NumCSE

Autumn Semester 2017

Prof. Rima Alaifari

Exercise sheet 6
Convolution and fast Fourier transform

A. Dabrowski

Problem 6.1: Convolution and FFT

We examine a computationally useful link between discrete and circular convolution via the fast fourier transform. We apply it to calculate the filtering of a noisy signal.

Template: `conv.cpp`

Let $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^n$ be two vectors with $n < m$. Implement `C++` functions which take as arguments Eigen vectors `x` and `y` and return:

(a) the discrete convolution of `x` and `y` (implement the definition of convolution);

SOLUTION:

```cpp
VectorXd PointA(const VectorXd &x, const VectorXd &y) {
    // returns discrete convolution of x and y (straightforward
       implementation)
    int m = x.size();
    int n = y.size();
    int L = m+n-1;
    VectorXd out = VectorXd::Zero(L);

    for (int i=0; i<L; i++) {
        for (int k=std::max(0,i+1-m); k<std::min(i+1,n); k++) {
            out(i) += x(i-k) * y(k);
        }
    }

    return out;
}
```

(b) the vector given by `ifft(fft(x)fft(ym))`, where `ym` has been obtained by appending $n - m$ zeros to `y`;

SOLUTION:

```cpp
VectorXd PointB(const VectorXd &x, const VectorXd &y) {
    // returns an overlapping convolution of x and y
    int m = x.size();
    int n = y.size();
    VectorXd ym = VectorXd::Zero(m);
    ym.head(n) = y;

    FFT<double> fft;
    VectorXcd a = fft.fwd(x);
    VectorXcd b = fft.fwd(ym);
    return fft.inv(a.cwiseProduct(b).eval()).real();
}
```

(c) the vector given by `ifft(fft(xL)fft(yL))`, where `xL` and `yL` have been obtained by padding with zeros `x` and `y` to length $L = m + n - 1$.

SOLUTION:

```cpp
VectorXd PointC(const VectorXd &x, const VectorXd &y) {
    // returns discrete convolution of x and y (fft implementation)
    int m = x.size();
    int n = y.size();
    int L = m+n-1;
    VectorXd yL = VectorXd::Zero(L);
    VectorXd xL = VectorXd::Zero(L);
    xL.head(m) = x;
    yL.head(n) = y;

    FFT<double> fft;
    VectorXcd a = fft.fwd(xL);
    VectorXcd b = fft.fwd(yL);
    return fft.inv(a.cwiseProduct(b).eval()).real();
}
```

(d) Will the result of Point a coincide/differ with the result from Point b or Point c? Why?

SOLUTION:

Point a and c will coincide because of the convolution theorem. In general Point b will give a different vector at the beginning and at the end because of the sum of overlapping parts.

(e) Test your code for `x` an input vector given by a constant signal with some random uniform noise and `y` a Gaussian truncated filter (see the template). Does the application of the filter reduce the noise of the signal?

SOLUTION:

If we assume that the peaks are produced by noise we are reducing noise and improving the signal. If the peaks were not the result of noise however, we would be losing information/resolution.

Problem 6.2: 2D convolution, FFT2 and Laplace filter

We review two dimensional convolution, its relation with the discrete Fourier transform and we implement a discrete Laplacian filter.

Template: `conv2.cpp`

Consider the 2-dimensional infinite arrays $X = (X_{k_1,k_2})_{k_1,k_2 \in \mathbb{Z}}$ and $Y = (Y_{k_1,k_2})_{k_1,k_2 \in \mathbb{Z}}$. We can define their convolution as the 2-dimensional infinite array $X * Y$ such that

$$(X * Y)_{k_1,k_2} = \sum_{j_1=-\infty}^{\infty} \sum_{j_2=-\infty}^{\infty} X_{j_1,j_2} Y_{k_1-j_1,k_2-j_2}.$$

In the same way as in the 1-dimensional case, given two matrices $X \in \mathbb{R}^{m_1,m_2}$ and $Y \in \mathbb{R}^{n_1,n_2}$ we can define their *discrete convolution* as the convolution between the zero extensions of X and Y trimmed of unnecessary zeros, and their *circular convolution* as the smallest period of the convolution between the periodic extension of X and the zero extension of Y.

Consider two matrices $A \in \mathbb{R}^{n,m}$ and $F \in \mathbb{R}^{k,k}$ with $k < n, m$.

(a) How should we extend A and F to larger matrices $\tilde{A}$ and $\tilde{F}$ so that the discrete convolution of A with F is equal to the circular convolution of $\tilde{A}$ with $\tilde{F}$?

SOLUTION:

By padding A and F with zeros so that the dimensions of $\tilde{A}$ and $\tilde{F}$ are $L_r \times L_c$ where $L_r =$(number of rows of A) $+$ (number of rows of F) $- 1$ and $L_c =$ (number of columns of A) $+$ (number of columns of F) $- 1$.

(b) By recalling to the 1-dimensional fast fourier transform, implement a `C++` function `fft2` which returns the 2-dimensional fast fourier transform of a matrix.

SOLUTION:

```
template <typename Scalar>
void fft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
    // see code 4.2.48 of lecture notes
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  C.resize(m,n);
  Eigen::MatrixXcd tmp(m,n);

  Eigen::FFT<double> fft; // Helper class for DFT
  // Transform rows of matrix Y
  for (idx_t k=0;k<m;k++) {
    Eigen::VectorXcd tv(Y.row(k));
    tmp.row(k) = fft.fwd(tv).transpose();
  }

  // Transform columns of temporary matrix
  for (idx_t k=0;k<n;k++) {
    Eigen::VectorXcd tv(tmp.col(k));
```

```cpp
    C.col(k) = fft.fwd(tv);
  }
}

template <typename Scalar>
void ifft2(Eigen::MatrixXcd &C, const Eigen::MatrixBase<Scalar> &Y) {
    // see code 4.2.49 of lecture notes
  using idx_t = Eigen::MatrixXcd::Index;
  const idx_t m = Y.rows(),n=Y.cols();
  fft2(C,Y.conjugate());
  C = C.conjugate()/(m*n);
}
```

(c) Implement an efficient `C++` function with arguments A and F which implements their discrete convolution.

Hint: use the result derived in the previous steps and the two dimensional circular convolution theorem.

SOLUTION:

```cpp
void conv2(MatrixXcd &C, const MatrixXcd &A1, const MatrixXcd &A2) {
    // returns discrete convolution between A1 and A2 (fft implementation)
    int n1 = A1.rows();
    int m1 = A1.cols();
    int n2 = A2.rows();
    int m2 = A2.cols();
    int Lrow = n1+n2-1;
    int Lcol = m1+m2-1;

    MatrixXcd A1_ext = MatrixXcd::Zero(Lrow, Lcol);
    MatrixXcd A2_ext = MatrixXcd::Zero(Lrow, Lcol);
    MatrixXcd tmp1 = MatrixXcd::Zero(Lrow, Lcol);
    MatrixXcd tmp2 = MatrixXcd::Zero(Lrow, Lcol);
    A1_ext.topLeftCorner(n1,m1) = A1;
    A2_ext.topLeftCorner(n2,m2) = A2;

    fft2(tmp1, A1_ext);
    fft2(tmp2, A2_ext);
    ifft2(C, tmp1.cwiseProduct(tmp2));
```

The discrete Laplacian filter is defined as

$$F = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}.$$

It is often used to detect edges in images.

(d) Test your filter on the black and white "picture" provided in the template.