Annales Univ. Sci. Budapest., Sect. Comp. 36 (2012) 241–254

RATIONAL FFT IMPLEMENTATION IN MATLAB

Levente Lócsi (Budapest, Hungary)

Communicated by Ferenc Schipp

(Received December 20, 2011; revised January 25, 2012;
accepted February 14, 2012)

Abstract. In this paper we present a MATLAB implementation of a method related to the fast calculation of Fourier coefficients with respect to a product system of rational complex functions. The elements of the product system used here are defined as compositions of two-factor Blaschke products. We provide new tools for the visualization of the function systems in question, elaborate the outlined algorithms in [8], describe the methods used for numerical calculation. The work presented here can be applied in signal processing and control theory, future applications include the analysis of ECG signals. The toolbox is available at:
`http://numanal.inf.elte.hu/~locsi/fftratsys/`

1. Introduction

In signal processing and in many areas of applied mathematics the fast calculation of Fourier coefficients via the so-called FFT (Fast Fourier Transform) is of great importance. Many modern applications would have been unthinkable without this method, first implemented in 1965 [3]. The traditional approach uses the complex trigonometric system $\exp ikx = \cos kx + i \sin kx$ $(k \in \mathbb{Z})$ as basis, sampled uniformly in $[0, 2\pi)$.

Key words and phrases: Blaschke products, FFT algorithms, MATLAB.
2010 Mathematics Subject Classification: 65T50, 30J10.
The Research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1./B-09/1/KMR-2010-0003).

In the recent decades rational function systems and non-uniform sampling has became more and more widely used and also mathematically studied. E.g. the Laguerre and Kautz systems are used in system identification [9, 10], and the non-equidistant discretization generated by similar systems have also been studied [6, 7].

In [8] F. Schipp outlines the concept of FFT for these rational systems. In what follows, we basically rely on this work of his, elaborating, implementing and demonstrating mainly all of the closely related theoretical work.

In Section 2 we recall the basic facts of the mathematical background: product systems, FFT algorithms and dyadic rational systems. In Sections 3 and 4 our implementation is presented through a short documentation and by two examples using the created MATLAB programs. Finally Section 5 exhibits some possible areas of further work and research.

For testing purposes in this work the three-factor Blaschke product

$$f(z) := \prod_{i=1}^{3} B_{a_i}(z) \quad (z \in \mathbb{C}), \text{ with}$$
$$a_1 = 0.3; a_2 = 0.8i; a_3 = -0.4 + 0.5i$$

will be used. (See Definition 2.2.) This function can be seen on Figure 1 as plotted by the command `plotd` described in Section 3. The zeros are also marked.

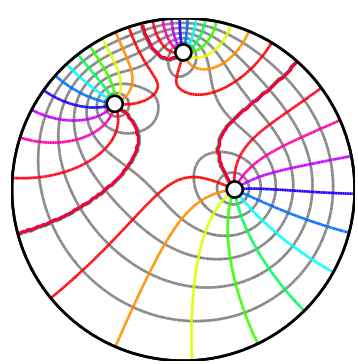


Figure 1. A three-factor Blaschke product on the complex unit disc.

2. FFT for rational product systems

In this section we recall the basic definitions and theorems regarding product systems, FFT algorithms and dyadic rational systems, which will be used later. We will restrict ourselves to the finite dimensional vector space $X := \mathbb{C}^N$ for some $N = 2^n$ ($n \in \mathbb{N}^+$), but until the point when we turn our attention to discrete calculations, everything could be analogously defined in the continuous case.

2.1. Product systems

For the definition of product systems let us fix a collection of function systems

$$\Phi_k := \{1,\ \varphi_k\} \qquad (0 \le k < n), \tag{2.1}$$

where each φ_k is an arbitrary complex valued function. Then with the unique dyadic expansion of $m \in \mathbb{N},\ 0 \le m < N$:

$$m = \sum_{k=0}^{n-1} m_k 2^k, \tag{2.2}$$

we write the following definition.

Definition 2.1. The product system $\Psi = \{\, \psi_m \,:\, 0 \le m < N \,\}$ of systems Φ_k $(0 \le k < n)$ is defined by its elements as

$$\psi_m := \prod_{k=0}^{n-1} \varphi_k^{m_k}. \tag{2.3}$$

Well-known examples for product systems are the Walsh system, the Walsh–Paley system, but also the trigonometric system can be considered as a product system.

