Algorithms of Scientific Computing

Discrete Cosine Transformation – Solution

Excercise 1: Discrete Cosine Transform

Algorithm for the Cosine Transform

The procedure \( \text{FFT}(f,N) \) computes the correct coefficients, if we pass the \( N + 1 \) data from field \( g \) as a dataset of length \( 2N \) with symmetry \( f_{-n} = f_n \).

From equation (1) of the worksheet we know that \( \text{FFT}(f,N) \) gets a dataset \( f \) with indices \( n = -N + 1, \ldots, N \). We only have to compute the \( F_k \) for \( k = 0, \ldots, N \).

So, the algorithm looks like this:

1. For all \( n = 0, \ldots, N \):
   - Set \( f[n] := g[n] = f_n \)
   - Set \( f[-n] := g[n] = f_n \)
2. Call \( \text{FFT}(f,N) \)
3. (Now the Fourier coefficients \( F_k \) are stored in the field \( f \))
   - For all \( k = 0, \ldots, N \):
     - Set \( g[k] := f[k] = F_k \)

Excercise 2: Fast Discrete Cosine Transform

The butterfly scheme is retrieved as usual:

\[
F_k = \frac{1}{2N} \sum_{n=-N+1}^{N} f_n \omega_{2N}^{-kn} = \frac{1}{2} \left( \frac{1}{N} \sum_{n=-N/2+1}^{N/2} f_{2n} \omega_{2N}^{-2kn} + \frac{1}{N} \sum_{n=-N/2+1}^{N/2} f_{2n-1} \omega_{2N}^{-k(2n-1)} \right)
\]

\[
= \frac{1}{2} \left( \frac{1}{N} \sum_{n=-N/2+1}^{N/2} f_{2n} \omega_N^{-kn} + \frac{1}{N} \sum_{n=-N/2+1}^{N/2} f_{2n-1} \omega_N^{-kn} \omega_{2N}^{k} \right)
\]

\[
= \frac{1}{2} \left( G_k + \omega_{2N}^{k} H_k \right)
\]
\[ F_{k+N} = \frac{1}{2} \left( G_{k+N} + \omega_{2N}^{k+N} H_{k+N} \right) = \frac{1}{2} \left( G_k - \omega_{2N}^k H_k \right) \]

For the datasets \( g_n := f_{2n} \) and \( h_n := f_{2n-1} \), respectively, we can try to find other symmetries:

\[ g_{-n} = f_2(-n) = f_{-2n} = f_{2n} = g_n \]

The “even” data also shows an even symmetry and therefore lead to another Cosine Transform but with half length.

Analog for the data with odd indices:

\[ h_{-n} = f_2(-n-1) = f_{-2n-1} = f_{2n+1} = f_{2(n+1)-1} = h_{n+1} \]

Again we get an “even” symmetry. However, this is the transform shown in the lecture, known as Quarter-Wave-DCT, again with half length.

For a dataset with the symmetry constraint \( f_{-n} = f_{n+1} \) we get accordingly

\[ g_{-n} = f_2(-n) = f_{-2n} = f_{2n+1} = h_{n+1} \]

and

\[ h_{-n} = f_{-2n-1} = f_{-2n+1} = f_{2n+2} = f_{2n-1} = g_{n+1} \]

### Fast Poisson Solver – Solution

#### Derivation of the System of Linear Equations – For the Sake of Completeness

In the lecture we derived the system of equations from a discrete model. However, the same systems of equations show up during the numerical solution of (partial) differential equations.

The examined one-dimensional Heat Transfer Problem is modelled by the so called Poisson equation with appropriate boundary conditions, this looks like shown here:

\[
-\frac{\partial^2}{\partial x^2} u(x) = f(x) \quad \text{for} \quad x \in (0, 1) \\
u(0) = u(1) = 0
\]

(1)

For a numerical solution we look for a solution \( u(x) \) on the discrete points \( x_n := nh \), with \( n = 0, \ldots, N \) and \( h := \frac{1}{N} \). So, in the equation (1) we substitute the partial derivative by an adequate difference quotient and get the following approximation on the points \( x_n \):

\[
-\frac{u(x_{n+1}) - 2u(x_n) + u(x_{n-1})}{h^2} \approx f(x_n) \quad \text{for} \quad n = 1, \ldots, N - 1 \\
u(x_0) = u(0) = 0 \\
u(x_N) = u(1) = 0
\]

(2)
With \( f_n := h^2 f(x_n) \) we get a system of linear equations for computing the approximation \( u_n \approx u(x_n) \):

\[
-u_{n+1} + 2u_n - u_{n-1} = f_n \quad \text{for} \quad n = 1, \ldots, N - 1
\]

\[ u_0 = u_N = 0 \tag{3} \]

