Computing Refinable Integrals — Documentation of the Program — — Version 1.1 —

Angela Kunoth*

May 26, 1995

Abstract

Based on the theory developed in [DM3], a C++ program is described which computes function values and derivatives of refinable functions and integrals of products of refinable functions (called refinable integrals) for up to four factors in the integral in one and two dimensions. In the three-dimensional case, integrals with at most three factors can be computed. The routines calculate the desired values exactly up to round-off while avoiding any quadrature rules by using the refinement equations for the computations. As input data, only the mask of the refinable functions and some parameters like the dimension and the number of refinable functions are needed.

Key words: C++–program, computing refinable integrals, evaluating multivariate functions, refinable shift-invariant spaces.

1 Introduction

The problem of quickly computing *refinable integrals*, that is, integrals of products of (derivatives of) functions, arises in various fields, not only when one wants to generate linear systems derived from Wavelet–Petrov–Galerkin schemes for finding the approximate solution of an elliptic boundary value problem. In this case, one usually has to compute integrals involving partial and normal derivatives induced by the underlying partial differential operator, and L_2–inner products for the right hand side. For instance, for a problem with variable coefficients, one is lead to compute quantities of the form

$$\int_{\mathbb{R}^d} \chi_{[0,1)^d}\, c_{\eta\beta}(x)\ (D^\eta \phi^1)(2^m x - \alpha^1)\ (D^\beta \phi^2)(2^m x - \alpha^2)\, dx\ , \quad \eta, \beta, \alpha^1, \alpha^2 \in \mathbb{Z}^d\ , \tag{1.1}$$

*Institute for Scientific Computation, Texas A&M University, College Station, Texas 77843–3404, U.S.A., E-mail: kunoth@isc.tamu.edu.

where ϕ^1, ϕ^2 are some trial functions each of which satisfies a *refinement equation*

$$\phi^i(x) = \sum_{\alpha\in\mathbb{Z}^d} a^i_\alpha\, \phi^i(2x-\alpha)\ , \qquad x\in\mathbb{R}^d, \qquad i=1,2\ . \tag{1.2}$$

Usually one chooses these functions to have compact support such that their *masks* $\mathbf{a}^i := \{a^i_\alpha : \alpha \in \mathbb{Z}^d\}$ are also *compactly supported*, i.e. only a finite number of the coefficients do not vanish. To approximate the non-constant coefficients, one could choose a possibly different refinable function $\bar\phi$ along with some local approximation scheme

$$(B_{\bar\ell} f)(x) := \sum_{\alpha\in\mathbb{Z}^d} F_{\bar\ell,\alpha}(f)\, \bar\phi(2^{\bar\ell}x-\alpha)\ , \tag{1.3}$$

where the $F_{\bar\ell,\alpha}$ are suitable functionals supported in a small neighborhood of $2^{-\bar\ell}\alpha$, and replace $c_{\eta\beta}(x)$ in (1.1) by $(B_{\bar\ell}\, c_{\eta\beta})(x)$. Such approximation schemes are well understood when $\bar\phi$ is for instance a tensor product B–spline or a box spline, see [BH, DM1, DM2]. Here $\bar\ell$ has to be chosen so that the overall accuracy is not decreased by the resulting 'quadrature error'. In particular, one can take a characteristic function and can thus approximate *non-smooth* coefficients locally. By a reduction process described in [DM3], the integrals (1.1) employing (1.3) can be transformed into quantities like (1.4) below.

The construction of wavelet-type functions often requires the computation of L_2–inner products as well. Or, since convolutions of box splines on a regular grid are again box splines, one can use the calculation of the integrals as a method for evaluating box splines rather than employing a recurrence formula, see [K2].

In general, the program can be used to determine any type of refinable integrals

$$\int_{\mathbb{R}^d} \varphi^0(x)\ \prod_{i=1}^{3}(D^{\mu^i}\varphi^i)(x-\alpha^i)\,dx\ , \qquad \alpha^i\in\mathbb{Z}^d\ , \tag{1.4}$$

quickly and exactly up to round-off in up to three dimensions where each φ^i is assumed to be refinable with compact support. Note that with four possible factors in the integral, one can compute any type of quantities from the applications mentioned here. In the three-dimensional case, the total number of factors is currently restricted by three since the actual version of the program would not be able to allocate enough storage needed for the computations.

By applying the refinement equation for the integral (1.4) corresponding to (1.2) which is given in (2.8) below, one can further obtain the terms (1.4) for all dyadic points $\alpha = \frac{\beta}{2^p}$, $\beta\in\mathbb{Z}^d$, $p\in\mathbb{Z}$, fast.

As for the theory, there have been some attempts in the univariate case to compute inner products of refinable functions or wavelets where only one of the factors is allowed to have derivatives, see [Be, LRT] (called 'connection coefficients' in the latter report). Using the refinement equations (1.2), the computation of such quantities is usually reduced to the solution of an eigenvector problem. However, in the case of more than one factor having derivatives in the univariate case or even multivariate functions and integrals, the solution of the eigenvector problem is no longer unique. Based on asymptotic expansions of the corresponding stationary subdivision schemes and on determining eigenvectors for compressed subdivision schemes, it is shown in [DM3] how uniqueness can be guaranteed for integrals of products of several refinable multivariate functions with an arbitrary number of derivatives in the integrals. The method is based on function evaluations of a refinable multivariate function at integers and can therefore be used as well for the fast evaluation of (derivatives of) multivariate refinable functions.

This paper is structured as follows. In Section 2, we briefly review the main results from [DM3]. Section 3 is devoted to a description of the C++–program that performs the computation. In Section 4, we show how to apply the program, and Section 5 finally provides several examples. (More examples and data can be found in [K2].)

Access: The source code in C++ and the example data from Section 5 can be obtained via anonymous ftp as follows:

```
>ftp ftp.igpm.rwth-aachen.de
Name: anonymous
Guest login ok, send ident as password.
Password:
ftp> cd programs/integrals
ftp> prompt
ftp> mget *
ftp> quit
```

Please notify the author by sending an e-mail to

`kunoth@isc.tamu.edu`

when you have got the programs and to report any errors or comments. In that way you will be informed of updated versions and extensions of the program.

The program runs under the operation system UNIX on SGI, HP, SUN Workstations with e.g. a CC or g++ compiler. The collection of files contain one which is called `read.me`. It briefly states how to obtain the executable file and how to run the program, see also Section 4. All computations are performed in double precision.

This report is an update of the first version [K3]. The main difference is that the format of all output data was changed to floating point representation.

2 Theoretical Results

To briefly review the main results from [DM3] needed for the understanding of the performance of the program, let φ^i, $i = 0, \dots, \ell$, be compactly supported continuous functions on $I\!R^d$ where each satisfies a refinement equation of the form

$$\varphi^i(x) = \sum_{\alpha \in \mathbb{Z}^d} a^i_\alpha \, \varphi^i(2x - \alpha) \, , \qquad x \in I\!R^d \, , \tag{2.1}$$

with some finitely supported masks $\mathbf{a}^i := \{a^i_\alpha : \alpha \in \mathbb{Z}^d\}$. (At the end of this section, we recall how to compute these mask coefficients for B– and box splines.) In addition, we assume that φ^i is *stable* for each $i = 0, \dots, \ell$, i.e.

$$\| \sum_{\alpha \in \mathbb{Z}^d} \lambda_\alpha \, \varphi^i(\cdot - \alpha) \|_{L_2(\mathbb{R}^d)} \sim \left(\sum_{\alpha \in \mathbb{Z}^d} |\lambda_\alpha|^2 \right)^{1/2} \tag{2.2}$$

for any $\lambda \in \ell_2(\mathbb{Z}^d)$.

We are interested in evaluating the terms

$$H(\alpha) = H(\alpha^1, \dots, \alpha^\ell) := \int_{\mathbb{R}^d} \varphi^0(x) \prod_{i=1}^{\ell} (D^{\mu^i} \varphi^i)(x - \alpha^i) \, dx \tag{2.3}$$

on $\mathbb{Z}^s$, $s = \ell d$. (The implementation described in this paper covers $\ell = 3$ for $d = 1, 2$ and $\ell = 2$ for $d = 3$.) Since each φ^i satisfies a refinement equation (2.1), the function

$$F(\alpha) := \int_{\mathbb{R}^d} \varphi^0(x) \prod_{i=1}^{\ell} \varphi^i(x - \alpha^i) \, dx \tag{2.4}$$

defined on $\mathbb{R}^s$, does too,

$$F(x) = \sum_{\mu \in \mathbb{Z}^s} c_\mu \, F(2x - \mu) \, , \qquad x \in \mathbb{R}^s \, , \tag{2.5}$$

with mask coefficients

$$c_\mu := 2^{-d} \sum_{\nu \in \mathbb{Z}^d} a^0_\nu \prod_{i=1}^{\ell} a^i_{\nu - \mu^i} \; . \tag{2.6}$$

Thus, the computation of terms (2.3) on all dyadic points reduces to evaluate H at lattice points. All other values of H can be obtained from these quantities by inserting the refinement equation for H,

$$2^{-|\mu|} H(x) = \sum_{\alpha \in \mathbb{Z}^s} c_\alpha \, H(2x - \alpha) \, , \qquad x \in \mathbb{R}^s \, , \tag{2.7}$$

where $|\mu| = |\mu^1| + \cdots + |\mu^\ell|$ is the total number of derivatives in (2.3).

