisturbance[2].m_pData[frameIndex - 1] -
    aDisturbance[2].m_pData[frameIndex])
2470            if (varianceDisturbanceUp > 0.0) varianceDisturbanceUp = 0.0
2471        }
2472        varianceDisturbanceUp /= (numberOfSpeechFrames + 0.01)
2473        varianceDisturbanceDown /= (numberOfSpeechFrames + 0.01)
2474        if (varianceDisturbanceUp < -0.85) varianceDisturbanceUp = -0.85
2475        if (varianceDisturbanceDown < -0.85) varianceDisturbanceDown = -0.85
2476
2477        globalScaleCorrection = pow(globalScaleCorrection, 0.06) + 0.02
2478        if (globalScaleCorrection < 1.0) globalScaleCorrection = 1.0
2479
2480        scaleCorrectionQualityPlusOld = 1.0
2481        scaleCorrectionQualityPlusAddedOld = 1.0
2482        scaleCorrectionIntellOld = 1.0
2483        scaleCorrectionMusicOld = 1.0
2484
2485        frameCorrelationTimeDisturbanceAvgCompensation000 = 0.0
2486        frameCorrelationTimeDisturbanceAvgCompensation000silent = 0.0
2487
2488        hulpLowOld = 0.0
2489        hulpHighOld = 0.0
2490        hulpLowOldNarrowband = 0.0
2491        hulpHighOldNarrowband = 0.0
2492        distortedLoudnessTimbrePerFrameNarrowbandAvg = 0.0
2493        distortedLoudnessTimbreHighPerFrameAvg = 0.0
2494        distortedLoudnessTimbreHighPerFrameAvgSilent = 0.0
2495        distortedLoudnessTimbreHighPerFrameAvgActive = 0.0
2496        originalLoudnessTimbrePerFrameDifferenceOld = 0.0
2497        distortedLoudnessTimbrePerFrameDifferenceOld = 0.0
2498        frameCorrelationTimeCompensationDifferenceAvg = 0.0
2499        //FINAL DISTURBANCE CALCULATION PER FRAME FOR ALL Lpqr powers OF WHICH A SUBSET IS USED IN
    THE FINAL MOS MODEL, START WITH COMPENSATION FACTORS
2500
2501        int LastVoicedFrame = -1
2502        XFLOAT sumWeights = 0
2503        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2504        {
2505            if (UseThisFrame[frameIndex])
2506            {
2507
2508                XFLOAT NearUnvoicedSectionEnd = 0
2509
2510                {
2511                    bool FrameIsVoiced = mpPitchVec[frameIndex] > 30
2512                    if (FrameIsVoiced)
2513                    {
2514                        for (i = frameIndex + 1  i<frameIndex + (2.0) + 1 && i <=
    statics->stopFrameIdx && mpPitchVec[i]>30  i++)
2515                        NearUnvoicedSectionEnd = (((0) > ((((1.0) < (1 - (i - frameIndex - 1) /
    (2.0))) ? (1.0) : (1 - (i - frameIndex - 1) / (2.0))))) ? (0) : ((((1.0) <
    (1 - (i - frameIndex - 1) / (2.0))) ? (1.0) : (1 - (i - frameIndex - 1) /
    (2.0)))))
2516                        LastVoicedFrame = frameIndex
2517                    }
```

```
2518                    else if (LastVoicedFrame >= 0)
2519                    {
2520                        NearUnvoicedSectionEnd = (((0) > ((((1.0) < (1 - (frameIndex -
    LastVoicedFrame) / (2.0))) ? (1.0) : (1 - (frameIndex - LastVoicedFrame) /
    (2.0))))) ? (0) : ((((1.0) < (1 - (frameIndex - LastVoicedFrame) / (2.0))) ?
    (1.0) : (1 - (frameIndex - LastVoicedFrame) / (2.0)))))
2521                    }
2522                }
2523
2524                scaleCorrectionQuality = (XFLOAT)(aOriginalTotalPower.m_pData[frameIndex] + (XFLOAT)
    1.0) / (aDistortedTotalPower.m_pData[frameIndex] + (XFLOAT) 1.0)
2525                scaleCorrectionQualityPlus = (XFLOAT)(aOriginalTotalPower.m_pData[frameIndex] +
    (XFLOAT) 100.0) / (aDistortedTotalPower.m_pData[frameIndex] + (XFLOAT) 100.0)
2526                scaleCorrectionQualityPlusAdded = (XFLOAT)(aOriginalTotalPower.m_pData[frameIndex] +
    (XFLOAT) 100.0) / (aDistortedTotalPower.m_pData[frameIndex] + (XFLOAT) 100.0)
2527                scaleCorrectionIntell = scaleCorrectionQuality
2528                scaleCorrectionMusic = scaleCorrectionQuality
2529                if (scaleCorrectionQuality < 1.0)
2530                {
2531                    scaleCorrectionQuality = (XFLOAT) 1.0
2532                    if (aSilent.m_pData[frameIndex])
2533                    {
2534                        scaleCorrectionQualityPlus = (XFLOAT) 1.0 *
    (XFLOAT)pow(scaleCorrectionQualityPlus, (XFLOAT) 0.005)
2535                        scaleCorrectionQualityPlusAdded =
    (XFLOAT)pow(scaleCorrectionQualityPlusAdded, (XFLOAT) 0.05)
2536                        scaleCorrectionIntell = (XFLOAT)pow(scaleCorrectionIntell, (XFLOAT) 0.01)
2537                        scaleCorrectionMusic = (XFLOAT)pow(scaleCorrectionMusic, (XFLOAT) 0.06)
2538                    }
2539                    else
2540                    {
2541                        scaleCorrectionQualityPlus = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlus, (XFLOAT) 0.01)
2542                        scaleCorrectionQualityPlusAdded = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlusAdded, (XFLOAT) 0.01)
2543                        scaleCorrectionIntell = (XFLOAT) 1.0 / (XFLOAT)pow(scaleCorrectionIntell,
    (XFLOAT) 0.004)
2544                        scaleCorrectionMusic = (XFLOAT)pow(scaleCorrectionMusic, (XFLOAT) 0.02)
2545                    }
2546                }
2547                else
2548                {
2549                    scaleCorrectionQuality = 1.0
2550                    if (aSilent.m_pData[frameIndex])
2551                    {
2552                        scaleCorrectionQualityPlus = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlus, (XFLOAT) 0.01)
2553                        scaleCorrectionQualityPlusAdded = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlusAdded, (XFLOAT) 0.15)
2554                        scaleCorrectionIntell = (XFLOAT) 1.0 / (XFLOAT)pow(scaleCorrectionIntell,
    (XFLOAT) 0.003)
2555                        scaleCorrectionMusic = (XFLOAT)pow(scaleCorrectionMusic, (XFLOAT) 0.01)
2556                    }
2557                    else
2558                    {
2559                        scaleCorrectionQualityPlus = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlus, (XFLOAT) 0.005)
2560                        scaleCorrectionQualityPlusAdded = (XFLOAT) 1.0 /
    (XFLOAT)pow(scaleCorrectionQualityPlusAdded, (XFLOAT) 0.008)
2561                        scaleCorrectionIntell = (XFLOAT) 1.0 / pow(scaleCorrectionIntell, (XFLOAT)
    0.002)
2562                        scaleCorrectionMusic = (XFLOAT)pow(scaleCorrectionMusic, (XFLOAT) 0.03)
2563                    }
2564                }
2565                if (frameIndex > (statics->startFrameIdx + 1)) scaleCorrectionIntell *=
    pow(scaleCorrectionIntellOld, 0.1)
2566                scaleCorrectionMusic *= pow(scaleCorrectionMusicOld, 0.2)
2567                scaleCorrectionQualityPlusOld = scaleCorrectionQualityPlus
```