Under some specific conditions FFT algorithms can be used to calculate the Fourier coefficients with respect to a product system. Namely, a sufficient condition is: if Φ_k is a sequence of *adapted conditionally orthonormal systems*. Thus Ψ is an orthonormal system. (See [8].)

2.2. FFT algorithms

The traditional FFT algorithms calculate the *discrete Fourier transform (DFT)* of a vector (or 'signal') $x \in X$ with respect to the first N elements of

the *discrete trigonometric system* defined as

$$\epsilon_m(t) = \exp(imt) \qquad (0 \le m < N,\, t = 2\pi l/N, 0 \le l < N)\,.$$

In the simplest case $N = 2^n$ ($n \in \mathbb{N}^+$) is assumed. Note that ϵ_m can be considered as the function z^m sampled uniformly at the points

$$T_N := \{\, \exp(it) \,:\, t = 2\pi l/N, 0 \le l < N \,\}$$

on the unit circle.

In this case, given a vector $x \in X$, the discrete Fourier coefficients are

$$c_m = [x, \epsilon_m]_N \qquad (0 \le m < N),$$

where the discrete scalar product is defined as

$$[f, g]_N := \frac{1}{N} \sum_{t \in T_N} f(t) \cdot \overline{g}(t) \qquad (f, g \in X). \tag{2.4}$$

They can be calculated by using the FFT (see Algorithm 1): $\mathcal{O}(n \cdot 2^n)$ operations are required compared to the naive DFT calculation with $\mathcal{O}(2^n \cdot 2^n)$ operations, i.e. $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$. An FFT implementation is also very efficient considering disk space usage. (A unit of calculation in the FFT is usually called a 'butterfly'.) See also [3, 5].

Algorithm 1: The traditional (power-of-two, DIF) FFT algorithm.

Input: $\mathsf{x} \in X = \mathbb{C}^N$ ($N = 2^n$).
Output: the DFT of x in bitreversed order.

```
1  parts ← 1;
2  partwidth ← N;
3  for phase ← 1 to n do
4      for part ← 0 to parts − 1 do
5          for butterfly ← 0 to partwidth/2 − 1 do
6              i ← parts · partwidth + butterfly + 1;
7              j ← i + partwidth/2;
8              a ← x [ i ];
9              b ← x [ j ];
10             x [ i ] ← (a + b)/2;
11             x [ j ] ← (a − b) · exp(−2πi · butterfly/partwidth)/2;
12     parts ← parts · 2;
13     partwidth ← partwidth/2;
```

Furthermore, each $x \in X$ can be reconstructed from its Fourier coefficients:

$$x(t) = \sum_{m=0}^{N-1} c_m \cdot \epsilon_m(t) \qquad (t \in T_N).$$

2.3. Dyadic rational systems

In the construction of the rational FFT algorithm, the Blaschke functions play an important role. Let us denote the complex unit disc by $\mathbb{D} := := \{ z \in \mathbb{C} : |z| < 1 \}$, and the complex unit circle or torus by $\mathbb{T} := \{z \in \mathbb{C} : : |z| = 1\}$.

Definition 2.2. Blaschke functions are complex valued functions of one complex variable of the form

$$B_a(z) = \frac{z - a}{1 - \overline{a}z} \qquad (z \in \mathbb{C} \setminus \{ 1/\overline{a} \}),$$

with parameter $a \in \mathbb{D}$. A Blaschke product is the product of finitely many Blaschke functions.

Blaschke functions have many interesting properties, e.g. they are bijections on both $\mathbb{D}$ and $\mathbb{T}$. Furthermore the equation

$$\prod_{j=1}^{n} B_{a_j}(z) = \tau \qquad (a_j \in \mathbb{D}, 1 \le j \le n, \tau \in \mathbb{T})$$

has exactly n solutions (see [8]): i.e. a Blaschke product of order n is an n-fold map on $\mathbb{T}$. Also, note that with the choice $a = 0$, $B_0 = z$, the products $B_0^m = z^m$ $(m \in \mathbb{N})$ on $\mathbb{T}$ would produce the trigonometric system.