By a change of indices, this equation can be rewritten as

$$2^{-|\mu|} H(\alpha) = \sum_{\nu \in \mathbb{Z}^s} c_{2\alpha - \nu} H(\nu) \, , \qquad \alpha \in \mathbb{Z}^s \, . \tag{2.8}$$

Since all masks involved have finite support, equation (2.8) can be interpreted as the problem of computing an *eigenvector*. Thus, one needs conditions that determine among possibly several eigenvectors the correct eigenvector *uniquely*.

One possibility involving appropriate factorizations of the *symbols*

$$a^i(z) := \sum_{\alpha \in \mathbb{Z}^d} a^i_\alpha \, z^\alpha$$

corresponding to the masks $\mathbf{a}^i$, $i = 1, \ldots, \ell$, according to partial derivatives in the integrals is described in [DM3] which is extended to directional derivatives in [K1] (see also [K2]). In this case, only the highest eigenvalue of the system (2.8) has to be computed which can be done iteratively by using the power method for large systems. This will be implemented in a forthcoming version of the program.

To state the main result, let us recall that for a mask $\mathbf{a}$ on $\mathbb{Z}^s$ of finite support, the *stationary subdivision scheme*

$$(S_{\mathbf{a}}\lambda)_\alpha := \sum_{\beta \in \mathbb{Z}^s} a_{\alpha-2\beta}\lambda_\beta \ , \qquad \alpha \in \mathbb{Z}^s \ , \tag{2.9}$$

is said to *converge* if for any $\lambda \in \ell_\infty(\mathbb{Z}^s)$ there is some $f_\lambda \in \mathcal{C}(\mathbb{R}^s)$ such that

$$\lim_{k\to\infty} \left(\sup_{\alpha \in \mathbb{Z}^s} \left| (S^k_{\mathbf{a}}\lambda)_\alpha - f_\lambda \left(\frac{\alpha}{2^k} \right) \right| \right) = 0 \ . \tag{2.10}$$

It was shown in [CDM] Theorem 2.1, p. 14, that a convergent $S_{\mathbf{a}}$ implies the existence of a unique $\varphi \in \mathcal{C}_0(\mathbb{R}^s)$ which is refinable with respect to $\mathbf{a}$. Conversely, $\mathbf{a}$–refinability and ℓ_∞–stability of $\varphi \in \mathcal{C}_0(\mathbb{R}^s)$ imply convergence in the sense of (2.10), see [CDM] Proposition 2.3, p. 23.

Based on asymptotic expansions of the stationary subdivision operator related to the mask (2.6), it is shown in [DM3] how in the multivariate case and with derivatives in the integrals (2.3) additional moment conditions guarantee uniqueness of the eigenvector.

Theorem 2.1 *Let $\varphi^j \in \mathcal{C}^m_0(\mathbb{R}^d)$ be $\mathbf{a}^j$–refinable and ℓ_∞–stable for each $j = 0, \ldots, \ell$. Then, for every $\mu \in \mathbb{Z}^s_+$, $|\mu| \le m$, there exists a unique finitely supported sequence $\{W_\beta : \beta \in \mathbb{Z}^s\}$ satisfying*

$$\begin{aligned} \sum_{\beta \in \mathbb{Z}^s} c_{2\alpha-\beta} W_\beta &= 2^{-|\mu|} W_\alpha \ , \qquad \alpha \in \mathbb{Z}^s \ , \\ \sum_{\alpha \in \mathbb{Z}^s} (-\alpha)^\nu \, W_\alpha &= \mu! \, \delta_{\nu\mu} \ , \qquad |\nu| \le |\mu| \ , \quad \nu, \mu \in \mathbb{Z}^s_+ \ , \end{aligned} \tag{2.11}$$

where $\mathbf{c}$ is the mask given in (2.6) for the function F defined by (2.4). Moreover, one has

$$H(\alpha) = (-1)^{|\mu|} W_\alpha \ , \qquad \alpha \in \mathbb{Z}^s \ . \tag{2.12}$$

Remark 2.2 *This theorem actually holds for far fewer smoothness assumptions. For instance, practically one can take φ^0 as characteristic function and φ^i to be μ^i times continuously differentiable, $i = 1, \ldots, \ell$, [D] when one wants to compute integrals with μ^i as in (2.3).*

Remark 2.3 *If one wishes to evaluate any multivariate function F satisfying a refinement equation (2.5) with mask coefficients $\mathbf{c}$ or compute its derivatives at lattice points, the solution W_α of (2.11) already gives the wanted quantities.*

Mask coefficients for B-splines and box splines
For univariate cardinal B–Splines of *order* k, N_k, the mask coefficients are computed as

$$N_k(x) = 2^{1-k} \sum_{j=0}^{k} \binom{k}{j} N_k(2x - j) \ , \qquad x \in I\!R \ , \tag{2.13}$$

see e.g. [DLy], giving, for example, for $k = 3$, $a_0 = a_3 = \frac{1}{4}$, $a_1 = a_2 = \frac{3}{4}$.

In higher dimensions, let

$$X = \{d^1, \ldots, d^r\} \tag{2.14}$$

be a set of direction vectors in $I\!R^d$ containing a basis for $I\!R^d$. From the corresponding symbol

$$a(\mathbf{z}) = 2^{d-r} \prod_{\ell=1}^{r} (1 + \mathbf{z}^{d^\ell}), \tag{2.15}$$

see also [DLy], the mask coefficients are obtained by expanding the product. (If one has several vectors, it is reasonable to use some symbolic system like `Mathematica` or `MATLAB` for the computation.)

For example, the bivariate box spline with direction vectors

$$X = \left\{ \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix} \right\}$$

which is denoted by $M(x|X)$ and which we will use in Section 5 has the symbol

$$\begin{aligned} a(\mathbf{z}) &= 2^{-1}(1+z_1)^2(1+z_2)^2(1+z_1z_2) \\ &= \frac{1}{8} + \frac{1}{4}z_1 + \frac{1}{4}z_2 + \frac{5}{8}z_1z_2 + \frac{1}{8}z_1^2 + \frac{1}{8}z_2^2 + \frac{1}{2}z_1^2z_2 + \frac{1}{2}z_1z_2^2 + \frac{5}{8}z_1^2z_2^2 \\ &\quad + \frac{1}{8}z_1^3z_2 + \frac{1}{8}z_1z_2^3 + \frac{1}{4}z_1^3z_2^2 + \frac{1}{4}z_1^2z_2^3 + \frac{1}{8}z_1^3z_2^3 \end{aligned}$$

so that the indices of the mask coefficients range from `0` to `3` each and the nonzero mask coefficients are given by

```
a[0][0] =  0.125     a[0][1] =  0.25      a[0][2] =  0.125
a[1][0] =  0.25      a[1][1] =  0.625     a[1][2] =  0.5
a[1][3] =  0.125     a[2][0] =  0.125     a[2][1] =  0.5
a[2][2] =  0.625     a[2][3] =  0.25      a[3][1] =  0.125
a[3][2] =  0.25      a[3][3] =  0.125   .
```

3 Description of the Program

For the description of the program, we assume that the reader is familiar with programming in C at least. Almost all parts of the C++–program are in fact functions or structures from C except the classes `mask` (for $d = 1$), `Mask` ($d = 2$) and `MMask` ($d = 3$) where quantities with multiindices like the coefficients in (2.6) are stored. Their definitions can be found in `intmisc.[hC]`. We comment on their element functions below.