```
2568                scaleCorrectionQualityPlusAddedOld = scaleCorrectionQualityPlusAdded
2569                scaleCorrectionIntellOld = scaleCorrectionIntell
2570                scaleCorrectionMusicOld = scaleCorrectionMusic
2571
2572                overallMovingAvgDisturbance = 0.01*aDisturbance[4].m_pData[frameIndex] +
    0.07*overallMovingAvgDisturbanceOld + 0.92*overallMovingAvgDisturbanceOldOld
2573                overallMovingAvgDisturbanceOldOld = overallMovingAvgDisturbanceOld
2574                overallMovingAvgDisturbanceOld = overallMovingAvgDisturbance
2575
2576                aTimeWeight.m_pData[frameIndex] = 1.0
2577                sumWeights++
2578                if (statics->nrSpeechFrames > 1000)
2579                {
2580                    XFLOAT timeWeightFactor = (statics->nrSpeechFrames - (XFLOAT)1000) /
    (XFLOAT)5500
2581                    if (timeWeightFactor > (XFLOAT) 0.5) timeWeightFactor = (XFLOAT) 0.5
2582                    aTimeWeight.m_pData[frameIndex] = (XFLOAT)(((XFLOAT) 1.0 - timeWeightFactor) +
    timeWeightFactor * (XFLOAT)frameIndex / (XFLOAT)statics->nrSpeechFrames)
2583                }
2584
2585                if (frameIndex > (statics->startFrameIdx + 2))
2586                {
2587                    frameCorrelationTimeOriginal =
    originalLoudnessDensity.FrameCorrelationTime(frameIndex, statics->nrFrames)
2588                    frameCorrelationTimeDistorted =
    distortedLoudnessDensity.FrameCorrelationTime(frameIndex, statics->nrFrames)
2589                    frameCorrelationTimeDisturbance =
    disturbanceDensity.FrameCorrelationTime(frameIndex, statics->nrFrames)
2590                    frameCorrelationTimeDisturbanceAvgCompensation000 +=
    frameCorrelationTimeDisturbance
2591                    if (aSilent.m_pData[frameIndex])
    frameCorrelationTimeDisturbanceAvgCompensation000silent +=
    frameCorrelationTimeDisturbance
2592                }
2593                else
2594                {
2595                    frameCorrelationTimeOriginal = 0.0
2596                    frameCorrelationTimeDistorted = 0.0
2597                    frameCorrelationTimeDisturbance = 0.0
2598                }
2599
2600                frameCorrelationTimeOriginal = 0.1*frameCorrelationTimeOriginal +
    0.9*frameCorrelationTimeOriginalOld
2601                frameCorrelationTimeDistorted = 0.1*frameCorrelationTimeDistorted +
    0.9*frameCorrelationTimeDistortedOld
2602                frameCorrelationTimeOriginalOld = frameCorrelationTimeOriginal
2603                frameCorrelationTimeDistortedOld = frameCorrelationTimeDistorted
2604
2605                if (frameCorrelationTimeOriginal < 0.55) frameCorrelationTimeOriginal = 0.55
2606                if (frameCorrelationTimeDistorted < 0.55) frameCorrelationTimeDistorted = 0.55
2607                if (frameCorrelationTimeOriginal < frameCorrelationTimeDistorted)
2608                    frameCorrelationTimeCompensationDifference = 1.0 +
    (frameCorrelationTimeDistorted - frameCorrelationTimeOriginal)
2609                else
2610                    frameCorrelationTimeCompensationDifference = 1.0
2611                frameCorrelationTimeCompensationDifferenceAvg +=
    frameCorrelationTimeCompensationDifference
2612
2613                frameCorrelationTimeOriginal = pow(frameCorrelationTimeOriginal, 15.0)
2614                frameCorrelationTimeDistorted = pow(frameCorrelationTimeDistorted, 15.0)
2615                frameCorrelationTimeCompensationOriginal = 1.0 - frameCorrelationTimeOriginal
2616                frameCorrelationTimeCompensationDistorted = 1.0 - frameCorrelationTimeDistorted
2617                if (frameCorrelationTimeCompensationOriginal < 0.4)
    frameCorrelationTimeCompensationOriginal = 0.4
2618                if (frameCorrelationTimeCompensationDistorted < 0.4)
    frameCorrelationTimeCompensationDistorted = 0.4
2619
2620                //compensation factor per frame for spectral flatness disturbance (pure tone=0.0 <
```

```
    spectral flatness < 1.0=pure white noise)
2621                frameFlatnessTimeOriginal =
    originalLoudnessDensity.SpectralFlatnessIandM(frameIndex)
2622                frameFlatnessTimeDistorted =
    distortedLoudnessDensity.SpectralFlatnessIandM(frameIndex)
2623
2624                if (maxFreqBarkSource < maxFreqBarkLowLimit) {
2625                    if (aSilent.m_pData[frameIndex]) {
2626                        frameFlatnessDisturbance =
    disturbanceDensity.SpectralFlatnessIandM(frameIndex)
2627                        if (frameFlatnessDisturbance < 0.4) frameFlatnessDisturbance = 0.4
2628                        frameFlatnessDisturbance = (0.9 + frameFlatnessDisturbance)
2629                        frameFlatnessDisturbanceAdded = pow(frameFlatnessDisturbance, 1.3)
2630                    }
2631                    else {
2632                        frameFlatnessDisturbance = disturbanceDensity.SpectralFlatness(frameIndex)
2633                        if (frameFlatnessDisturbance < 0.4) frameFlatnessDisturbance = 0.4
2634                        if (aDisturbance[1].m_pData[frameIndex] > 70.0){
2635                            frameFlatnessDisturbance = (0.8 + frameFlatnessDisturbance)
2636                            frameFlatnessDisturbanceAdded = frameFlatnessDisturbance
2637                        }
2638                        else {
2639                            frameFlatnessDisturbance = (0.9 + frameFlatnessDisturbance)
2640                            frameFlatnessDisturbanceAdded = frameFlatnessDisturbance
2641                        }
2642                    }
2643                }
2644                else {
2645                    if (aSilent.m_pData[frameIndex]) {
2646                        frameFlatnessDisturbance =
    disturbanceDensity.SpectralFlatnessIandM(frameIndex)
2647                        if (frameFlatnessDisturbance < 0.5) frameFlatnessDisturbance = 0.5
2648                        frameFlatnessDisturbance = (0.7 + frameFlatnessDisturbance)
2649                        frameFlatnessDisturbanceAdded = pow(frameFlatnessDisturbance, 1.5)
2650                    }
2651                    else {
2652                        frameFlatnessDisturbance = disturbanceDensity.SpectralFlatness(frameIndex)
2653                        if (frameFlatnessDisturbance < 0.4) frameFlatnessDisturbance = 0.4
2654                        if (aDisturbance[1].m_pData[frameIndex] > 70.0){
2655                            frameFlatnessDisturbance = (0.6 + frameFlatnessDisturbance)
2656                            frameFlatnessDisturbanceAdded = frameFlatnessDisturbance
2657                        }
2658                        else {
2659                            frameFlatnessDisturbance = (0.6 + frameFlatnessDisturbance)
2660                            frameFlatnessDisturbanceAdded = frameFlatnessDisturbance
2661                        }
2662                    }
2663                }
2664
2665                //compensation factor per frame for timbre differences in speech
2666                if (aActiveFreqresponse.m_pData[frameIndex])
2667                {
2668                    hulpLowNarrowband = distortedLoudnessDensity.IntegralLowNarrowband(POLQAHandle,
    frameIndex)
2669                    hulpHighNarrowband = distortedLoudnessDensity.IntegralHighNarrowband(frameIndex)
2670                    distortedLoudnessTimbrePerFrameNarrowband = 4.0*hulpHighNarrowband -
    hulpLowNarrowband
2671                    if (distortedLoudnessTimbrePerFrameNarrowband < 0.0)
    distortedLoudnessTimbrePerFrameNarrowband = 0.0
2672                }
2673                else
2674                {
2675                    distortedLoudnessTimbrePerFrameNarrowband = 0.0
2676                }
2677                distortedLoudnessTimbrePerFrameNarrowbandAvg +=
    distortedLoudnessTimbrePerFrameNarrowband
2678
2679                //compensation factor per frame for timbre differences in speech and noise
```