We will use two-factor (or second order) Blaschke products of a special form:

$$A_k := B_{a_k} B_{-a_k} \qquad (a_k \in \mathbb{D}, 1 \le k \le n).$$

Set

$$A_0 := B_{a_0} \qquad (a_0 \in \mathbb{D}).$$

With these A_k $(0 \le k \le n)$ functions we now define the *generators of the product system* as in (2.1):

$$\begin{aligned} \varphi_0 &:= A_0, \\ \varphi_k &:= A_k \circ A_{k-1} \circ \cdots \circ A_1 \circ A_0 \\ &= A_k \circ \varphi_{k-1} \qquad\qquad (0 < k \le n). \end{aligned}$$

Then we can form a product system in exactly the same way as described in Section 2.1 by using (2.2) and (2.3).

Making use of the fact that the composition of Blaschke products is again a Blaschke product (with an additional phase factor), one finds, that

- φ_0 is a Blaschke function,
- φ_k is a Blaschke product of order 2^k $(1 \le k \le n)$,
- ψ_0 is the constant one function,
- ψ_m is a Blaschke product of order m $(1 \le m < N = 2^n)$.

Note that A_n is not present in any way in the product system Ψ. Indeed, A_n only appears in φ_n which is a Blaschke product of order 2^n, and the functions ψ_m are defined only for $0 \le m < 2^n$.

On the other hand φ_n has a role in the definition of the discretization points on $\mathbb{T}$. Namely, choosing

$$T_N := \varphi_n^{-1}(\tau) = \{\, z \in \mathbb{T} : \varphi_n(z) = \tau \,\} \qquad (N = 2^n) \tag{2.5}$$

with an arbitrary $\tau \in \mathbb{T}$, the functions φ_k sampled at the points of T_N give rise to a sequence of *adapted conditionally orthonormal systems*. Thus the functions ψ_m form an orthonormal system with respect to the scalar product in (2.4). Furthermore one can apply FFT to calculate the Fourier coefficients.

In Algorithm 2 we indicated the only difference in the rational FFT compared to the trigonometric algorithm. In Line 11 one should use the values given by the generating functions instead of the exponential expression, supposing that Phi [m, k] contains the sampled values of φ_k in the mth point of T_N.

Algorithm 2: The rational FFT algorithm. (Cf. Algorithm 1.)

```
10 ...
11 x [ j ] ← (a − b) · conj(Phi [ butterfly + 1, phase ]) /2;
12 ...
```

3. Implementation

In this section we describe our MATLAB implementation of the theory summarized in Section 2. Our goal is to give an overview of the implementation

with a short documentation of the programs, and also to provide some examples. We follow the guidelines of *reproducible research* (see [2]). The created programs can be downloaded from

`http://numanal.inf.elte.hu/~locsi/fftratsys/`.

3.1. General considerations

We have chosen the Mathworks MATLAB environment to realize the ideas presented above, because it is convenient for visualization, and it has a wide range of programming tools. For a comprehensive guide to MATLAB see [1] or [11].

MATLAB also supports *object oriented programming,* which enabled us to deal with Blaschke functions, Blaschke products and Blaschke compositions in an elegant and type safe way.

In order to use the programs, please download them from the above link, extract them in your current MATLAB working directory, or make sure some other way, that they are available on the `path`.

3.2. Short documentation of programs

In this section we enumerate the programs (`m` files) created. A few words and little examples are attached to each of these.

They are divided in four groups.

3.2.1. Classes

`blaschke`. This class implements the Blaschke functions. One may create an instance of this class (i.e. a Blaschke function) in the following way:

```
b = blaschke(0.5);
```

supplying the parameter a as argument. Then one can calculate the function values at given points (say 1 and i), and also the *function handle* is available if needed:

```
b.values([1 1i])
bh = b.handle();
```

The class also supports some plotting routines, and it is robust (one can not create Blaschke functions with invalid parameters, e.g. $a \notin \mathbb{D}$).