The main program `intmain.C` reads in the names of input (and possibly output) data files, the dimension, the derivatives (if different from zero) and the number of refinable functions in the integral, `nrefin`. `nfrefin=1` stands for function evaluations or derivatives of a function at integers. `nrefin=2` means computing inner products, and `nrefin=3` and `nrefin=4` compute the integrals for three, respectively four, factors in the integrand. Finally in `intmain.C` the function `integrals()` is called which has dimension `d` and number of refinable functions `nrefin` as its arguments. The file `intref.C` contains only this function which distinguishes between different cases for `d` and `nrefin`. In each case, first the first and last nonvanishing indices of the mask coefficients from the refinement equation (2.1) for $i = 0$ are read in (which could be negative) and stored in the vector `k_0`. Then the mask coefficients are read in and stored in the vector `a_0` ($d = 1$), the matrix `A_0` ($d = 2$) or the tensor `AA_0` ($d = 3$). Note that for the latter two cases, the data in the input file should be ordered relative to the loops

```
for (i=id; i<= iu; i++)
    for (j=jd; j<= ju; j++)
        for (r=rd; r<= ru; r++)
            read a_ijr ,
```

i.e., the last index is counted first, for example for $d = 2$,

```
a_00    a_01    a_02
a_10    a_11    a_12
a_20    a_21    a_22
```

or for $d = 3$,

```
a_000   a_001   a_002
a_010   a_011   a_012
a_020   a_021   a_022
 ...
a_200   a_201   a_202
a_210   a_211   a_212
a_220   a_221   a_222   .
```

For control purposes, the mask data is given out into the output data file. If none is specified, it will be the file called `stdout`. In the literature, the mask coefficients are often scaled in different ways, e.g. in [Dau2] one assumes that the sum of all mask coefficients is $\sqrt{2}$ in the univariate case. According to the theory in [CDM] and [DM3], the mask coefficients will be scaled in the program such that

$$\sum_{\beta \in \mathbb{Z}^d} a_{\alpha - 2\beta} = 1 \ , \qquad \alpha \in \mathbb{Z}^d \ . \tag{3.1}$$

If `nrefin` > 1, this procedure is repeated for $i \leq$ `nrefin`: first the first and last nonvanishing indices of the mask coefficients are read in and stored in the vector `k_1`, then the mask is read, stored in `a_1`, `A_1` or `AA_1` and scaled relative to (3.1), and so on.

All quantities (vectors, matrices, tensors) allocate their storage dynamically depending on the supports of the masks stored in `k_i`. (Recall that supp $\varphi^i \subseteq [k\_i[0], k\_i[1]]$ when the mask $\mathbf{a}^i$ for φ^i has nonvanishing coefficients $a_{k\_i[0]}, \ldots, a_{k\_i[1]}$. Thus, the ranges of the masks determine the support of the functions F and H which are zero whenever the supports of all refinable functions in the integrals do not overlap.) In particular, the size of the eigenvector problem (2.11) is determined by these values.

Denoting by C the coefficient matrix of the first equation in (2.11) and by W the vector with W_α as its α–th component, the system (2.11) can be written as

$$\begin{pmatrix} C - 2^{-|\mu|}I \\ M \end{pmatrix} W = \begin{pmatrix} 0 \\ r \end{pmatrix} . \tag{3.2}$$

Here M is the matrix built from the left hand side of the second equation in (2.11) and r is the corresponding right hand side. Note that r has only one nonzero entry and that the system in (3.2) has at least as many rows as columns. The additional rows ensure that the system has full rank. In the file `intref.[hC]`, the functions `matrix_(d)d_(nfac)nfac()` set up the matrix A such that the last column of A is the right hand side $\binom{0}{r}$.

For `nrefin>1`, the coefficients c_α in (2.11) are computed in the (overloaded) functions `n()` of the classes `mask`, `Mask` and `MMask` in `intmisc.[hC]`. (In the case `nrefin=1`, the c_α are the a_α from the refinement equation, and no new coefficients have to be computed.)

For the setup of the matrix A, the coefficients with multiindices are transformed as in (3.3). Now the QR–factorization of A (with right hand side as last column of A) `qr(A,...)` and subsequent backsubstitution `backsub()` yields a long vector `h[vd],...,h[vu]`. The transformation of this vector back into coefficients with multiindices (the classes `mask`, `Mask` and `MMask`) is performed as in the example `d=1, nfac=4`:

```
h[lenr*lenj*i + lenr*j + r] = W(i,j,r)                    (3.3)
```

where `len` and `lenr` are the 'length' of the second and third indices. This tranformation is done in the (overloaded) element function `s()` of the classes `mask`, `Mask` and `MMask` for `nrefin>1`. (if `nrefin=1`, this element function is called `sf()`). Finally in `backsub()` the solution $H(\alpha)$ is computed as in (2.12) and its nonzero values are printed out. Note that pointers to the masks `cc, hh` (respectively `Cc, Hh` or `CC, HH`) are returned by the functions used in the function `integrals()` so that one could take up these to write new functions that e.g. compute the values of $H(\alpha)$ at dyadic points.

4 Application of the Program

(For a short version of this section with examples, see the file `read.me`.) After getting the source code as explained at the end of Section 1, the executable file `app` to compile and link the files is obtained by typing `Make` within the UNIX operating system with g++-compilers. If error messages are reported here, move the file `Makefile` to `Makefile.old` and `makefile.old` to `makefile` and try the command `make`. In the case of another C++–compiler like `CC`, replace all commands `g++` by your C++–command in `Makefile` or `makefile` and type `make` again.

After `app` has been generated, it is to be called by

```
> app -i maskin -n nrefin
```

and these are minimal parameters that have to be given in. The number `nrefin` after `-n` denotes the number of refinable functions in the integral. Here `nrefin=1` stands for evaluating a function or its derivative, `nrefin=2` means two factors in the integral, so this is like an inner product, and `nrefin=3` or `4` relates to three, respectively four, factors in the integral. `maskin` is a file containing some necessary input data that must fit to the number given after -n. For example, the input data file for evaluating the univariate B–Spline of order three, N_3, at integers would be : (`nrefin=1`)

```
---maskin---
1
0 3
0.25  0.75  0.75  0.25
------------
```

The first number is the *dimension* d of the underlying space (`d=1` in this example, otherwise `d=1,2` or `d=3`). In the one-dimensional case, the next two numbers are the first and last nonvanishing indices of the mask of the refinable function, here N_3, followed by the mask coefficients `a[0]=0.25, a[1]=0.75, a[2]=0.75, a[3]=0.25`. In two dimensions, the next four numbers after the dimension must be the first and last nonvanishing indices of the first and second index of the mask of the refinable function followed by the mask coefficients like `a[0][0]=0.125, a[0][1]=0.25` etc when the input data file contains the following data. Here is an example for `nrefin=1`, evaluating the bivariate box spline already mentioned at the end of Section 2 with twice the direction vector $(1,0)^T$, twice the direction vector $(0,1)^T$ and once the direction vector $(1,1)^T$:

```
---maskin---
2
0 3 0 3
0.125  0.25   0.125   0.0
0.25   0.625  0.5     0.125
0.125  0.5    0.625   0.25
0.0    0.125  0.25    0.125
------------
```

If the mask values do not satisfy (3.1), they will be scaled accordingly.

When `nrefin=2`, the input data file must contain two pairs of indices and two masks (which could be equal) in the order : range of indices of the first mask, first mask coefficients, range of indices of second mask, second mask coefficients, and so on, for example, when `d=1`:

```
---maskin---
1
0 3
0.25  0.75  0.75  0.25
-1 1
0.5  1.0  0.5
------------
```

Note that the order of the numbers is essential for the program to deliver correct values. Correspondingly, for `nrefin=3` or `4`, one must give in three, respectively four, pairs of indices and three, respectively four, masks (which could all be equal).

When the program has run correctly, it writes

```
---------finished----------- .
```

Without any parameters besides `-i` and `-n` indicated above, the program calculates the desired values without derivatives and writes them into the file `stdout`. Type in `app` to get further options :

```
> app -i maskin -n number [-d derivatives] [-o dataout]
```

When `-o` together with a filename is used, the output data is written into this file instead of `stdout`. The derivatives one wants to compute can be typed in after `-d` (without spacing between these numbers). For instance,

```
> app -i bspline3in -n 1 -d 1 -o bspline3out
```

computes the first derivative of the B–Spline N_3 at integer values and writes the result into the file `bspline3out` when `bspline3in` contains the mask coefficients of N_3.

When the file `4bspline3in` contains four times the mask of N_3,

```
--- 4bspline3in----
1
0 3
0.25  0.75  0.75  0.25
0 3
0.25  0.75  0.75  0.25
0 3
0.25  0.75  0.75  0.25
0 3
0.25  0.75  0.75  0.25
-----------------------
```

one could type in

```
> app -i 4bspline3in -n 4 -d 101 -o 4bspline3out
```

to obain the values

```
hh(i,j,r) = integral of
        phi^0(x)*(D^1 phi^1)(x-i)*(D^0 phi^2)(x-j)*(D^1 phi^3)(x-r)dx
```

for $i, j, r \in \mathbb{Z}$.