```
2680                originalLoudnessTimbrePerFrame = (originalLoudnessDensity.IntegralLow2(frameIndex,
    statics->listeningCondition) - originalLoudnessDensity.IntegralHigh2(frameIndex,
    statics->listeningCondition))
2681                distortedLoudnessTimbrePerFrame = (distortedLoudnessDensity.IntegralLow2(frameIndex,
    statics->listeningCondition) - distortedLoudnessDensity.IntegralHigh2(frameIndex,
    statics->listeningCondition))
2682                if ((frameIndex > (statics->startFrameIdx + 2)) &&
    (!aSuperSilent.m_pData[frameIndex]))
2683                {
2684                    originalLoudnessTimbrePerFrameDifference =
    (originalLoudnessDensity.IntegralHigh3(frameIndex - 2) -
    originalLoudnessDensity.IntegralHigh3(frameIndex))
2685                    distortedLoudnessTimbrePerFrameDifference =
    (distortedLoudnessDensity.IntegralHigh3(frameIndex - 2) -
    distortedLoudnessDensity.IntegralHigh3(frameIndex))
2686                }
2687                else
2688                {
2689                    originalLoudnessTimbrePerFrameDifference = 0.0
2690                    distortedLoudnessTimbrePerFrameDifference = 0.0
2691                }
2692                if (originalLoudnessTimbrePerFrameDifference < 0.0)
    originalLoudnessTimbrePerFrameDifference = 0.0
2693                if (distortedLoudnessTimbrePerFrameDifference<1.0)
    distortedLoudnessTimbrePerFrameDifference = 1.0
2694                if (distortedLoudnessTimbrePerFrameDifference>9.0)
    distortedLoudnessTimbrePerFrameDifference = 9.0
2695                originalLoudnessTimbrePerFrameDifference +=
    originalLoudnessTimbrePerFrameDifferenceOld
2696                distortedLoudnessTimbrePerFrameDifference +=
    distortedLoudnessTimbrePerFrameDifferenceOld
2697
2698                originalLoudnessTimbrePerFrameDifferenceOld =
    originalLoudnessTimbrePerFrameDifference
2699                distortedLoudnessTimbrePerFrameDifferenceOld =
    distortedLoudnessTimbrePerFrameDifference
2700
2701                originalLoudnessTimbrePerFrameDifferenceCompensation =
    pow((originalLoudnessTimbrePerFrameDifference + 1.0) /
    (distortedLoudnessTimbrePerFrameDifference + 1.0), 0.005)
2702                if (originalLoudnessTimbrePerFrameDifferenceCompensation < 1.0)
    originalLoudnessTimbrePerFrameDifferenceCompensation = 1.0
2703                originalLoudnessTimbrePerFrameDifferenceCompensationAdd =
    pow((originalLoudnessTimbrePerFrameDifference + 1.0) /
    (distortedLoudnessTimbrePerFrameDifference + 1.0), 0.04)
2704                if (originalLoudnessTimbrePerFrameDifferenceCompensationAdd < 0.95)
    originalLoudnessTimbrePerFrameDifferenceCompensationAdd = 0.95
2705
2706                differenceInLoudnessTimbrePerFrame = fabs(distortedLoudnessTimbrePerFrame -
    originalLoudnessTimbrePerFrame)
2707                hulpLow = 0.8*distortedLoudnessDensity.IntegralLow2(frameIndex,
    statics->listeningCondition) + 0.2*hulpLowOld
2708                hulpHigh = 0.8*distortedLoudnessDensity.IntegralHigh2(frameIndex,
    statics->listeningCondition) + 0.2*hulpHighOld
2709                hulpLowOld = hulpLow
2710                hulpHighOld = hulpHigh
2711
2712                distortedLoudnessTimbrePerFrame = fabs(hulpLow - 2.0*hulpHigh) - 140.0
2713                if (distortedLoudnessTimbrePerFrame < 0.0) distortedLoudnessTimbrePerFrame = 0.0
2714                distortedLoudnessTimbreHighPerFrame = 2.3*hulpHigh - hulpLow
2715                if (distortedLoudnessTimbreHighPerFrame < 0.0) distortedLoudnessTimbreHighPerFrame =
    0.0
2716
2717                if (aSilent.m_pData[frameIndex])
2718                {
2719                    distortedLoudnessTimbreHighPerFrameAvgSilent +=
    distortedLoudnessTimbreHighPerFrame
2720                }
```

```
2721                else
2722                {
2723                    distortedLoudnessTimbreHighPerFrameAvgActive +=
    distortedLoudnessTimbreHighPerFrame
2724                }
2725                distortedLoudnessTimbreHighPerFrameAvg += distortedLoudnessTimbreHighPerFrame
2726                XFLOAT silentWeight = pow((numberOfSuperSilentFrames + 10.0) / (numberOfSpeechFrames
    + 10.0), 0.1)
2727
2728                int jj=0
2729                //FINAL DISTURBANCE CALCULATION IN EACH FRAME FOR AL Lp's OVER FREQUENCY OF WHICH A
    SUBSET IS USED IN THE FINAL MOS MODEL
2730                for (jj = 0  jj < NUMBER_OF_POWERS_OVER_FREQ  jj++) {
2731                    if (frameIndex >(statics->startFrameIdx + 3))
2732                        if ((aDisturbance[jj].m_pData[frameIndex - 1] > 60.0) &&
    (aDisturbance[jj].m_pData[frameIndex - 2] > 60.0) &&
    (aAddedDisturbance[jj].m_pData[frameIndex - 1] > 300.0) &&
    (aAddedDisturbance[jj].m_pData[frameIndex - 2] > 300.0))
    aDisturbance[jj].m_pData[frameIndex] /= 2.0
2733
2734                    aDisturbance[jj].m_pData[frameIndex] *=
    0.85*originalLoudnessTimbrePerFrameDifferenceCompensation
2735                    aAddedDisturbance[jj].m_pData[frameIndex] *=
    0.84*originalLoudnessTimbrePerFrameDifferenceCompensationAdd
2736
2737                    //LOW LEVEL compensation for global zero output in case of severe time clipping
2738                    aDisturbance[jj].m_pData[frameIndex] /= globalScaleCorrectionActive
2739                    aAddedDisturbance[jj].m_pData[frameIndex] /= globalScaleCorrectionActiveAdded
2740
2741                    aDisturbance[jj].m_pData[frameIndex] /= pow((aPowerRatioaAvg + 1.0), 0.02)
2742                    aAddedDisturbance[jj].m_pData[frameIndex] /= pow((aPowerRatioaAvg + 1.0), 0.13)
2743                    aDisturbance[jj].m_pData[frameIndex] *= scaleCorrectionQualityPlus
2744                    aAddedDisturbance[jj].m_pData[frameIndex] *= scaleCorrectionQualityPlusAdded
2745
2746                    //BEGIN compensation per frame factor for correlation differences, FRAME REPEAT
    repeat modelling
2747                    aDisturbance[jj].m_pData[frameIndex] *=
    1.04*pow(frameCorrelationTimeCompensationDifference, 1.1)
2748                    aAddedDisturbance[jj].m_pData[frameIndex] *= 1.04
2749
2750                    aDisturbance[jj].m_pData[frameIndex] /= pow(hulpLevel, 0.4)
2751                    aAddedDisturbance[jj].m_pData[frameIndex] *= pow(hulpLevel, 0.4)
2752
2753                    //compensation factor per frame for TIMBRE differences
2754                    if (frameIndex > (statics->startFrameIdx + 10) &&
    aActiveFreqresponse.m_pData[frameIndex])
2755                    {
2756                        aAddedDisturbance[jj].m_pData[frameIndex] *= pow((1.0 +
    distortedLoudnessTimbrePerFrameNarrowband), 0.05)
2757                    }
2758
2759                    if (frameIndex > (statics->startFrameIdx))
2760                    {
2761                        if (maxFreqBarkSource < maxFreqBarkHighLimit)
2762                        {
2763                            if (maxFreqBarkSource<maxFreqBarkLowLimit)
2764                            {
2765                                aAddedDisturbance[jj].m_pData[frameIndex] +=
    0.006*distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbreAdd
2766                                aDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
    distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.06)
2767                                aAddedDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
    distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.02)
2768                            }
2769                            else
2770                            {
2771                                aAddedDisturbance[jj].m_pData[frameIndex] +=
    0.006*distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbreAdd
```