`blaschkep`. This is the class implementing Blaschke products. Naturally one can create Blaschke products (but only valid ones) by supplying the desired parameters, and the calculation of function values, function handles and plotting routines are also available. For instance one can create a Blaschke product of order 3 (as seen on Figure 1) with the following command:

```
bp = blaschkep([0.3 0.8*1i -0.4+0.5*1i]);
```

`blaschkec`. This class allows the construction of *Blaschke compositions,* i.e. compositions of Blaschke products, as described in Section 2.3. In order to create a Blaschke composition object one has to supply a row vector of Blaschke products. E.g.:

```
b1 = blaschkep([-0.2 0.2]);
b2 = blaschkep([0.1 0.5]*1i);
bc = blaschkec([b1 b2]);
```

In addition to similar methods to those of previous classes (values, handles, plotting), this class also implements methods for converting a Blaschke composition to a Blaschke product and for finding inverse images of the points of the torus—needed to realize (2.5). One can find that both of the above tasks can be reduced to recursively solving quadratic equations (when using two-factor Blaschke products). These methods are demonstrated in Section 4.

3.2.2. Functions with mathematical objectives

Apart from the member methods of the above introduced classes there are some other functions serving the calculation of rational FFTs. We present them below, but their use can be only examined in Section 4.

`prodsysgen`. Given a sequence of Blaschke products this function creates the *generators of the product system* as Blaschke compositions as seen in Section 2.3.

`prodsys`. This function assembles the product system from its generators as seen in (2.3). (Note again that φ_n will not be used.)

`samples`. The FFT works on discrete points, so we have to sample all of our functions at hand at given points of $\mathbb{T}$. By sampling the functions in Ψ we will get the matrix for the calculation of the DFT by definition. By sampling only the φ_k $(1 \leq k < n)$ functions, we get the values needed in the FFT algorithm. Of course the signal to be processed also has to be sampled.

`rfft`. The implementation of the rational FFT.

3.2.3. Visualization methods

We have created some methods primarily for visualization purposes. By means of them the verification of the aforementioned programs is also much easier.

`plotd.` The name of this command stands for *plot on the disc.* Our functions are most interesting on $\mathbb{D}$, so we have created this function to give us an idea of the behaviour of the functions here—both magnitude and phase. We kept in mind also, that such a visualization is desired which also looks fine in monochrome printed materials. So this program uses contour lines to display the change in magnitude and phase of the function values. This is achieved by applying the MATLAB command `contour` twice.[1] E.g. see Figure 1 in Section 1 (there we also added the zeros), or similar figures later in the paper.

The lines quasi concentric to the unit circle express the magnitude of the function values and the lines crossing at the zeros describe the change in the phase of the function values.

`plotdm.` Plots multiple functions on the disc, waiting 1 second between each `plotd` plot. This method is useful for displaying the generators of the product system, or the product system itself.

`plotdp.` Plots points of the disc. (Also displays points outside $\mathbb{D}$, but the unit circle is always shown.)

Figure 1 was created by the code fragment:

```
as = [0.3 0.8*1i -0.4+0.5*1i];
rb = blaschkep(as);
plotd(rb)
hold on
plotdp(as)
```

`plott.` This routine plots the function values on the *torus.* Both real and imaginary components.

`plottm.` Similar to `plotdm`, but with `plott`. Plots multiple functions on $\mathbb{T}$.

`stemt.` Visualizes non-uniformly sampled function values on $\mathbb{T}$ using the built-in MATLAB command `stem.` It is convenient for the display of both sampled signals and sampled basis functions. The torus is presented as the real interval $[0, 2\pi)$.

3.2.4. Other tools

The programs listed in this section are also used in this context.

[1] A minor drawback is the slight visibility of the nonexistent 'jump' in phase between 2π and 0.

`randd`. This command produces pseudorandom complex numbers inside $\mathbb{D}$. It is built on the basic MATLAB command `rand`.

`bitreversal`. The FFT algorithm produces its output in bitreversed order, so another small step is needed to rearrange the values in 'normal' order: this is done by the `bitreversal` method. It uses a quick algorithm to build the bitreversed indices (cf. [4]).

`rffttest`. Basically goes through all the commands mentioned here: from creating a product system using Blaschke compositions, through the visualization of the generated system, to the calculation and validation of the rational FFT.

`test_asprod`. This script helped in the testing of the `blaschkec` class' method for converting a Blaschke composition to a Blaschke product.

4. Examples

Now we will demonstrate how the created tools work together to realize a rational FFT. For a more detailed demonstration please run the `rffttest` script. We provide two examples, which differ only in the initial choice of parameters a_0 and a_1. The first example will exhibit a truly rational system (and FFT algorithm) with non-uniform sampling points, while the second example will result in the trigonometric system with uniform sampling points and the traditional FFT.