In the two–dimensional case, the derivatives should be ordered to give all the derivatives for each function one after another, for instance, when `boxsplinein` is some input data containing three bivariate mask coefficients for box splines, type in

```
> app -i boxsplinein -n 3 -d 1001
```

to compute the values

$$Hh(i,j,r,s) = \int_{\mathbb{R}^d} \varphi^0 \ (D^{(1,0)}\varphi^1)(x-i, y-j) \ (D^{(0,1)}\varphi^2)(x-r, y-s) \ dx \ dx \ .$$

In the case where the number of derivatives is too high according to the smoothness of the refinable function, the program should terminate since then the desired eigenvector is not uniquely determined.

5 Examples

In this section we provide several examples to demonstrate the use of the program. Except for the last two examples which need a couple of minutes on an SGI IP20 (Irix5.2) workstation, all the other examples are computed within a few seconds. For two and three dimensions and more than two factors, the QR factorization of the matrix A takes up most of the time. It is planned to use an iterative solution method in a future version to speed up this computation and make it feasible to compute refinable integrals in three dimensions with four factors.

In order to save space here in the printouts below, the mask and data is printed in two columns. To fit these, at times some zeroes have been discarded.

The first example evaluates the Daubechies' refinable function ϕ_N generating orthogonal wavelets with compact support for $N = 3$ at integers, see [Dau1] and there in particular p. 980 for the mask coefficients. The remaining values for $N = 2, 4, \ldots, 9$ as well as the values for the Daubechies biorthogonal refinable functions φ and $\tilde{\varphi}$ for some N and $\tilde{N}$ from [CDF] can be found in the appendix in [K2].

Example 5.1 (Evaluation of Daubechies' refinable function φ_3)

Task:	Compute $\varphi_3(\alpha)$ for $\alpha \in \mathbb{Z}$
Dimension:	`d=1`
Number of refinable functions:	`nrefin=1`
Input file:	`1_1b`

```
----- 1_1b ----------------------
1
0 5
0.332670552950
0.806891509311
0.459877502118
-0.135011020010
-0.085441273882
0.035226291882
---------------------------------------
```

Call: `app -i 1_1b -n 1`
Output file: `stdout`

```
----- stdout -------------------------
read-in-file:  1_1b
read-out-file: stdout
---------------------------------------
mask:
a[ 0]= 3.326705529500e-01  a[ 1]= 8.068915093110e-01
a[ 2]= 4.598775021180e-01  a[ 3]=-1.350110200100e-01
a[ 4]=-8.544127388200e-02  a[ 5]= 3.522629188200e-02
alpha = (0) :
sum of mask coefficients before scaling =  7.071067811860e-01
sum of mask coefficients after  scaling =  1.0000000000
alpha = (1) :
sum of mask coefficients before scaling =  7.071067811860e-01
sum of mask coefficients after  scaling =  1.0000000000

mask coefficients after scaling:
a[ 0]= 4.704672077844e-01  a[ 1]= 1.141116915837e+00
a[ 2]= 6.503650005260e-01  a[ 3]=-1.909344155689e-01
a[ 4]=-1.208322083105e-01  a[ 5]= 4.981749973189e-02

---------------------------------------
solution:
(D^0 phi)(1) = 1.286335069451e+00  (D^0 phi)(2) = -3.858369610687e-01
(D^0 phi)(3) = 9.526754600145e-02  (D^0 phi)(4) =  4.234345615870e-03

=======================================
```

In the following examples, the mask coefficients already satisfy (3.1) so that the printout of the scaled mask coefficients has been discarded.

Example 5.2 (Evaluation of a bivariate box spline)

Task:	Compute $D^{(0,1)}M\left(\alpha \,\middle|\, \begin{smallmatrix}1\,1\,0\,0\,1\\0\,0\,1\,1\,1\end{smallmatrix}\right)$ for $\alpha \in \mathbb{Z}^2$
Dimension:	`d=2`
Number of refinable functions:	`nrefin=1`
Input file:	`2_1_221`

```
------ 2_1_221 ------------------------
2
0 3 0 3
0.125    0.25     0.125     0.0
0.25     0.625    0.5       0.125
0.125    0.5      0.625     0.25
0.0      0.125    0.25      0.125
---------------------------------------
```

Call:	`app -i 2_1_221 -n 1 -d 01`
Output file:	`stdout`

```
------ stdout ------------------------
read-in-file:  2_1_221
read-out-file: stdout
---------------------------------------
mask:
a[ 0][ 0] =  1.250000000000e-01  a[ 0][ 1] =  2.500000000000e-01
a[ 0][ 2] =  1.250000000000e-01  a[ 1][ 0] =  2.500000000000e-01
a[ 1][ 1] =  6.250000000000e-01  a[ 1][ 2] =  5.000000000000e-01
a[ 1][ 3] =  1.250000000000e-01  a[ 2][ 0] =  1.250000000000e-01
a[ 2][ 1] =  5.000000000000e-01  a[ 2][ 2] =  6.250000000000e-01
a[ 2][ 3] =  2.500000000000e-01  a[ 3][ 1] =  1.250000000000e-01
a[ 3][ 2] =  2.500000000000e-01  a[ 3][ 3] =  1.250000000000e-01
---------------------------------------
solution:
(D^(0,1) phi)( 1, 1) =  5.00e-01   (D^(0,1) phi)( 1, 2) = -5.00e-01
(D^(0,1) phi)( 2, 1) =  5.00e-01   (D^(0,1) phi)( 2, 2) = -5.00e-01
=======================================
```

Example 5.3 (Evaluation of a three–dimensional box spline)

Task:	Compute $M\left(\alpha \,\middle|\, \begin{smallmatrix}1\,1\,0\,0\,0\,1\\0\,0\,1\,0\,0\,1\\0\,0\,0\,1\,1\,1\end{smallmatrix}\right)$ for $\alpha \in \mathbb{Z}^3$
Dimension:	`d=3`
Number of refinable functions:	`nrefin=1`
Input file:	`3_1_2121`

```
------ 3_1_2121 -------------------------
3
0 4 0 4 0 4
0.0625   0.0625   0.0      0.0      0.0
0.0625   0.125    0.0625   0.0      0.0
0.0      0.0625   0.0625   0.0      0.0
0.0      0.0      0.0      0.0      0.0
0.0      0.0      0.0      0.0      0.0

0.0625   0.125    0.0625   0.0      0.0
0.125    0.3125   0.25     0.0625   0.0
0.0625   0.25     0.3125   0.125    0.0
0.0      0.0625   0.125    0.0625   0.0
0.0      0.0      0.0      0.0      0.0

0.0      0.0625   0.0625   0.0      0.0
0.0625   0.25     0.3125   0.125    0.0
0.0625   0.3125   0.5      0.3125   0.0625
0.0      0.125    0.3125   0.25     0.0625
0.0      0.0      0.0625   0.0625   0.0

0.0      0.0      0.0      0.0      0.0
0.0      0.0625   0.125    0.0625   0.0
0.0      0.125    0.3125   0.25     0.0625
0.0      0.0625   0.25     0.3125   0.125
0.0      0.0      0.0625   0.125    0.0625

0.0      0.0      0.0      0.0      0.0
0.0      0.0      0.0      0.0      0.0
0.0      0.0      0.0625   0.0625   0.0
0.0      0.0      0.0625   0.125    0.0625
0.0      0.0      0.0      0.0625   0.0625
---------------------------------------
```