```
2772                                aDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.07)
2773                                aAddedDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.03)
2774                            }
2775                        }
2776                        else
2777                        {
2778                            aAddedDisturbance[jj].m_pData[frameIndex] +=
0.006*distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbreAdd
2779                            aDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.08)
2780                            aAddedDisturbance[jj].m_pData[frameIndex] /= pow((1.0 +
distortedLoudnessTimbreHighPerFrame / noiseIndicatorTimbre), 0.02)
2781                        }
2782                    }
2783
2784                    //compensation factor for SPECTRAL FLATNESS
2785                    if (aSilent.m_pData[frameIndex])
2786                    {
2787                        hulp = (frameFlatnessDisturbance /
frameFlatnessDisturbanceAvgCompensationSilent)
2788                        if (aListeningCondition == STANDARD_IRS) {
2789                            aDisturbance[jj].m_pData[frameIndex] *= (1.1*hulp)
2790                        }
2791                        else {
2792                            if (hulp > 1.2) hulp = 1.2
2793                            aDisturbance[jj].m_pData[frameIndex] *= hulp
2794                        }
2795                        hulp = (frameFlatnessDisturbanceAdded /
frameFlatnessDisturbanceAvgCompensationAddedSilent)
2796                        if (aListeningCondition == STANDARD_IRS) {
2797                            aAddedDisturbance[jj].m_pData[frameIndex] *= (1.1*hulp)
2798
2799                        }
2800                        else {
2801                            if (hulp > 1.7) hulp = 1.7
2802                            aAddedDisturbance[jj].m_pData[frameIndex] *= hulp
2803                        }
2804
2805                    }
2806                    else
2807                    {
2808                        aDisturbance[jj].m_pData[frameIndex] *= (frameFlatnessDisturbance /
frameFlatnessDisturbanceAvgCompensationActive)
2809                        aAddedDisturbance[jj].m_pData[frameIndex] *= (frameFlatnessDisturbanceAdded
/ frameFlatnessDisturbanceAvgCompensationAddedActive)
2810                    }
2811
2812                    //compensation for NOISE CONTRAST IN SILENT PERIODS
2813                    if (aSuperSilent.m_pData[frameIndex]) aDisturbance[jj].m_pData[frameIndex] *=
pow(noiseContrastMax1, 0.8)
2814
2815                    //compensation for delay ALIGN JUMPS
2816                    if (FrameFlags[frameIndex] & 0x0000001) {
2817                        for (count = (-noiseIndicatorAlignJumpsIntWB)  count < 0  count++)
2818                        {
2819                            hulp = (1.0 - 1.0*count /
(delayJumpCompWB*noiseIndicatorAlignJumpsIntWB)) / (1.0 + 1.0 /
delayJumpCompWB)
2820                            if (frameIndex>(statics->startFrameIdx - count))
2821                            {
2822                                if (maxFreqBarkSource < maxFreqBarkHighLimit) {
2823                                    aDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp, 0.3)
2824                                    aAddedDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp,
0.3)
2825                                } else {
2826                                    aDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp, 0.6)
```

```
2827                                    aAddedDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp,
    0.6)
2828                                }
2829                            }
2830                        }
2831                        for (count = 0  count < (noiseIndicatorAlignJumpsIntWB + 1)  count++)
2832                        {
2833                            hulp = (1.0 + 1.0*count /
    (delayJumpCompWB*noiseIndicatorAlignJumpsIntWB)) / (1.0 + 1.0 /
    delayJumpCompWB)
2834                            if (frameIndex < (statics->stopFrameIdx - count))
2835                            {
2836                                if (maxFreqBarkSource < maxFreqBarkHighLimit) {
2837                                    aDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp, 0.3)
2838                                    aAddedDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp,
    0.3)
2839                                } else {
2840                                    aDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp, 0.6)
2841                                    aAddedDisturbance[jj].m_pData[frameIndex + count] *= pow(hulp,
    0.6)
2842                                }
2843                            }
2844                        }
2845                    }
2846                    if (aDisturbance[jj].m_pData[frameIndex] < 2.0)
    aDisturbance[jj].m_pData[frameIndex] = 2.0
2847                    if (aAddedDisturbance[jj].m_pData[frameIndex] < 2.0)
    aAddedDisturbance[jj].m_pData[frameIndex] = 2.0
2848
2849                    //CLIP TO MAXIMUM DEGRADATION
2850                    if (aDisturbance[jj].m_pData[frameIndex]         >
    1.07*maxDisturbance*globalScaleCorrectionIntellLevelCorrectionForMaximumD)
    aDisturbance[jj].m_pData[frameIndex] =
    1.07*maxDisturbance*globalScaleCorrectionIntellLevelCorrectionForMaximumD
2851                    if (aAddedDisturbance[jj].m_pData[frameIndex] >
    (2.0*maxDisturbance*globalScaleCorrectionIntellLevelCorrectionForMaximumA))
    aAddedDisturbance[jj].m_pData[frameIndex] =
    2.0*maxDisturbance*globalScaleCorrectionIntellLevelCorrectionForMaximumA
2852
2853                    hulp1 = (aOriginalLoudness.m_pData[frameIndex]) - 27.0
2854                    if (hulp1 < 1.0) hulp1 = 1.0
2855                    hulp1 = pow(hulp1, 0.3)
2856                    aDisturbance[jj].m_pData[frameIndex] /= hulp1
2857
2858                    hulp1 = (aOriginalLoudness.m_pData[frameIndex]) - 18.0
2859                    if (hulp1 < 0.95) hulp1 = 0.95
2860                    hulp1 = pow(hulp1, 0.2*aPureFrqLoudnessMeanCompensation)
2861                    aAddedDisturbance[jj].m_pData[frameIndex] /= hulp1
2862                    hulp1 = (aDistortedLoudness.m_pData[frameIndex]) - 23.0
2863                    if (hulp1 < 1.0) hulp1 = 1.0
2864                    hulp1 = pow(hulp1, 0.18*aPureFrqLoudnessMeanCompensation)
2865                    aAddedDisturbance[jj].m_pData[frameIndex] /= hulp1
2866                    aDisturbance[jj].m_pData[frameIndex] *= pow(fractionOfUsedFrames, 0.75)
2867                    aAddedDisturbance[jj].m_pData[frameIndex] *= pow(fractionOfUsedFrames, 0.7)
2868                    aAddedDisturbance[jj][frameIndex] *= pow(fractionOfSilentFrames, 0.05)
2869
2870                    //compensation for DISTURBANCE VARIANCE
2871                    aDisturbance[jj].m_pData[frameIndex] *= pow((0.98*varianceDisturbanceUp + 1.05),
    0.7)
2872                    aAddedDisturbance[jj].m_pData[frameIndex] *= pow((varianceDisturbanceUp + 1.0),
    0.9)
2873                    aDisturbance[jj].m_pData[frameIndex] *= pow((varianceDisturbanceDown + 1.0),
    0.1)
2874
2875                    if (frameIndex > (statics->startFrameIdx + 10))
2876                    {
2877                        //compensation for LOUDNESS JUMPS
2878
```