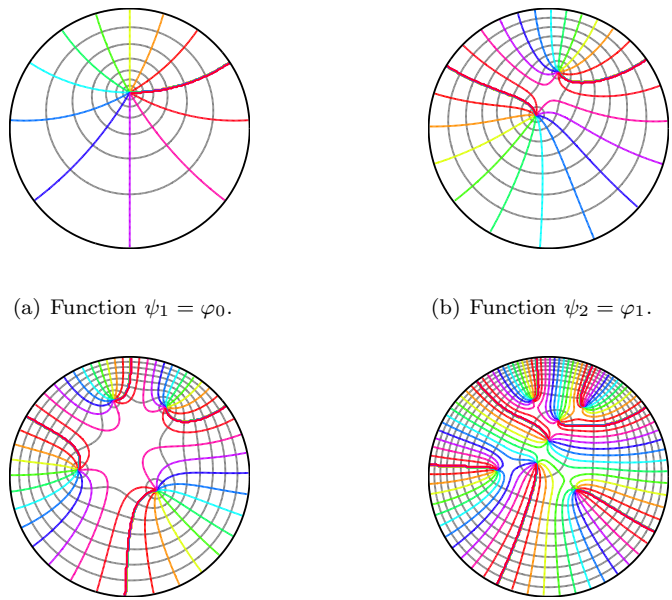
4.1. A truly rational system

In this example we will create a product system generated by Blaschke compositions and calculate the FFT of our test signal with respect to this system. Consider the following code:

```
a0 = 0.3*1i;
a1 = 0.1 + 0.2*1i;
b0 = blaschkep(a0);
b1 = blaschkep([a1 -a1]);
psg = prodsysgen([b0 b1 b1 b1 b1 b1]);
ps = prodsys(psg);
```

Now we can visualize our system (excluding the constant 1 function), e.g. this way (Figure 2 shows some elements of the generated set of 32 functions):

```
plotdm(ps(2:end))
```



(a) Function $\psi_1 = \varphi_0$.

(b) Function $\psi_2 = \varphi_1$.

(c) Function $\psi_4 = \varphi_2$.

(d) Function $\psi_7 = \varphi_2\varphi_1\varphi_0$.

Figure 2. Some elements of the rational system.

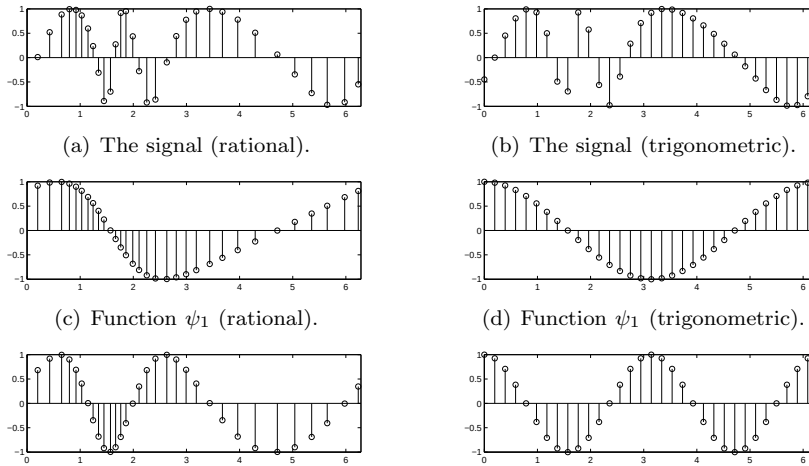
We need to calculate the discretization points $T_N = \varphi_n^{-1}(\tau)$ with $\tau = 1$ being the default parameter, as seen in (2.5). Next we sample the signal (see `rb` above) and the generators. (One can also plot the signal, see Figure 3.)

```
ip = psg(end).inverset();
sig = samples(ip,rb);
G = samples(ip,psg);
stemt(ip,sig)
```

And finally one can apply the rational FFT:

```
c = rfft(sig,G)
```

As a simple validation of the results, one might want to compare these coefficients to the result of the naive DFT calculation as a matrix multiplication using e.g. the maximum norm:



(a) The signal (rational). (b) The signal (trigonometric).

(c) Function ψ_1 (rational). (d) Function ψ_1 (trigonometric).

(e) Function ψ_2 (rational). (f) Function ψ_2 (trigonometric).