Call:	`app -i 3_1_2121 -n 1`
Output file:	`stdout`

```
------ stdout -------------------------
read-in-file:  3_1_2121
read-out-file: stdout
---------------------------------------
mask:
a[ 0][ 0][ 0] = 6.250000000e-02  a[ 0][ 0][ 1] = 6.250000000e-02
a[ 0][ 1][ 0] = 6.250000000e-02  a[ 0][ 1][ 1] = 1.250000000e-01
a[ 0][ 1][ 2] = 6.250000000e-02  a[ 0][ 2][ 1] = 6.250000000e-02
```

```
a[ 0][ 2][ 2] = 6.250000000e-02  a[ 1][ 0][ 0] = 6.250000000e-02
a[ 1][ 0][ 1] = 1.250000000e-01  a[ 1][ 0][ 2] = 6.250000000e-02
a[ 1][ 1][ 0] = 1.250000000e-01  a[ 1][ 1][ 1] = 3.125000000e-01
a[ 1][ 1][ 2] = 2.500000000e-01  a[ 1][ 1][ 3] = 6.250000000e-02
a[ 1][ 2][ 0] = 6.250000000e-02  a[ 1][ 2][ 1] = 2.500000000e-01
a[ 1][ 2][ 2] = 3.125000000e-01  a[ 1][ 2][ 3] = 1.250000000e-01
a[ 1][ 3][ 1] = 6.250000000e-02  a[ 1][ 3][ 2] = 1.250000000e-01
a[ 1][ 3][ 3] = 6.250000000e-02  a[ 2][ 0][ 1] = 6.250000000e-02
a[ 2][ 0][ 2] = 6.250000000e-02  a[ 2][ 1][ 0] = 6.250000000e-02
a[ 2][ 1][ 1] = 2.500000000e-01  a[ 2][ 1][ 2] = 3.125000000e-01
a[ 2][ 1][ 3] = 1.250000000e-01  a[ 2][ 2][ 0] = 6.250000000e-02
a[ 2][ 2][ 1] = 3.125000000e-01  a[ 2][ 2][ 2] = 5.000000000e-01
a[ 2][ 2][ 3] = 3.125000000e-01  a[ 2][ 2][ 4] = 6.250000000e-02
a[ 2][ 3][ 1] = 1.250000000e-01  a[ 2][ 3][ 2] = 3.125000000e-01
a[ 2][ 3][ 3] = 2.500000000e-01  a[ 2][ 3][ 4] = 6.250000000e-02
a[ 2][ 4][ 2] = 6.250000000e-02  a[ 2][ 4][ 3] = 6.250000000e-02
a[ 3][ 1][ 1] = 6.250000000e-02  a[ 3][ 1][ 2] = 1.250000000e-01
a[ 3][ 1][ 3] = 6.250000000e-02  a[ 3][ 2][ 1] = 1.250000000e-01
a[ 3][ 2][ 2] = 3.125000000e-01  a[ 3][ 2][ 3] = 2.500000000e-01
a[ 3][ 2][ 4] = 6.250000000e-02  a[ 3][ 3][ 1] = 6.250000000e-02
a[ 3][ 3][ 2] = 2.500000000e-01  a[ 3][ 3][ 3] = 3.125000000e-01
a[ 3][ 3][ 4] = 1.250000000e-01  a[ 3][ 4][ 2] = 6.250000000e-02
a[ 3][ 4][ 3] = 1.250000000e-01  a[ 3][ 4][ 4] = 6.250000000e-02
a[ 4][ 2][ 2] = 6.250000000e-02  a[ 4][ 2][ 3] = 6.250000000e-02
a[ 4][ 3][ 2] = 6.250000000e-02  a[ 4][ 3][ 3] = 1.250000000e-01
a[ 4][ 3][ 4] = 6.250000000e-02  a[ 4][ 4][ 3] = 6.250000000e-02
a[ 4][ 4][ 4] = 6.250000000e-02
-------------------------------------
solution:
(D^(0,0,0) phi)( 1, 1, 1) =  6.250000000000e-02
(D^(0,0,0) phi)( 1, 1, 2) =  2.083333333333e-02
(D^(0,0,0) phi)( 1, 2, 1) =  2.083333333333e-02
(D^(0,0,0) phi)( 1, 2, 2) =  6.250000000000e-02
(D^(0,0,0) phi)( 2, 1, 1) =  2.083333333333e-02
(D^(0,0,0) phi)( 2, 1, 2) =  6.250000000000e-02
(D^(0,0,0) phi)( 2, 2, 1) =  6.250000000000e-02
(D^(0,0,0) phi)( 2, 2, 2) =  3.750000000000e-01
(D^(0,0,0) phi)( 2, 2, 3) =  6.250000000000e-02
(D^(0,0,0) phi)( 2, 3, 2) =  6.250000000000e-02
(D^(0,0,0) phi)( 2, 3, 3) =  2.083333333333e-02
(D^(0,0,0) phi)( 3, 2, 2) =  6.250000000000e-02
(D^(0,0,0) phi)( 3, 2, 3) =  2.083333333333e-02
(D^(0,0,0) phi)( 3, 3, 2) =  2.083333333333e-02
(D^(0,0,0) phi)( 3, 3, 3) =  6.250000000000e-02
```

```
=======================================
```

Example 5.4 (Four factors in the integral)

Task:	Compute
$\int_{\mathbb{R}} \chi_{[0,1)}\ N_3(x-i)\ N_3'(x-j)\ N_3'(x-r)\ dx$	
	(N_3 cardinal B–Spline of order 3)
	for $i, j, r \in \mathbb{Z}$
Dimension:	`d=1`
Number of refinable functions:	`nrefin=4`
Input file:	`1_4a`

```
------ 1_4a --------------------------
1
0 1
1.0    1.0
0 3
0.25   0.75   0.75   0.25
0 3
0.25   0.75   0.75   0.25
0 3
0.25   0.75   0.75   0.25
---------------------------------------
```

Call:	`app -i 1_4a -n 4 -d 011`
Output file:	`stdout`

```
------ stdout -------------------------
read-in-file:  1_4a
read-out-file: stdout
---------------------------------------
mask of first factor:
a[ 0]= 1.000000000e+00  a[ 1]= 1.000000000e+00
---------------------------------------
mask of second factor:
a[ 0]= 2.500000000e-01  a[ 1]= 7.500000000e-01
a[ 2]= 7.500000000e-01  a[ 3]= 2.500000000e-01
---------------------------------------
mask of third factor:
a[ 0]= 2.500000000e-01  a[ 1]= 7.500000000e-01
a[ 2]= 7.500000000e-01  a[ 3]= 2.500000000e-01
---------------------------------------
mask of fourth factor:
a[ 0]= 2.500000000e-01  a[ 1]= 7.500000000e-01
a[ 2]= 7.500000000e-01  a[ 3]= 2.500000000e-01
---------------------------------------
```

```
hh(i,j,r) = integral of
   phi_0(x)*(D^0 phi_1)(x-i)*(D^1 phi_2)(x-j)*(D^1 phi_3)(x-r)dx =
hh(-2,-2,-2) =  1.000000000000e-01   hh(-2,-2,-1) = -7.500000000000e-02
hh(-2,-2, 0) = -2.500000000000e-02   hh(-2,-1,-2) = -7.500000000000e-02
hh(-2,-1,-1) =  6.666666666667e-02   hh(-2,-1, 0) =  8.333333333333e-03
hh(-2, 0,-2) = -2.500000000000e-02   hh(-2, 0,-1) =  8.333333333333e-03
hh(-2, 0, 0) =  1.666666666667e-02   hh(-1,-2,-2) =  2.166666666667e-01
hh(-1,-2,-1) = -1.000000000000e-01   hh(-1,-2, 0) = -1.166666666667e-01
hh(-1,-1,-2) = -1.000000000000e-01   hh(-1,-1,-1) =  2.000000000000e-01
hh(-1,-1, 0) = -1.000000000000e-01   hh(-1, 0,-2) = -1.166666666667e-01
hh(-1, 0,-1) = -1.000000000000e-01   hh(-1, 0, 0) =  2.166666666667e-01
hh( 0,-2,-2) =  1.666666666667e-02   hh( 0,-2,-1) =  8.333333333333e-03
hh( 0,-2, 0) = -2.500000000000e-02   hh( 0,-1,-2) =  8.333333333333e-03
hh( 0,-1,-1) =  6.666666666667e-02   hh( 0,-1, 0) = -7.500000000000e-02
hh( 0, 0,-2) = -2.500000000000e-02   hh( 0, 0,-1) = -7.500000000000e-02
hh( 0, 0, 0) =  1.000000000000e-01
=======================================
```

Example 5.5 (Three factors in the integral)

Task:	Compute

$$\int_{\mathbb{R}^2} \chi_{[0,1)^2}(x,y)\ (D^{(1,0)}M)\left(x-i, y-j\,\middle|\,\begin{matrix}1\,1\,0\,0\,1\\0\,0\,1\,1\,1\end{matrix}\right) \cdot (D^{(1,0)}M)\left(x-r, y-s\,\middle|\,\begin{matrix}1\,1\,0\,0\,1\\0\,0\,1\,1\,1\end{matrix}\right)\ dx\,dy$$

for $i,j,r,s \in \mathbb{Z}$

Dimension:	d=2
Number of refinable functions:	nrefin=3
Input file:	2_3_221

```
----- 2_3_221 -------------------------
2
0 1 0 1
1.0  1.0
1.0  1.0
0 3 0 3
0.125  0.25   0.125   0.0
0.25   0.625  0.5     0.125
0.125  0.5    0.625   0.25
0.0    0.125  0.25    0.125
0 3 0 3
0.125  0.25   0.125   0.0
0.25   0.625  0.5     0.125
0.125  0.5    0.625   0.25
0.0    0.125  0.25    0.125
---------------------------------------
```