```
2879                        hulp1 = (aOriginalLoudness.m_pData[frameIndex] + 0.0001) /
    (aOriginalLoudness.m_pData[frameIndex - 1] + 0.0001)
2880                        if (hulp1 < 1.0) hulp1 = 1.0
2881                        aDisturbance[jj].m_pData[frameIndex] /= pow((hulp1 + 1.0), 0.04)
2882                        aAddedDisturbance[jj].m_pData[frameIndex] *= pow((hulp1 + 1.0), 0.01)
2883                        hulp = aDistortedLoudness.m_pData[frameIndex]
2884                        if (hulp < 11.0) hulp = 11.0
2885                        hulp = pow(hulp, 3.0)
2886                        hulp2 = (aDistortedLoudness.m_pData[frameIndex] * hulp + 1.0e-8) /
    (aDistortedLoudness.m_pData[frameIndex - 1] * hulp + 1.0e-8)
2887                        if (hulp2 < 1.0) hulp2 = 1.0
2888                        aAddedDisturbance[jj].m_pData[frameIndex] /= pow((hulp2 + 1.0), 0.04)
2889                        hulp1 = (aOriginalLoudness.m_pData[frameIndex - 1] * hulp + 1.0e-8) /
    (aOriginalLoudness.m_pData[frameIndex] * hulp + 1.0e-8)
2890                        if (hulp1 < 1.0) hulp1 = 1.0
2891                        aDisturbance[jj].m_pData[frameIndex] *= pow((hulp1 + 1.0), 0.03)
2892
2893                        if (frameIndex<statics->stopFrameIdx)
2894                        {
2895                            hulp1 = mpPitchVec[frameIndex]
2896                            hulp2 = mpPitchVecDeg[frameIndex]
2897                            if (hulp1<40.0) hulp1 = 40.0
2898                            if (hulp1>400.0) hulp1 = 400.0
2899                            if (hulp2<40.0) hulp2 = 40.0
2900                            if (hulp2>400.0) hulp2 = 400.0
2901                            hulp = (hulp2 + 1.0) / (hulp1 + 1.0)
2902                            if (hulp>1.0) hulp = 1.0 / hulp
2903
2904                            if (mpPitchVec[frameIndex] <0.1 && mpPitchVecDeg[frameIndex]>20.0)
2905                            {
2906                                aDisturbance[jj].m_pData[frameIndex] /= 1.07
2907                                aAddedDisturbance[jj].m_pData[frameIndex] /= 1.07
2908                            }
2909                            if (mpPitchVec[frameIndex] > 20.0 && mpPitchVecDeg[frameIndex] < 0.1)
2910                            {
2911                                aDisturbance[jj].m_pData[frameIndex] *= 1.07
2912                                aAddedDisturbance[jj].m_pData[frameIndex] *= 1.07
2913                            }
2914                        }
2915                        if (aDisturbance[jj].m_pData[frameIndex] < 2.0)
    aDisturbance[jj].m_pData[frameIndex] = 2.0
2916                        if (aAddedDisturbance[jj].m_pData[frameIndex] < 2.0)
    aAddedDisturbance[jj].m_pData[frameIndex] = 2.0
2917                    }
2918                }
2919
2920            }
2921            else
2922            {
2923                for (int jj = 0  jj < NUMBER_OF_POWERS_OVER_FREQ  jj++)
2924                {
2925                    aDisturbance[jj].m_pData[frameIndex] = 0.0
2926                    aAddedDisturbance[jj].m_pData[frameIndex] = 0.0
2927                }
2928            }
2929        }
2930
2931        distortedLoudnessTimbrePerFrameNarrowbandAvg /= (numberOfaActiveFreqresponse + 0.1)
2932        distortedLoudnessTimbrePerFrameNarrowbandAvg000 =
    distortedLoudnessTimbrePerFrameNarrowbandAvg
2933        if (distortedLoudnessTimbrePerFrameNarrowbandAvg000<250.0)
    distortedLoudnessTimbrePerFrameNarrowbandAvg000 = 250.0
2934        if (distortedLoudnessTimbrePerFrameNarrowbandAvg000>800.0)
    distortedLoudnessTimbrePerFrameNarrowbandAvg000 = 800.0
2935        distortedLoudnessTimbrePerFrameNarrowbandAvg000 = 1.0 /
    pow((distortedLoudnessTimbrePerFrameNarrowbandAvg000 / 500.0), 0.02)
2936
2937        distortedLoudnessTimbreHighPerFrameAvg /= (numberOfSpeechFrames + 0.1)
```

```
2938        distortedLoudnessTimbreHighPerFrameAvgSilent /= (numberOfSilentFrames + 0.1)
2939
2940        distortedLoudnessTimbreHighPerFrameAvgActive /= (numberOfActiveFrames + 0.1)
2941        distortedLoudnessTimbreHighPerFrameAvgActive000 =
    distortedLoudnessTimbreHighPerFrameAvgActive
2942        if (distortedLoudnessTimbreHighPerFrameAvgActive<15.0)
    distortedLoudnessTimbreHighPerFrameAvgActive = 15.0
2943        if (distortedLoudnessTimbreHighPerFrameAvgActive>60.0)
    distortedLoudnessTimbreHighPerFrameAvgActive = 60.0
2944        distortedLoudnessTimbreHighPerFrameAvgActive = 1 /
    distortedLoudnessTimbreHighPerFrameAvgActive
2945        distortedLoudnessTimbreHighPerFrameAvgActive *= 2.5
2946        frameCorrelationTimeCompensationDifferenceAvg /= (numberOfSpeechFrames + 0.1)
2947
2948        distortedLoudnessTimbreHighPerFrameAvgActive000 -= 250.0
2949        if (distortedLoudnessTimbreHighPerFrameAvgActive000 > 80.0)
    distortedLoudnessTimbreHighPerFrameAvgActive000 = 80.0
2950        distortedLoudnessTimbreHighPerFrameAvgActive000 /= 3000.0
2951        distortedLoudnessTimbreHighPerFrameAvgActive000 +=
    distortedLoudnessTimbreHighPerFrameAvgActive
2952
2953        frameCorrelationTimeDisturbanceAvgCompensation000silent /= (numberOfSilentFrames + 0.1)
2954        frameCorrelationTimeDisturbanceAvgCompensation000silent = 1.01 /
    pow((frameCorrelationTimeDisturbanceAvgCompensation000silent + 1.0), 0.05)
2955
2956        frameCorrelationTimeDisturbanceAvgCompensation000 /= (numberOfSpeechFrames + 0.1)
2957        frameCorrelationTimeDisturbanceAvgCompensation000 = 1.0 /
    pow((frameCorrelationTimeDisturbanceAvgCompensation000 + 0.9), 0.05)
2958
2959        distortedLoudnessTimbreHighPerFrameAvgXnoiseIndicatorTimbre = pow((0.95 +
    distortedLoudnessTimbreHighPerFrameAvg*noiseIndicatorTimbre), 0.03)
2960
2961        avgPitchLoudFrames000 = 0.0
2962        count0 = 0
2963        avgPitchLoudFramesRise000 = 0.0
2964        avgPitchLoudFramesDrop000 = 0.0
2965        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
2966        {
2967            if (frameIndex > (statics->startFrameIdx + 5))
2968            {
2969                if (aOriginalLoudness.m_pData[frameIndex - 5] > 1.0 &&
    aOriginalLoudness.m_pData[frameIndex - 4] > 1.0 &&
2970                    aOriginalLoudness.m_pData[frameIndex - 3] > 1.0 &&
    aOriginalLoudness.m_pData[frameIndex - 2] > 1.0 &&
2971                    aOriginalLoudness.m_pData[frameIndex - 1] > 1.0 &&
    aOriginalLoudness.m_pData[frameIndex] > 1.0 &&
2972                    mpPitchVec[frameIndex - 5] > 30.0 && mpPitchVec[frameIndex - 4] > 30.0 &&
    mpPitchVec[frameIndex - 3] > 30.0 &&
2973                    mpPitchVec[frameIndex - 2] > 30.0 && mpPitchVec[frameIndex - 1] > 30.0 &&
    mpPitchVec[frameIndex] > 30.0)
2974                {
2975                    avgPitchLoudFrames000 += mpPitchVec[frameIndex]
2976                    count0++
2977                    hulp1 = (mpPitchVec[frameIndex - 5] + mpPitchVec[frameIndex - 4] +
    mpPitchVec[frameIndex - 3]) / 3.0
2978                    hulp2 = (mpPitchVec[frameIndex - 2] + mpPitchVec[frameIndex - 1] +
    mpPitchVec[frameIndex]) / 3.0
2979                    if (hulp2 > hulp1)
2980                    {
2981                        avgPitchLoudFramesRise000 += (hulp2 - hulp1)
2982                    }
2983                    else
2984                    {
2985                        avgPitchLoudFramesDrop000 += (hulp1 - hulp2)
2986                    }
2987                }
2988            }
2989
```