Figure 3. The signal and some elements of the rational system sampled at the discretization points on $\mathbb{T}$ in the general rational case (left), and in the special trigonometric case (right). Only real parts are shown.

```
F = samples(ip,ps);
cm = F' * sig / length(sig);
norm(c-cm,'inf')
```

It is correct up to numerical precision.

4.2. The trigonometric system as a special case

The same can be done for another set of parameters. Let us now initialize

```
a0 = 0;
a1 = 0;
```

and do the rest again. We shall arrive at the trigonometric system, uniform sampling points, the traditional FFT. We do not include the plots in this case, but the reader is encouraged to try the MATLAB functions him- or herself and examine the results.

Figure 3 shows 6 `stemt` plots. One can compare both the sampled signal and some elements of the used basis in the general rational case, and in the trigonometric case. Observe, that with a well-chosen set of parameters, a non-uniformly sampled signal might provide more information than a uniformly

sampled one: the signal changes more rapidly in the first half. Also in this case a suitable orthogonal system is needed, which is now provided by our tools.

5. Further work

This implementation can be extended in many ways. Without the constraint of using second order Blaschke products with parameters a and $-a$, one might want to use arbitrary a_1 and a_2. This way the product system would not be orthogonal: biorthogonal systems should be used, but the FFT could still be applied. One might also want to drop the constraint of using only *second* order Blaschke products.

How to choose the parameters for the Blaschke products, if a given set of zeros are desired for the product system is also an open mathematical problem.

The method presented here is to be applied in the case of processing ECG signals.

6. Summary

In this paper we presented our implementation of the topic related to FFT algorithms for dyadic rational product systems defined by Blaschke products.

We provided an easy to use, robust MATLAB toolkit supporting the study of this kind of systems and algorithms. These tools can be freely downloaded. We provided examples and a short documentation of the created programs.

Hopefully these programs and visualization methods will help the future study and maybe teaching of the related mathematical background.

Acknowledgements. The author would like to thank Prof. Ferenc Schipp for his motivating and insistent help in this research area.

A preliminary version of this work was presented on the Conference on Dyadic Analysis and Related Fields with Applications held in September 2011 at the College of Nyíregyháza, thanks to the organizers.

The author also wishes to thank the organizers of the 9th Joint Conference on Mathematics and Computer Science (MaCS 2012).

References

[1] **Attaway, S.,** *Matlab,* Butterworth–Heinemann, Waltham and Oxford, 2011.

[2] **Buckheit, J.B. and D.L. Donoho,** WaveLab and Reproducible Research, `http://www-stat.stanford.edu/~wavelab/`

[3] **Cooley, J.W. and J.W. Tukey,** An algorithm for the machine calculation of complex Fourier series, *Mathematics of Computation,* **19** (1965), 297–301.

[4] **Drouiche, K.,** A new efficient computational algorithm for bit reversal mapping, *IEEE Transactions on Signal Processing,* **49** (2001), 251–254.

[5] **Elliott, D.F. and K.R. Rao,** *Fast Transforms,* Academic Press, London, 1982.

[6] **Lócsi, L.,** Calculating non-equidistand discretizations generated by Blaschke products, *Acta Cybernetica*, **20** (2011), 111–123.

[7] **Pap, M. and F. Schipp,** Malmquist–Takenaka systems and equilibrium conditions, *Mathematica Pannonica,* **12** (2001), 185–194.

[8] **Schipp, F.,** Fast Fourier transform for rational systems, *Mathematica Pannonica,* **13** (2002), 265–275.

[9] **Schipp, F. and J. Bokor,** Rational bases generated by Blaschke product systems, *Proceedings of the 13th IFAC Symposium on System Identification* (Rotterdam, 2003), 1351–1356.

[10] **Schipp, F. and A. Soumelidis,** On the Fourier coefficients with respect to the discrete Laguerre system, *Annales Univ. Sci. Budapest, Sectio Computatorica,* **34** (2011), 223–233.

[11] **Stoyan, G.,** *Matlab,* Typotex Kiadó, Budapest, 2008 (in hungarian).

L. Lócsi
Department of Numerical Analysis
Faculty of Informatics
Eötvös Loránd University
Pázmány P. sétány 1/C.
H-1117 Budapest, Hungary
`locsi@inf.elte.hu`