Call: `app -i 2_3_221 -n 3 -d 1010`
Output file: `stdout`

```
----- stdout -------------------------
read-in-file:  2_3_221
read-out-file: stdout
--------------------------------------
mask of first factor:
a[ 0][ 0] =  1.000000000e+00  a[ 0][ 1] =  1.000000000e+00
a[ 1][ 0] =  1.000000000e+00  a[ 1][ 1] =  1.000000000e+00
--------------------------------------
mask of second factor:
a[ 0][ 0] =  1.250000000e-01  a[ 0][ 1] =  2.500000000e-01
a[ 0][ 2] =  1.250000000e-01  a[ 1][ 0] =  2.500000000e-01
a[ 1][ 1] =  6.250000000e-01  a[ 1][ 2] =  5.000000000e-01
a[ 1][ 3] =  1.250000000e-01  a[ 2][ 0] =  1.250000000e-01
a[ 2][ 1] =  5.000000000e-01  a[ 2][ 2] =  6.250000000e-01
a[ 2][ 3] =  2.500000000e-01  a[ 3][ 1] =  1.250000000e-01
a[ 3][ 2] =  2.500000000e-01  a[ 3][ 3] =  1.250000000e-01
--------------------------------------
mask of third factor:
a[ 0][ 0] =  1.250000000e-01  a[ 0][ 1] =  2.500000000e-01
a[ 0][ 2] =  1.250000000e-01  a[ 1][ 0] =  2.500000000e-01
a[ 1][ 1] =  6.250000000e-01  a[ 1][ 2] =  5.000000000e-01
a[ 1][ 3] =  1.250000000e-01  a[ 2][ 0] =  1.250000000e-01
a[ 2][ 1] =  5.000000000e-01  a[ 2][ 2] =  6.250000000e-01
a[ 2][ 3] =  2.500000000e-01  a[ 3][ 1] =  1.250000000e-01
a[ 3][ 2] =  2.500000000e-01  a[ 3][ 3] =  1.250000000e-01
--------------------------------------
solution:
Hh(i,j,r,s) = integral of
   phi_0(x,y)*(D^(1,0)phi_1)(x-i,y-j)*(D^(1,0)phi_2)(x-r,y-s) dx dy :
Hh(-2,-2,-2,-2)=  3.055555555555e-02 Hh(-2,-2,-2,-1)=  4.305555555556e-02
Hh(-2,-2,-2, 0)=  1.388888888889e-03 Hh(-2,-2,-1,-2)= -2.500000000000e-02
Hh(-2,-2,-1,-1)= -4.166666666667e-03 Hh(-2,-2,-1, 0)=  4.166666666667e-03
Hh(-2,-2, 0,-2)= -5.555555555555e-03 Hh(-2,-2, 0,-1)= -3.888888888889e-02
Hh(-2,-2, 0, 0)= -5.555555555556e-03 Hh(-2,-1,-2,-2)=  4.305555555556e-02
Hh(-2,-1,-2,-1)=  1.583333333333e-01 Hh(-2,-1,-2, 0)=  2.361111111111e-02
Hh(-2,-1,-1,-2)= -4.027777777778e-02 Hh(-2,-1,-1,-1)= -9.166666666667e-02
Hh(-2,-1,-1, 0)=  1.527777777778e-02 Hh(-2,-1, 0,-2)= -2.777777777778e-03
Hh(-2,-1, 0,-1)= -6.666666666667e-02 Hh(-2,-1, 0, 0)= -3.888888888889e-02
Hh(-2, 0,-2,-2)=  1.388888888889e-03 Hh(-2, 0,-2,-1)=  2.361111111111e-02
Hh(-2, 0,-2, 0)=  8.333333333333e-03 Hh(-2, 0,-1,-2)= -1.388888888889e-03
Hh(-2, 0,-1,-1)= -2.083333333333e-02 Hh(-2, 0,-1, 0)= -2.777777777778e-03
Hh(-2, 0, 0,-1)= -2.777777777778e-03 Hh(-2, 0, 0, 0)= -5.555555555556e-03
Hh(-1,-2,-2,-2)= -2.500000000000e-02 Hh(-1,-2,-2,-1)= -4.027777777778e-02
Hh(-1,-2,-2, 0)= -1.388888888889e-03 Hh(-1,-2,-1,-2)=  2.777777777778e-02
Hh(-1,-2,-1,-1)=  2.500000000000e-02 Hh(-1,-2,-1, 0)= -2.777777777778e-03
Hh(-1,-2, 0,-2)= -2.777777777777e-03 Hh(-1,-2, 0,-1)=  1.527777777778e-02
Hh(-1,-2, 0, 0)=  4.166666666666e-03 Hh(-1,-1,-2,-2)= -4.166666666666e-03
Hh(-1,-1,-2,-1)= -9.166666666667e-02 Hh(-1,-1,-2, 0)= -2.083333333333e-02
Hh(-1,-1,-1,-2)=  2.500000000000e-02 Hh(-1,-1,-1,-1)=  1.833333333333e-01
Hh(-1,-1,-1, 0)=  2.500000000000e-02 Hh(-1,-1, 0,-2)= -2.083333333333e-02
Hh(-1,-1, 0,-1)= -9.166666666667e-02 Hh(-1,-1, 0, 0)= -4.166666666666e-03
Hh(-1, 0,-2,-2)=  4.166666666667e-03 Hh(-1, 0,-2,-1)=  1.527777777778e-02
Hh(-1, 0,-2, 0)= -2.777777777778e-03 Hh(-1, 0,-1,-2)= -2.777777777778e-03
Hh(-1, 0,-1,-1)=  2.500000000000e-02 Hh(-1, 0,-1, 0)=  2.777777777778e-02
Hh(-1, 0, 0,-2)= -1.388888888889e-03 Hh(-1, 0, 0,-1)= -4.027777777778e-02
Hh(-1, 0, 0, 0)= -2.500000000000e-02 Hh( 0,-2,-2,-2)= -5.555555555555e-03
Hh( 0,-2,-2,-1)= -2.777777777778e-03 Hh( 0,-2,-1,-2)= -2.777777777778e-03
Hh( 0,-2,-1,-1)= -2.083333333333e-02 Hh( 0,-2,-1, 0)= -1.388888888889e-03
Hh( 0,-2, 0,-2)=  8.333333333333e-03 Hh( 0,-2, 0,-1)=  2.361111111111e-02
Hh( 0,-2, 0, 0)=  1.388888888889e-03 Hh( 0,-1,-2,-2)= -3.888888888889e-02
Hh( 0,-1,-2,-1)= -6.666666666667e-02 Hh( 0,-1,-2, 0)= -2.777777777778e-03
Hh( 0,-1,-1,-2)=  1.527777777778e-02 Hh( 0,-1,-1,-1)= -9.166666666667e-02
Hh( 0,-1,-1, 0)= -4.027777777778e-02 Hh( 0,-1, 0,-2)=  2.361111111111e-02
Hh( 0,-1, 0,-1)=  1.583333333333e-01 Hh( 0,-1, 0, 0)=  4.305555555556e-02
Hh( 0, 0,-2,-2)= -5.555555555556e-03 Hh( 0, 0,-2,-1)= -3.888888888889e-02
Hh( 0, 0,-2, 0)= -5.555555555556e-03 Hh( 0, 0,-1,-2)=  4.166666666667e-03
Hh( 0, 0,-1,-1)= -4.166666666667e-03 Hh( 0, 0,-1, 0)= -2.500000000000e-02
Hh( 0, 0, 0,-2)=  1.388888888889e-03 Hh( 0, 0, 0,-1)=  4.305555555556e-02
Hh( 0, 0, 0, 0)=  3.055555555556e-02
==========================================
```

In the last example, we list here the mask coefficients in the input and the output only once.

Example 5.6 (Two factors in the integral)

Task: Compute

$$\int_{\mathbb{R}^3} M(x,y,z)\ M(x-i,y-j,z-r)\ dx\,dy\,dz$$

$$\text{for } M(x,y,z) = M\left(x,y,z \,\middle|\, \begin{matrix} 1\,1\,0\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1 \\ 0\,0\,0\,1\,1\,1 \end{matrix}\right)$$

and $i,j,r \in \mathbb{Z}$

Dimension:	`d=3`
Number of refinable functions:	`nrefin=2`
Input file:	`3_2_2121`

```
------ 3_2_2121 --------------------------
3
```

```
0 4 0 4 0 4
0.0625    0.0625    0.0       0.0      0.0
0.0625    0.125     0.0625    0.0      0.0
0.0       0.0625    0.0625    0.0      0.0
0.0       0.0       0.0       0.0      0.0
0.0       0.0       0.0       0.0      0.0

0.0625    0.125     0.0625    0.0      0.0
0.125     0.3125    0.25      0.0625   0.0
0.0625    0.25      0.3125    0.125    0.0
0.0       0.0625    0.125     0.0625   0.0
0.0       0.0       0.0       0.0      0.0

0.0       0.0625    0.0625    0.0      0.0
0.0625    0.25      0.3125    0.125    0.0
0.0625    0.3125    0.5       0.3125   0.0625
0.0       0.125     0.3125    0.25     0.0625
0.0       0.0       0.0625    0.0625   0.0

0.0       0.0       0.0       0.0      0.0
0.0       0.0625    0.125     0.0625   0.0
0.0       0.125     0.3125    0.25     0.0625
0.0       0.0625    0.25      0.3125   0.125
0.0       0.0       0.0625    0.125    0.0625

0.0       0.0       0.0       0.0      0.0
0.0       0.0       0.0       0.0      0.0
0.0       0.0       0.0625    0.0625   0.0
0.0       0.0       0.0625    0.125    0.0625
0.0       0.0       0.0       0.0625   0.0625
---------------------------------------
```