```
2990            for (i = 0  i < NUMBER_OF_POWERS_OVER_FREQ  i++) {
2991                if (frameIndex > 10 && aActiveFreqresponse.m_pData[frameIndex]) {
2992                    aAddedDisturbance[i].m_pData[frameIndex] *=
    distortedLoudnessTimbreHighPerFrameAvgXnoiseIndicatorTimbre
2993                }
2994            }
2995        }
2996        avgPitchLoudFrames000 /= (count0 + 0.1)
2997        avgPitchLoudFramesCompensation = pow((avgPitchLoudFrames000 + 1.0), 0.006) / 1.01
2998        avgPitchLoudFramesRise000 /= (count0 + 0.1)
2999        avgPitchLoudFramesDrop000 /= (count0 + 0.1)
3000        if (avgPitchLoudFrames000<50.0) avgPitchLoudFrames000 = 50.0
3001        if (avgPitchLoudFrames000>170.0) avgPitchLoudFrames000 = 170.0
3002        if (avgPitchLoudFramesRise000<1.0) avgPitchLoudFramesRise000 = 1.0
3003        if (avgPitchLoudFramesDrop000<1.0) avgPitchLoudFramesDrop000 = 1.0
3004        if (avgPitchLoudFramesRise000>20.0) avgPitchLoudFramesRise000 = 20.0
3005        if (avgPitchLoudFramesDrop000>20.0) avgPitchLoudFramesDrop000 = 20.0
3006
3007        //END POLQAMAIN PART 2
3008
3009        hulp1 = 0.0
3010        hulp2 = 0.0
3011        for (frameIndex = (statics->startFrameIdx + 5)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)
3012        {
3013            if (UseThisFrame[frameIndex])
3014            {
3015                if (aActiveFreqresponse.m_pData[frameIndex])
3016                {
3017                    hulp1 += originalPitchPowerDensity.Total((frameIndex), 50.0, 3500.0)
3018                    hulp2 += distortedPitchPowerDensity.Total((frameIndex), 50.0, 3500.0)
3019                }
3020            }
3021        }
3022        hulp1 /= (numberOfaActiveFreqresponse + 1.0)
3023        hulp2 /= (numberOfaActiveFreqresponse + 1.0)
3024        scale = 1.0e9/(hulp1+1.0)
3025        for (frameIndex = (statics->startFrameIdx+5)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)     {
3026            if (UseThisFrame[frameIndex]) {
3027                originalPitchPowerDensity. MultiplyWith (frameIndex, scale)
3028            }
3029        }
3030        scale = 1.0e9/(hulp2+1.0)
3031        for (frameIndex = (statics->startFrameIdx+5)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)     {
3032            if (UseThisFrame[frameIndex]) {
3033                distortedPitchPowerDensity. MultiplyWith (frameIndex, scale)
3034            }
3035        }
3036
3037        count = 0
3038        scaleDistortion = 0.0
3039        scaleDistortion2 = 0.0
3040        oldOldScale = 1.0
3041        oldScale = 1.0
3042        for (frameIndex = (statics->startFrameIdx + 18)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)     {
3043            if (UseThisFrame[frameIndex]) {
3044                aOriginalTotalPower.m_pData[frameIndex] =
    originalPitchPowerDensity.Total((frameIndex), 200.0, 3000.0)
3045                aDistortedTotalPower.m_pData[frameIndex] =
    distortedPitchPowerDensity.Total((frameIndex), 200.0, 3000.0)
3046                scale = 1.0
3047                count1 = 0
3048
3049                for (i = -10  i < 0  i++) {
3050                    if ((aOriginalTotalPower.m_pData[frameIndex + i]>6.0e6)) {
```

```
3051                        count1++
3052                        hulp1 = originalPitchPowerDensity.Total((frameIndex + i), 200.0, 3500.0)
3053                        hulp2 = distortedPitchPowerDensity.Total((frameIndex + i), 200.0, 3500.0)
3054                        scale *= (hulp1 + (XFLOAT) 4.0e4) / (hulp2 + (XFLOAT) 4.0e4)
3055                    }
3056                }
3057
3058                scale = pow(scale, 4.0 / (count1 + 1.0))
3059                if (scale > 22.0) scale = 22.0
3060                if ((aOriginalTotalPower.m_pData[frameIndex]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 1]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 2]>1.0e7)
3061                    && (aOriginalTotalPower.m_pData[frameIndex - 3]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 4]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 5]>1.0e7)
3062                    && (aOriginalTotalPower.m_pData[frameIndex - 6]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 7]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 8]>1.0e7)
3063                    && (aDistortedTotalPower.m_pData[frameIndex]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 1]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 2]>1.0e7)
3064                    && (aDistortedTotalPower.m_pData[frameIndex - 3]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 4]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 5]>1.0e7)
3065                    && (aDistortedTotalPower.m_pData[frameIndex - 6]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 7]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 8]>1.0e7))
3066                {
3067                    hulp = aDisturbance[2].m_pData[frameIndex] - 10.0
3068                    if (hulp<0.0) hulp = 0.0
3069                    hulp = pow(hulp, 4.0)
3070                    hulp3 = fabs(oldOldScale - scale) / (hulp + 1.0e-3) - 300.0
3071                    if (hulp3<1.0) hulp3 = 1.0
3072                    scaleDistortion += hulp3
3073
3074                    hulp3 = fabs(oldOldScale - scale) / (1.0e-3)
3075                    if (hulp3<0.0) hulp3 = 0.0
3076                    scaleDistortion2 += hulp3
3077
3078                    count++
3079
3080                    oldOldScale = oldScale
3081                    oldScale = scale
3082                }
3083            }
3084        }
3085        scaleDistortion /= (count + 0.1)
3086        scaleDistortion2 /= (count + 0.1)
3087
3088        count = 0
3089        scaleDistortion3 = 0.0
3090        scaleDistortion4 = 0.0
3091        oldOldScale = 1.0
3092        oldScale = 1.0
3093        for (frameIndex = (statics->startFrameIdx + 18)  frameIndex <= (statics->stopFrameIdx)
    frameIndex++)     {
3094            if (UseThisFrame[frameIndex]) {
3095                aOriginalTotalPower.m_pData[frameIndex] =
    originalPitchPowerDensity.Total((frameIndex), 300.0, 4000.0)
3096                aDistortedTotalPower.m_pData[frameIndex] =
    distortedPitchPowerDensity.Total((frameIndex), 300.0, 4000.0)
3097                scale = 1.0
3098                count1 = 0
3099
3100                for (i = -10  i < 0  i++) {
3101                    if ((aOriginalTotalPower.m_pData[frameIndex + i]>3.0e6)) {
3102                        count1++
3103                        hulp1 = originalPitchPowerDensity.Total((frameIndex + i), 200.0, 3500.0)
```

```
3104                        hulp2 = distortedPitchPowerDensity.Total((frameIndex + i), 200.0, 3500.0)
3105                        scale *= (hulp2 + (XFLOAT) 5.0e4) / (hulp1 + (XFLOAT) 5.0e4)
3106                    }
3107                }
3108
3109                scale = pow(scale, 1.0 / (count1 + 1.0))
3110                if (scale > 15.0) scale = 15.0
3111                if ((aOriginalTotalPower.m_pData[frameIndex]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 1]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 2]>1.0e7)
3112                    && (aOriginalTotalPower.m_pData[frameIndex - 3]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 4]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 5]>1.0e7)
3113                    && (aOriginalTotalPower.m_pData[frameIndex - 6]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 7]>1.0e7) &&
    (aOriginalTotalPower.m_pData[frameIndex - 8]>1.0e7)
3114                    && (aDistortedTotalPower.m_pData[frameIndex]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 1]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 2]>1.0e7)
3115                    && (aDistortedTotalPower.m_pData[frameIndex - 3]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 4]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 5]>1.0e7)
3116                    && (aDistortedTotalPower.m_pData[frameIndex - 6]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 7]>1.0e7) &&
    (aDistortedTotalPower.m_pData[frameIndex - 8]>1.0e7))
3117                {
3118
3119                    hulp = aDisturbance[2].m_pData[frameIndex] - 10.0
3120                    if (hulp<0.0) hulp = 0.0
3121                    hulp = pow(hulp, 4)
3122                    hulp3 = fabs(oldOldScale - scale) / (hulp + 1.0e-3) - 300.0
3123                    if (hulp3<1.0) hulp3 = 1.0
3124                    scaleDistortion3 += hulp3
3125
3126                    hulp3 = fabs(oldOldScale - scale) / (1.0e-3)
3127                    if (hulp3<0.0) hulp3 = 0.0
3128                    scaleDistortion4 += hulp3
3129
3130                    count++
3131
3132                    oldOldScale = oldScale
3133                    oldScale = scale
3134                }
3135            }
3136        }
3137        scaleDistortion3 /= (count + 0.1)
3138        scaleDistortion4 /= (count + 0.1)
3139
3140        scaleDistortion -= 30.0
3141        if (scaleDistortion<1.0)  scaleDistortion = 1.0
3142
3143        scaleDistortion2 -= 30.0
3144        if (scaleDistortion2<1.0) scaleDistortion2 = 1.0
3145
3146        scaleDistortion3 -= 30.0
3147        if (scaleDistortion3<1.0)  scaleDistortion3 = 1.0
3148
3149        int numberOfSilentInstertions = 1
3150        for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
3151            if ((!(pMarkSectionFlags[frameIndex] & 8)) && (!(pMarkSectionFlags[frameIndex] & 2)))
    numberOfSilentInstertions++
3152        }
3153        XFLOAT numberOfSilentInstertionsCompensation = pow(numberOfSilentInstertions / 2.5, 0.02)
3154        XFLOAT numberOfSilentInstertionsCompensationAdded = pow(numberOfSilentInstertions / 2.5,
    0.04)
3155
3156        hulpDelayMem = 0.0
```

