srfft.h

Go to the documentation of this file.

```
// matrix/srfft.h

// Copyright 2009-2011  Microsoft Corporation;  Go Vivace Inc.
//                2014  Daniel Povey
//
// See ../../COPYING for clarification regarding multiple authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//  http://www.apache.org/licenses/LICENSE-2.0
//
// THIS CODE IS PROVIDED *AS IS* BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
// KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED
// WARRANTIES OR CONDITIONS OF TITLE, FITNESS FOR A PARTICULAR PURPOSE,
// MERCHANTABLITY OR NON-INFRINGEMENT.
// See the Apache 2 License for the specific language governing permissions and
// limitations under the License.
//
// This file includes a modified version of code originally published in Malvar,
// H., "Signal processing with lapped transforms, " Artech House, Inc., 1992.  The
// current copyright holder of the original code, Henrique S. Malvar, has given
// his permission for the release of this modified version under the Apache
// License v2.0.

#ifndef KALDI_MATRIX_SRFFT_H_
#define KALDI_MATRIX_SRFFT_H_

#include "matrix/kaldi-vector.h"
#include "matrix/kaldi-matrix.h"

namespace kaldi {



// This class is based on code by Henrique (Rico) Malvar, from his book
// "Signal Processing with Lapped Transforms" (1992).  Copied with
// permission, optimized by Go Vivace Inc., and converted into C++ by
// Microsoft Corporation
// This is a more efficient way of doing the complex FFT than ComplexFft
// (declared in matrix-functios.h), but it only works for powers of 2.
// Note: in multi-threaded code, you would need to have one of these objects per
// thread, because multiple calls to Compute in parallel would not work.
template<typename Real>
class SplitRadixComplexFft {
 public:
  typedef MatrixIndexT Integer;

  // N is the number of complex points (must be a power of two, or this
  // will crash).  Note that the constructor does some work so it's best to
  // initialize the object once and do the computation many times.
  SplitRadixComplexFft(Integer N);

  // Does the FFT computation, given pointers to the real and
  // imaginary parts.  If "forward", do the forward FFT; else
  // do the inverse FFT (without the 1/N factor).
  // xr and xi are pointers to zero-based arrays of size N,
  // containing the real and imaginary parts
  // respectively.
  void Compute(Real *xr, Real *xi, bool forward) const;

  // This version of Compute takes a single array of size N*2,
  // containing [ r0 im0 r1 im1 ... ].  Otherwise its behavior is  the
  // same as the version above.
  void Compute(Real *x, bool forward);


  // This version of Compute is const; it operates on an array of size N*2
  // containing [ r0 im0 r1 im1 ... ], but it uses the argument "temp_buffer" as
  // temporary storage instead of a class-member variable.  It will allocate it if
  // needed.
  void Compute(Real *x, bool forward, std::vector<Real> *temp_buffer) const;

  ~SplitRadixComplexFft();

 protected:
  // temp_buffer_ is allocated only if someone calls Compute with only one Real*
  // argument and we need a temporary buffer while creating interleaved data.
  std::vector<Real> temp_buffer_;
 private:
  void ComputeTables();
  void ComputeRecursive(Real *xr, Real *xi, Integer logn) const;
  void BitReversePermute(Real *x, Integer logn) const;

  Integer N_;
  Integer logn_;  // log(N)

  Integer *brseed_;
  // brseed is Evans' seed table, ref:  (Ref: D. M. W.
  // Evans, "An improved digit-reversal permutation algorithm ...",
  // IEEE Trans. ASSP, Aug. 1987, pp. 1120-1125).
  Real **tab_;       // Tables of butterfly coefficients.

  KALDI_DISALLOW_COPY_AND_ASSIGN(SplitRadixComplexFft);
};

template<typename Real>
class SplitRadixRealFft: private SplitRadixComplexFft<Real> {
 public:
  SplitRadixRealFft(MatrixIndexT N):  // will fail unless N>=4 and N is a power of 2.
      SplitRadixComplexFft<Real> (N/2), N_(N) { }

  void Compute(Real *x, bool forward);


  void Compute(Real *x, bool forward, std::vector<Real> *temp_buffer) const;

 private:
  KALDI_DISALLOW_COPY_AND_ASSIGN(SplitRadixRealFft);
  int N_;
};



} // end namespace kaldi


#endif

```