Call:	`app -i 3_2_2121 -n 2`
Output file:	`stdout`

```
------- stdout ------------------------
read-in-file:  3_2_2121
read-out-file: stdout
---------------------------------------
mask of first factor:
a[ 0][ 0][ 0] =  6.250000000e-02  a[ 0][ 0][ 1] =  6.250000000e-02
a[ 0][ 1][ 0] =  6.250000000e-02  a[ 0][ 1][ 1] =  1.250000000e-01
a[ 0][ 1][ 2] =  6.250000000e-02  a[ 0][ 2][ 1] =  6.250000000e-02
a[ 0][ 2][ 2] =  6.250000000e-02  a[ 1][ 0][ 0] =  6.250000000e-02
a[ 1][ 0][ 1] =  1.250000000e-01  a[ 1][ 0][ 2] =  6.250000000e-02
```

```
a[ 1][ 1][ 0] =  1.250000000e-01  a[ 1][ 1][ 1] =  3.125000000e-01
a[ 1][ 1][ 2] =  2.500000000e-01  a[ 1][ 1][ 3] =  6.250000000e-02
a[ 1][ 2][ 0] =  6.250000000e-02  a[ 1][ 2][ 1] =  2.500000000e-01
a[ 1][ 2][ 2] =  3.125000000e-01  a[ 1][ 2][ 3] =  1.250000000e-01
a[ 1][ 3][ 1] =  6.250000000e-02  a[ 1][ 3][ 2] =  1.250000000e-01
a[ 1][ 3][ 3] =  6.250000000e-02  a[ 2][ 0][ 1] =  6.250000000e-02
a[ 2][ 0][ 2] =  6.250000000e-02  a[ 2][ 1][ 0] =  6.250000000e-02
a[ 2][ 1][ 1] =  2.500000000e-01  a[ 2][ 1][ 2] =  3.125000000e-01
a[ 2][ 1][ 3] =  1.250000000e-01  a[ 2][ 2][ 0] =  6.250000000e-02
a[ 2][ 2][ 1] =  3.125000000e-01  a[ 2][ 2][ 2] =  5.000000000e-01
a[ 2][ 2][ 3] =  3.125000000e-01  a[ 2][ 2][ 4] =  6.250000000e-02
a[ 2][ 3][ 1] =  1.250000000e-01  a[ 2][ 3][ 2] =  3.125000000e-01
a[ 2][ 3][ 3] =  2.500000000e-01  a[ 2][ 3][ 4] =  6.250000000e-02
a[ 2][ 4][ 2] =  6.250000000e-02  a[ 2][ 4][ 3] =  6.250000000e-02
a[ 3][ 1][ 1] =  6.250000000e-02  a[ 3][ 1][ 2] =  1.250000000e-01
a[ 3][ 1][ 3] =  6.250000000e-02  a[ 3][ 2][ 1] =  1.250000000e-01
a[ 3][ 2][ 2] =  3.125000000e-01  a[ 3][ 2][ 3] =  2.500000000e-01
a[ 3][ 2][ 4] =  6.250000000e-02  a[ 3][ 3][ 1] =  6.250000000e-02
a[ 3][ 3][ 2] =  2.500000000e-01  a[ 3][ 3][ 3] =  3.125000000e-01
a[ 3][ 3][ 4] =  1.250000000e-01  a[ 3][ 4][ 2] =  6.250000000e-02
a[ 3][ 4][ 3] =  1.250000000e-01  a[ 3][ 4][ 4] =  6.250000000e-02
a[ 4][ 2][ 2] =  6.250000000e-02  a[ 4][ 2][ 3] =  6.250000000e-02
a[ 4][ 3][ 2] =  6.250000000e-02  a[ 4][ 3][ 3] =  1.250000000e-01
a[ 4][ 3][ 4] =  6.250000000e-02  a[ 4][ 4][ 3] =  6.250000000e-02
a[ 4][ 4][ 4] =  6.250000000e-02
---------------------------------------
mask of second factor:  ...
---------------------------------------
solution:
HH(i,j,r) = integral of
     phi_0(x,y,z)*(D^(0,0,0)phi_1)(x-i,y-j,z-r) dx dy dz :
HH(-3,-3,-3)= 1.174317580585e-06  HH(-3,-3,-2)= 4.164913019099e-06
HH(-3,-3,-1)= 1.722332451530e-07  HH(-3,-2,-3)= 4.164913019101e-06
HH(-3,-2,-2)= 6.048518418314e-05  HH(-3,-2,-1)= 2.887255491424e-05
HH(-3,-2, 0)= 1.722332451434e-07  HH(-3,-1,-3)= 1.722332451529e-07
HH(-3,-1,-2)= 2.887255491425e-05  HH(-3,-1,-1)= 6.048518418313e-05
HH(-3,-1, 0)= 4.164913019092e-06  HH(-3, 0,-2)= 1.722332451244e-07
HH(-3, 0,-1)= 4.164913019113e-06  HH(-3, 0, 0)= 1.174317580571e-06
HH(-2,-3,-3)= 4.164913019066e-06  HH(-2,-3,-2)= 6.048518418308e-05
HH(-2,-3,-1)= 2.887255491421e-05  HH(-2,-3, 0)= 1.722332451131e-07
HH(-2,-2,-3)= 6.048518418310e-05  HH(-2,-2,-2)= 1.941287878788e-03
HH(-2,-2,-1)= 2.566932970579e-03  HH(-2,-2, 0)= 2.150723504890e-04
HH(-2,-2, 1)= 1.722332451207e-07  HH(-2,-1,-3)= 2.887255491423e-05
HH(-2,-1,-2)= 2.566932970579e-03  HH(-2,-1,-1)= 8.862621753247e-03
HH(-2,-1, 0)= 2.566932970579e-03  HH(-2,-1, 1)= 2.887255491427e-05
HH(-2, 0,-3)= 1.722332451424e-07  HH(-2, 0,-2)= 2.150723504890e-04
HH(-2, 0,-1)= 2.566932970579e-03  HH(-2, 0, 0)= 1.941287878788e-03
HH(-2, 0, 1)= 6.048518418312e-05  HH(-2, 1,-2)= 1.722332451709e-07
HH(-2, 1,-1)= 2.887255491425e-05  HH(-2, 1, 0)= 6.048518418311e-05
HH(-2, 1, 1)= 4.164913019109e-06  HH(-1,-3,-3)= 1.722332451225e-07
HH(-1,-3,-2)= 2.887255491418e-05  HH(-1,-3,-1)= 6.048518418305e-05
HH(-1,-3, 0)= 4.164913019053e-06  HH(-1,-2,-3)= 2.887255491421e-05
HH(-1,-2,-2)= 2.566932970579e-03  HH(-1,-2,-1)= 8.862621753247e-03
HH(-1,-2, 0)= 2.566932970579e-03  HH(-1,-2, 1)= 2.887255491418e-05
HH(-1,-1,-3)= 6.048518418310e-05  HH(-1,-1,-2)= 8.862621753247e-03
HH(-1,-1,-1)= 5.461605451840e-02  HH(-1,-1, 0)= 3.789657487574e-02
HH(-1,-1, 1)= 2.566932970579e-03  HH(-1,-1, 2)= 4.164913019114e-06
HH(-1, 0,-3)= 4.164913019094e-06  HH(-1, 0,-2)= 2.566932970579e-03
HH(-1, 0,-1)= 3.789657487574e-02  HH(-1, 0, 0)= 5.461605451840e-02
HH(-1, 0, 1)= 8.862621753247e-03  HH(-1, 0, 2)= 6.048518418314e-05
HH(-1, 1,-2)= 2.887255491424e-05  HH(-1, 1,-1)= 2.566932970579e-03
HH(-1, 1, 0)= 8.862621753247e-03  HH(-1, 1, 1)= 2.566932970579e-03
HH(-1, 1, 2)= 2.887255491427e-05  HH(-1, 2,-1)= 4.164913019090e-06
HH(-1, 2, 0)= 6.048518418313e-05  HH(-1, 2, 1)= 2.887255491424e-05
HH(-1, 2, 2)= 1.722332451722e-07  HH( 0,-3,-2)= 1.722332451285e-07
HH( 0,-3,-1)= 4.164913019066e-06  HH( 0,-3, 0)= 1.174317580551e-06
HH( 0,-2,-3)= 1.722332451417e-07  HH( 0,-2,-2)= 2.150723504890e-04
HH( 0,-2,-1)= 2.566932970579e-03  HH( 0,-2, 0)= 1.941287878788e-03
HH( 0,-2, 1)= 6.048518418308e-05  HH( 0,-1,-3)= 4.164913019079e-06
HH( 0,-1,-2)= 2.566932970579e-03  HH( 0,-1,-1)= 3.789657487574e-02
HH( 0,-1, 0)= 5.461605451840e-02  HH( 0,-1, 1)= 8.862621753247e-03
HH( 0,-1, 2)= 6.048518418310e-05  HH( 0, 0,-3)= 1.174317580575e-06
HH( 0, 0,-2)= 1.941287878788e-03  HH( 0, 0,-1)= 5.461605451840e-02
HH( 0, 0, 0)= 1.486554533430e-01  HH( 0, 0, 1)= 5.461605451840e-02
HH( 0, 0, 2)= 1.941287878788e-03  HH( 0, 0, 3)= 1.174317580567e-06
HH( 0, 1,-2)= 6.048518418310e-05  HH( 0, 1,-1)= 8.862621753247e-03
HH( 0, 1, 0)= 5.461605451840e-02  HH( 0, 1, 1)= 3.789657487574e-02
HH( 0, 1, 2)= 2.566932970579e-03  HH( 0, 1, 3)= 4.164913019085e-06
HH( 0, 2,-1)= 6.048518418311e-05  HH( 0, 2, 0)= 1.941287878788e-03
HH( 0, 2, 1)= 2.566932970579e-03  HH( 0, 2, 2)= 2.150723504890e-04
HH( 0, 2, 3)= 1.722332451582e-07  HH( 0, 3, 0)= 1.174317580572e-06
HH( 0, 3, 1)= 4.164913019086e-06  HH( 0, 3, 2)= 1.722332451545e-07
HH( 1,-2,-2)= 1.722332451408e-07  HH( 1,-2,-1)= 2.887255491420e-05
HH( 1,-2, 0)= 6.048518418309e-05  HH( 1,-2, 1)= 4.164913019079e-06
HH( 1,-1,-2)= 2.887255491421e-05  HH( 1,-1,-1)= 2.566932970579e-03
HH( 1,-1, 0)= 8.862621753247e-03  HH( 1,-1, 1)= 2.566932970579e-03
HH( 1,-1, 2)= 2.887255491422e-05  HH( 1, 0,-2)= 6.048518418309e-05
HH( 1, 0,-1)= 8.862621753247e-03  HH( 1, 0, 0)= 5.461605451840e-02
HH( 1, 0, 1)= 3.789657487574e-02  HH( 1, 0, 2)= 2.566932970579e-03
HH( 1, 0, 3)= 4.164913019082e-06  HH( 1, 1,-2)= 4.164913019081e-06
HH( 1, 1,-1)= 2.566932970579e-03  HH( 1, 1, 0)= 3.789657487574e-02
HH( 1, 1, 1)= 5.461605451840e-02  HH( 1, 1, 2)= 8.862621753247e-03
HH( 1, 1, 3)= 6.048518418310e-05  HH( 1, 2,-1)= 2.887255491422e-05
HH( 1, 2, 0)= 2.566932970579e-03  HH( 1, 2, 1)= 8.862621753247e-03
HH( 1, 2, 2)= 2.566932970579e-03  HH( 1, 2, 3)= 2.887255491422e-05
HH( 1, 3, 0)= 4.164913019080e-06  HH( 1, 3, 1)= 6.048518418310e-05
HH( 1, 3, 2)= 2.887255491422e-05  HH( 1, 3, 3)= 1.722332451506e-07
HH( 2,-1,-1)= 4.164913019080e-06  HH( 2,-1, 0)= 6.048518418310e-05
HH( 2,-1, 1)= 2.887255491422e-05  HH( 2,-1, 2)= 1.722332451508e-07
HH( 2, 0,-1)= 6.048518418310e-05  HH( 2, 0, 0)= 1.941287878788e-03
HH( 2, 0, 1)= 2.566932970579e-03  HH( 2, 0, 2)= 2.150723504890e-04
HH( 2, 0, 3)= 1.722332451504e-07  HH( 2, 1,-1)= 2.887255491422e-05
HH( 2, 1, 0)= 2.566932970579e-03  HH( 2, 1, 1)= 8.862621753247e-03
HH( 2, 1, 2)= 2.566932970579e-03  HH( 2, 1, 3)= 2.887255491422e-05
HH( 2, 2,-1)= 1.722332451499e-07  HH( 2, 2, 0)= 2.150723504890e-04
HH( 2, 2, 1)= 2.566932970579e-03  HH( 2, 2, 2)= 1.941287878788e-03
HH( 2, 2, 3)= 6.048518418310e-05  HH( 2, 3, 0)= 1.722332451500e-07
HH( 2, 3, 1)= 2.887255491422e-05  HH( 2, 3, 2)= 6.048518418310e-05
HH( 2, 3, 3)= 4.164913019080e-06  HH( 3, 0, 0)= 1.174317580568e-06
HH( 3, 0, 1)= 4.164913019080e-06  HH( 3, 0, 2)= 1.722332451500e-07
HH( 3, 1, 0)= 4.164913019080e-06  HH( 3, 1, 1)= 6.048518418310e-05
HH( 3, 1, 2)= 2.887255491422e-05  HH( 3, 1, 3)= 1.722332451500e-07
HH( 3, 2, 0)= 1.722332451500e-07  HH( 3, 2, 1)= 2.887255491422e-05
HH( 3, 2, 2)= 6.048518418310e-05  HH( 3, 2, 3)= 4.164913019080e-06
HH( 3, 3, 1)= 1.722332451500e-07  HH( 3, 3, 2)= 4.164913019080e-06
HH( 3, 3, 3)= 1.174317580568e-06
=======================================
```