```
3157        XFLOAT freqShiftChanges
3158        int hulpCountMin = -((int)(((5.0*0.75 + 3.5) > 0) ? (5.0*0.75 + 3.5)+0.5f : (5.0*0.75 +
    3.5)-0.5f))
3159        int hulpCountMax = ((int)(((7.5*0.75 - 1.75) > 0) ? (7.5*0.75 - 1.75)+0.5f : (7.5*0.75 -
    1.75)-0.5f))
3160        int jjMax = 6
3161        freqShiftChangesAvg = 0.0
3162        for (frameIndex = (statics->startFrameIdx+12)  frameIndex <= (statics->stopFrameIdx-4)
    frameIndex++) {
3163                hulp0 = ( (XFLOAT) (abs(pOverviewHolder->m_DelayPerFrame[frameIndex + 1] -
    pOverviewHolder->m_DelayPerFrame[frameIndex]))*1000.0/ (XFLOAT)
    pOverviewHolder->m_SampleFrequencyHz) - 1.0
3164                if (hulp0<0.0) hulp0 = 0.0
3165                if (hulp0>8.0) hulp0 = 8.0
3166                hulp1 = ( (XFLOAT) (abs(pOverviewHolder->m_DelayPerFrame[frameIndex] -
    pOverviewHolder->m_DelayPerFrame[frameIndex - 1]))*1000.0/ (XFLOAT)
    pOverviewHolder->m_SampleFrequencyHz) - 1.0
3167                if (hulp1<0.0) hulp1 = 0.0
3168                if (hulp1>8.0) hulp1 = 8.0
3169                hulp2 = ( (XFLOAT) (abs(pOverviewHolder->m_DelayPerFrame[frameIndex - 1] -
    pOverviewHolder->m_DelayPerFrame[frameIndex - 2]))*1000.0/ (XFLOAT)
    pOverviewHolder->m_SampleFrequencyHz) - 1.0
3170                if (hulp2<0.0) hulp2 = 0.0
3171                if (hulp2>8.0) hulp2 = 8.0
3172                hulp3 = ( (XFLOAT) (abs(pOverviewHolder->m_DelayPerFrame[frameIndex - 2] -
    pOverviewHolder->m_DelayPerFrame[frameIndex - 3]))*1000.0/ (XFLOAT)
    pOverviewHolder->m_SampleFrequencyHz) - 1.0
3173                if (hulp3<0.0) hulp3 = 0.0
3174                if (hulp3>8.0) hulp3 = 8.0
3175                hulp4 = ( (XFLOAT) (abs(pOverviewHolder->m_DelayPerFrame[frameIndex - 3] -
    pOverviewHolder->m_DelayPerFrame[frameIndex - 4]))*1000.0/ (XFLOAT)
    pOverviewHolder->m_SampleFrequencyHz) - 1.0
3176                if (hulp4<0.0) hulp4 = 0.0
3177                if (hulp4>8.0) hulp4 = 8.0
3178                hulpDelay = hulp0 + hulp1 + hulp2 + hulp3 + hulp4
3179                if (hulpDelay<0.0) hulpDelay = 0.0
3180                if (hulpDelay>(8.0)) hulpDelay = (8.0)
3181                hulpDelay *= (XFLOAT)0.028*(pow(aDistortedLoudness.m_pData[frameIndex - 5],
    (XFLOAT)1.8) + (XFLOAT)0.001)
3182                hulpDelay *= (XFLOAT)0.028*(pow(aDistortedLoudness.m_pData[frameIndex - 3],
    (XFLOAT)1.8) + (XFLOAT)0.001)
3183                hulpDelay *= (XFLOAT)0.028*(pow(aDistortedLoudness.m_pData[frameIndex - 1],
    (XFLOAT)1.8) + (XFLOAT)0.001)
3184                hulpDelay *= (XFLOAT)0.028*(pow(aDistortedLoudness.m_pData[frameIndex + 1],
    (XFLOAT)1.8) + (XFLOAT)0.001)
3185                hulpDelay *= compensation_FFLAG_SHORT_DELAY_CHANGE
3186                hulpDelayMem = hulpDelay
3187
3188                freqShiftChanges = 0.0
3189
3190                for (hulpCount = -1  hulpCount < 1  hulpCount++) freqShiftChanges +=
    1.0*abs(bestSpectrumShift[frameIndex + hulpCount] - bestSpectrumShift[frameIndex +
    hulpCount - 1])
3191                freqShiftChangesAvg += freqShiftChanges
3192
3193                for (i = 0  i < NUMBER_OF_POWERS_OVER_FREQ  i++) {
3194
3195                    if (maxFreqBarkSource < maxFreqBarkHighLimit) {
3196                        for (hulpCount = hulpCountMin  hulpCount < hulpCountMax  hulpCount++) {
3197                            aDisturbance[i].m_pData[frameIndex + hulpCount] -= 0.00006*hulpDelay
3198                            aAddedDisturbance[i].m_pData[frameIndex + hulpCount] -=
    0.00006*hulpDelay
3199                            if (aDisturbance[i].m_pData[frameIndex + hulpCount] < 0.0)
    aDisturbance[i].m_pData[frameIndex + hulpCount] = 0.0
3200                            if (aAddedDisturbance[i].m_pData[frameIndex + hulpCount] < 0.0)
    aAddedDisturbance[i].m_pData[frameIndex + hulpCount] = 0.0
3201                        }
3202                        for (int jj = 1  jj < jjMax  jj++) aDisturbance[i].m_pData[frameIndex - jj]
```

```
    /= pow((freqShiftChanges + 1.0), 0.12)
3203                    }
3204                    else {
3205                        for (hulpCount = hulpCountMin  hulpCount < hulpCountMax  hulpCount++) {
3206                            aDisturbance[i].m_pData[frameIndex + hulpCount] -= 0.00006*hulpDelay
3207                            aAddedDisturbance[i].m_pData[frameIndex + hulpCount] -=
    0.00006*hulpDelay
3208                            if (aDisturbance[i].m_pData[frameIndex + hulpCount] < 0.0)
    aDisturbance[i].m_pData[frameIndex + hulpCount] = 0.0
3209                            if (aAddedDisturbance[i].m_pData[frameIndex + hulpCount] < 0.0)
    aAddedDisturbance[i].m_pData[frameIndex + hulpCount] = 0.0
3210                        }
3211                        for (int jj = 1  jj < jjMax  jj++) aDisturbance[i].m_pData[frameIndex - jj]
    /= pow((freqShiftChanges + 1.0), 0.12)
3212                    }
3213                }
3214            }
3215
3216            for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx  frameIndex++)
    {
3217            }
3218
3219            count = 0
3220            number_of_sections_inserted = 0
3221            number_of_sections_critical = 0
3222            number_of_sections_invalid = 0
3223            bool startedInsertion = false
3224            int indexStartedInsertion = 0
3225            XFLOAT penalizationLevelBeforeAfterInsertion = 0.0
3226            penalizationLevelBeforeAfterInsertionAvg000 = 0.0
3227            for (frameIndex = statics->startFrameIdx  frameIndex < (statics->stopFrameIdx - 1)
    frameIndex++)
3228            {
3229                delayReliabilityPerFrameWeight =
    (pOverviewHolder->m_DelayReliabilityPerFrame[frameIndex] +
    pOverviewHolder->m_DelayReliabilityPerFrame[frameIndex + 1]) / 2.0
3230                if (delayReliabilityPerFrameWeight<0.03) delayReliabilityPerFrameWeight = 0.03
3231                if (delayReliabilityPerFrameWeight>0.9) delayReliabilityPerFrameWeight = 0.9
3232                delayReliabilityPerFrameWeight /= 0.9
3233                delayReliabilityPerFrameWeight = pow(delayReliabilityPerFrameWeight, 0.04)
3234                hulp1 = 1.0
3235                hulp2 = 1.0
3236                if (pMarkSectionFlags[frameIndex] & 4)
3237                {
3238                    number_of_sections_invalid++
3239                }
3240                else if (pMarkSectionFlags[frameIndex] & 2)
3241                {
3242                    number_of_sections_critical++
3243                    hulp2 = pow((1.0*pMarkSectionFlags[frameIndex]) + 0.1, 0.4)
3244                }
3245                else if (pMarkSectionFlags[frameIndex] & 1)
3246                {
3247                    number_of_sections_inserted++
3248                    hulp1 = pow((1.0*pMarkSectionFlags[frameIndex]) + 0.1, 0.3)
3249                }
3250                if (!(pMarkSectionFlags[frameIndex] & 8)) {
3251                    if (startedInsertion == true)
3252                    {
3253                        penalizationLevelBeforeAfterInsertion = 0.0
3254                        if (indexStartedInsertion >(statics->startFrameIdx + 2))
3255                            penalizationLevelBeforeAfterInsertion +=
    2.0*aDistortedLoudness.m_pData[indexStartedInsertion - 1]
3256                        if (frameIndex <(statics->stopFrameIdx - 1))
3257                        if ((frameIndex - indexStartedInsertion) > 10)
    penalizationLevelBeforeAfterInsertion += (aDistortedLoudness.m_pData[frameIndex]
    + aDistortedLoudness.m_pData[frameIndex + 1]) / 14.0
3258                        penalizationLevelBeforeAfterInsertionAvg000 +=
```