In the two-dimensional case the Poisson equation derives to

\[
-\frac{\partial^2}{\partial x^2} u(x, y) - \frac{\partial^2}{\partial y^2} u(x, y) = f(x, y)
\]

\[ u(x, y) = 0 \quad \text{if} \quad x \in \{0, 1\} \quad \text{oder} \quad y \in \{0, 1\}. \tag{4} \]

We are now looking for the approximations \( u_{n,m} \approx u(x_n, y_m) \), where \( x_n := nh \) and \( y_m := mh \) for \( n, m = 0, \ldots, N \). So, we use a regular Grid with points \( x_n, y_m \), where we use the same number of points in x- and y-direction. The partial derivation in equation (4) we approximate analog to the one-dimensional case by means of the difference quotients

\[
\frac{\partial^2}{\partial x^2} u(x_n, y_m) \approx \frac{u_{n+1,m} - 2u_{n,m} + u_{n-1,m}}{h^2} \quad \text{and} \quad \frac{\partial^2}{\partial y^2} u(x_n, y_m) \approx \frac{u_{n,m+1} - 2u_{n,m} + u_{n,m-1}}{h^2}.
\]

If we set \( f_{nm} := h^2 f(x_n, y_m) \), we get the following system of linear equations:

\[
-u_{n,m+1} - u_{n+1,m} + 4u_{n,m} - u_{n-1,m} - u_{n,m-1} = f_{nm} \quad \text{for} \quad n, m = 1, \ldots, N - 1
\]

\[ u_{0,m} = u_{n,0} = 0 \quad \text{for} \quad n, m = 0, \ldots, N \tag{6} \]

**Exercise 3: Two-Dimensional Fast Poisson Solver**

We insert the transformations

\[
u_{nm} = 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} \quad \text{and} \quad f_{nm} = 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} F_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} \tag{7}\]

into the system of equations

\[
-u_{n,m+1} - u_{n+1,m} + 4u_{n,m} - u_{n-1,m} - u_{n,m-1} = f_{n,m} \quad \text{for} \quad n, m = 1, \ldots, N - 1 \tag{8}\]

and get

\[
\begin{align*}
- & \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi(n+1)k}{N} \sin \frac{\pi ml}{N} - \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi(n-1)k}{N} \sin \frac{\pi ml}{N} \\
- & \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi(m+1)l}{N} - \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi(m-1)l}{N} \\
+ & 4 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} = \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} F_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N}
\end{align*}
\]
As in the one-dimensional case we can convert as follows:

\[
\sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi(n+1)k}{N} \sin \frac{\pi ml}{N} + \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi(n-1)k}{N} \sin \frac{\pi ml}{N} = 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \cos \frac{\pi k}{N} \sin \frac{\pi ml}{N},
\]

and

\[
\sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi(m+1)l}{N} + \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi(m-1)l}{N} = 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} \cos \frac{\pi l}{N}.
\]

So, we get

\[
- 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} \cos \frac{\pi k}{N} + 2 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} \cos \frac{\pi l}{N} + 4 \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} U_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N} = \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} F_{kl} \sin \frac{\pi nk}{N} \sin \frac{\pi ml}{N}
\]

This holds, if

\[
U_{kl} \left(4 - 2 \cos \frac{\pi k}{N} - 2 \cos \frac{\pi l}{N}\right) = F_{kl},
\]

holds for all \(k, l = 1, \ldots, N - 1\).

From this we get the dependency

\[
U_{kl} = \frac{F_{kl}}{4 - 2 \cos \frac{\pi k}{N} - 2 \cos \frac{\pi l}{N}}.
\]  

(9)

Algorithm:

The two-dimensional algorithms is virtually identically to the one-dimensional:

1. Compute the coefficients \(F_{k,l}\) by means of a (two-dimensional) Fast Sine Transform.
2. Compute the coefficients \(U_{k,l}\) as shown in equation (9) for all \(k, l = 1, \ldots, N - 1\).
3. Compute the unknown \(u_{n,m}\) out of the coefficients \(U_{k,l}\) with a (two-dimensional) Inverse, Fast Sine Transform.
Both of the Sine Transforms take $O(N^2 \log N)$ operations each, while step 2 needs only $O(N^2)$ operations. In total the system of equations can be solved by this algorithm in $O(N^2 \log N)$ operations.

Some observations and remarks:

- The 2d system of equations can not be written in a form, so that the associated matrix is narrow band matrix (or even a tri-diagonal matrix). So, it cannot be solved directly in $O(N^2)$ operations.
  
  - Usually the system of equations is written, so that the associated matrix is a band matrix of width $N$. So, a direct solver (Gauß elimination) takes $O(N^4)$ operations.
  
  - The nested dissection gives a matrix, which can be solved by a Gauß elimination with $O(N^3)$ operations.

So, both methods have a worse complexity than the algorithm based on the Sine Transform.

- The derivation of the 2d case shows that this method can be transferred easily to the 3d case and higher dimensional cases. The complexity is $O(N^d \log N)$ in each case.