Acknowledgements: I am deeply grateful to Wolfgang Dahmen for his many valuable comments on the theory, and I also wish to thank Folkmar Bornemann, Andreas Hohmann and Manfred Schulte for helping with an earlier version of the program and Titus Barsch for pointing out a programming error.

References

[Be] G. Beylkin, *On the representation of operators in bases of compactly supported wavelets*, Siam J. Numer. Anal. 6, 1992, 1716-1740.

[BH] C. de Boor and K. Höllig, *B-Splines from parallelepipeds*, J. Analyse Math. 42, 1982, 99-115.

[CDM] A.S. Cavaretta, W. Dahmen and C. A. Micchelli, *Stationary Subdivision*, Memoirs of Amer. Math. Soc., Vol. 93, # 453, 1991.

[CDF] A. Cohen, I. Daubechies and J.-C. Feauveau, *Biorthogonal bases of compactly supported wavelets*, Comm. Pure Appl. Math. 45, 1992, 485-560.

[DLy] M. Dæhlen and T. Lyche, *Box splines and applications*, in: Geometric Modelling: Methods and Applications, H. Hagen, D. Roller (eds.), Springer, 1991, 35-94.

[D] W. Dahmen, private communication.

[DM1] W. Dahmen and C.A. Micchelli, *Recent progress in multivariate splines*, in: Approximation Theory IV, C.K. Chui, L.L. Schumaker, J.D. Ward (eds.), Academic Press, New York, 1983, 27-121.

[DM2] W. Dahmen and C. A. Micchelli, *Translates of multivariate splines*, Linear Alg. Appl. 52/53, 1983, 217-234.

[DM3] W. Dahmen and C.A. Micchelli, *Using the refinement equation for evaluating integrals of wavelets*, Siam J. Numer. Anal. 30, 1993, 507-537.

[Dau1] I. Daubechies, *Orthonormal bases of wavelets with compact support*, Comm. Pure and Appl. Math. 41, 1987, 909-996.

[Dau2] I. Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Maths., SIAM, 1992.

[K1] A. Kunoth, *On the fast evaluation of integrals of refinable functions*, in: Wavelets, Images, and Surface Fitting, P. J. Laurent, A. Le Méhauté, and L. L. Schumaker (eds.), AKPeters, Boston, 1994, 327-334.

[K2] A. Kunoth, *Multilevel Preconditioning*, Verlag Shaker, Aachen, Germany, January 1994.

[K3] A. Kunoth, *Computing integrals of refinable functions — Documentation of the program — Version 1.0*, SINTEF Technical Report STF33 A94045, SINTEF, Oslo, Norway, September 1994.

[LRT] A. Latto, H.L. Resnikoff and E. Tenenbaum, *The evaluation of connection coefficients of compactly supported wavelets*, in: Proceedings of the USA-French Workshop on Wavelets and Turbulence, Princeton University, 1991.