```
    penalizationLevelBeforeAfterInsertion
3259                    for (int j = indexStartedInsertion  j < frameIndex  j++){
3260                        for (i = 0  i < NUMBER_OF_POWERS_OVER_FREQ  i++){
3261                            aDisturbance[i].m_pData[j] +=
    0.15*penalizationLevelBeforeAfterInsertion*pow((i*1.0 + 1.0), 0.2)
3262                            aAddedDisturbance[i].m_pData[j] +=
    penalizationLevelBeforeAfterInsertion*pow((i*1.0 + 1.0), 0.2)
3263                        }
3264                    }
3265                    startedInsertion = false
3266                }
3267            }
3268            else {
3269                if (startedInsertion == false)
3270                {
3271                    startedInsertion = true
3272                    indexStartedInsertion = frameIndex
3273                }
3274                for (i = 0  i < NUMBER_OF_POWERS_OVER_FREQ  i++) {
3275                    aDisturbance[i].m_pData[frameIndex] = 17.0*pow((i*1.0 + 1.0), 0.4)
3276                    aAddedDisturbance[i].m_pData[frameIndex] = 0.0
3277                }
3278            }
3279        }
3280        fraction_of_sections_inserted = number_of_sections_inserted / (numberOfSpeechFrames + 0.01)
3281        fraction_of_sections_critical = number_of_sections_critical / (numberOfSpeechFrames + 0.01)
3282        fraction_of_sections_invalid = number_of_sections_invalid / (numberOfSpeechFrames + 0.01)
3283        penalizationLevelBeforeAfterInsertionAvg000 = penalizationLevelBeforeAfterInsertionAvg000 /
    (numberOfSpeechFrames + 0.01)
3284        freqShiftChangesAvg /= (numberOfSpeechFrames + 0.01)
3285
3286        if (statics->nrSpeechFrames > 0)
3287        {
3288            averageScale = (XFLOAT)pow((averageScale / statics->nrSpeechFrames), (XFLOAT) 5e-3)
3289        }
3290        else
3291        {
3292            averageScale = (XFLOAT) 1.0
3293        }
3294
3295        int NumFrames = statics->nrFrames
3296        pOverviewHolder->m_NumberOfFrames = NumFrames
3297        pOverviewHolder->m_aTransformLength = aTransformLength
3298        pOverviewHolder->m_OverlapCoeff = 0.75
3299        pOverviewHolder->m_SampleFrequencyHz = (long)statics->sampleRate
3300        pOverviewHolder->m_NumberOfBands=statics->aNumberOfBarkBands
3301
3302        pOverviewHolder->m_ResultFlags |= (RESF_LEVEL1_AVAILABLE)
3303
3304
3305
3306        if (gBatchMode)
3307        {
3308            s.Format("PSQM command completed succesfully!\n")
3309
3310            return TRUE
3311        }
3312
3313        if (Mode & RESF_LEVEL3_AVAILABLE)
3314        {
3315            pOverviewHolder->m_pDisturbance = (double*)matMalloc(NumFrames * sizeof(double))
3316            pOverviewHolder->m_pAddedDisturbance = (double*)matMalloc(NumFrames * sizeof(double))
3317            pOverviewHolder->m_pDistortedLoudness = (double*)matMalloc(NumFrames * sizeof(double))
3318            pOverviewHolder->m_pOriginalLoudness = (double*)matMalloc(NumFrames * sizeof(double))
3319            pOverviewHolder->m_pTime = (double*)matMalloc(NumFrames * sizeof(double))
3320
3321            pOverviewHolder->m_StartFrameIndex = statics->startFrameIdx
3322            pOverviewHolder->m_StopFrameIndex = statics->stopFrameIdx
```

```
3323            for (frameIndex = statics->startFrameIdx  frameIndex <= statics->stopFrameIdx
    frameIndex++)
3324            {
3325                pOverviewHolder->m_pDisturbance[frameIndex] = aDisturbance[(3 - MINIMUM_POWER_FREQ)
    / STEP_POWER_FREQ].m_pData[frameIndex]
3326                pOverviewHolder->m_pAddedDisturbance[frameIndex] =
    aAddedDisturbance[0].m_pData[frameIndex]
3327                pOverviewHolder->m_pDistortedLoudness[frameIndex] =
    aDistortedLoudness.m_pData[frameIndex]
3328                pOverviewHolder->m_pOriginalLoudness[frameIndex] =
    aOriginalLoudness.m_pData[frameIndex]
3329                pOverviewHolder->m_pTime[frameIndex] = frameIndex * (aTransformLength *
    OverlapCoeff) / statics->sampleRate
3330            }
3331
3332            CreateArrayFromCSignal(&pOverviewHolder->m_pOriginalHzPowerSpectrum,
    &originalHzPowerSpectrum)
3333            CreateArrayFromCSignal(&pOverviewHolder->m_pDistortedHzPowerSpectrum,
    &distortedHzPowerSpectrum)
3334            CreateArrayFromCSignal(&pOverviewHolder->m_pOriginalPitchPowerDensity,
    &originalPitchPowerDensity)
3335            CreateArrayFromCSignal(&pOverviewHolder->m_pDistortedPitchPowerDensity,
    &distortedPitchPowerDensity)
3336            CreateArrayFromCSignal(&pOverviewHolder->m_pOriginalLoudnessDensity,
    &originalLoudnessDensity)
3337            CreateArrayFromCSignal(&pOverviewHolder->m_pDistortedLoudnessDensity,
    &distortedLoudnessDensity)
3338            CreateArrayFromCSignal(&pOverviewHolder->m_pDisturbanceDensity, &disturbanceDensity)
3339
3340            pOverviewHolder->m_ResultFlags |= RESF_LEVEL3_AVAILABLE
3341        }
3342        else
3343        {
3344            pOverviewHolder->m_StartFrameIndex = 0
3345            pOverviewHolder->m_StopFrameIndex=0
3346
3347            pOverviewHolder->m_pDisturbance = 0
3348            pOverviewHolder->m_pAddedDisturbance = 0
3349            pOverviewHolder->m_pDistortedLoudness = 0
3350            pOverviewHolder->m_pOriginalLoudness = 0
3351            pOverviewHolder->m_pTime = 0
3352
3353            pOverviewHolder->m_SizeofAlignedOriginaTimeSeries = 0
3354            pOverviewHolder->m_AlignedOriginalTimeSeries = 0
3355
3356            pOverviewHolder->m_SizeofAlignedDistortedTimeSeries = 0
3357            pOverviewHolder->m_AlignedDistortedTimeSeries = 0
3358
3359        }
3360
3361        if (UseThisFrame) delete[] UseThisFrame
3362        if (FrameFlags) delete[] FrameFlags
3363        if (bestSpectrumShift)
3364            matFree(bestSpectrumShift)
3365        if (bestWarpingFacPerFrame)
3366            matFree(bestWarpingFacPerFrame)
3367
3368        return TRUE
3369    }
3370 }
3371
3372
```