Computations of the Symmetric Cosine Transform Using Forsythe and Clenshaw's Recurrence Formulae

Maurice F. Aburdene[a], Hoang M. Le[a], John E. Dorband[b]
[a]Bucknell University, Lewisburg, PA 178371
[b]NASA Goddard Space Flight Center, Greenbelt, MD 20771

ABSTRACT

The discrete cosine transform (DCT) is commonly used in signal processing, image processing, communication systems and control systems. We use two methods based on the algorithms of Clenshaw and Forsyth to compute the recursive DCT in parallel. The symmetrical discrete cosine transform (SCT) is computed first and then it can be used as an intermediate tool to compute other forms of the DCT. The advantage of the SCT is that both the forward SCT and its inverse can be computed by the same method and hardware implementation. Although Clenshaw's algorithm is the more efficient in computational complexity, it is not necessarily the more accurate one. The computational accuracy of these algorithms is discussed. In addition, the front-to-back forms of Clenshaw and Forsyth's algorithms are implemented in aCe C , a parallel programming language.

Keywords: Discrete cosine transforms, symmetrical cosine transform, Clenshaw's recurrence, Forsyth's recurrence, parallel computing

1. INTRODUCTION

The discrete cosine transform (DCT) is commonly used in signal processing, image processing, communication systems and control systems. This paper will use two methods to compute the recursive DCT in parallel. These two methods are based on the recursive algorithms of Forsythe [1] and Clenshaw [2, 3,4] for the evaluation of a Chebyshev sum of terms. The essential computation in all these transforms is of the form

$$Y(k) = \sum_{n=0}^{N} y(n)\cos(\frac{kn\pi}{N}) \quad \text{for} \quad k = 0,1,...,N . \tag{1}$$

Initially, we will compute the symmetrical discrete cosine transform (SCT) [5,6] using both methods and then show how to use the SCT as an intermediate tool to compute the DCT-II. The SCT of N+1 data points *x*(0), x(1) , ..., x(N-1), x(N) is given by

$$X(m) = \sqrt{\frac{2}{N}} k_m \sum_{n=0}^{N} k_n x(n)\cos(\frac{mn\pi}{N}) \quad \text{for} \quad m = 0,1,...,N . \tag{2}$$

The inverse SCT is given by

$$x(n) = \sqrt{\frac{2}{N}} k_n \sum_{m=0}^{N} k_m X(m)\cos(\frac{mn\pi}{N}) \tag{3}$$

Image and Video Communications and Processing 2005, edited by Amir Said, John G. Apostolopoulos, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 5685 © 2005 SPIE and IS&T · 0277-786X/05/$15

where

$k_m = \frac{1}{\sqrt{2}}$ for m = 0, N and 1 otherwise

$k_n = \frac{1}{\sqrt{2}}$ for n = 0, N and 1 otherwise

As can be seen from equations (2) and (3), both the forward SCT and its inverse transform can be computed by the same hardware implementation.

The DCT-II and its inverse are given by

$$Y(k) = \frac{2}{N}\gamma_k \sum_{n=0}^{N-1} y(n)\cos[k(n+\frac{1}{2})\frac{\pi}{N}] \quad (k = 0, 1, ..., N-1) \tag{4}$$

$$y(n) = \sum_{k=0}^{N-1} Y(k)\cos[k(n+\frac{1}{2})\frac{\pi}{N}] \quad (n = 0, 1, ..., N-1) \tag{5}$$

with $\gamma_0 = \frac{1}{2}$ and $\gamma_k = 1$ for $k \neq 0$

To convert the DCT-II form into the SCT form, we will use trigonometric identities to convert equation (2) into the form of equation (1)[2,6]. A new set of coefficients, $\tilde{y}(0)$, $\tilde{y}(1)$,…, $\tilde{y}(N-1)$, $\tilde{y}(N)$, is computed from $y(0)$, $y(1)$,…, $y(N-1)$ and these new values are used in equation (1).

The identity 2cos(a)cos(b) = cos(a+b) + cos(a-b) will be used in the following steps:

$$Y(k) = \frac{2}{N}\gamma_k \sum_{n=0}^{N-1} y(n)\cos[k(n+\frac{1}{2})\frac{\pi}{N}]$$

$$= \frac{2}{N}\gamma_k \sum_{n=0}^{N-1} \frac{y(n)}{\cos(k\frac{\pi}{2N})}\cos(k\frac{\pi}{2N})\cos[k(n+\frac{1}{2})\frac{\pi}{N}]$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \sum_{n=0}^{N-1} y(n)\{2\cos(k\frac{\pi}{2N})\cos[k(n+\frac{1}{2})\frac{\pi}{N}]\}$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \sum_{n=0}^{N-1} y(n)[\cos(k(n+1)\frac{\pi}{N}) + \cos(kn\frac{\pi}{N})]$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \{y(0) + \sum_{n=1}^{N-1}(y(n-1)+y(n))\cos(kn\frac{\pi}{N}) + y(N-1)\cos(kN\frac{\pi}{N})\}$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \sum_{n=0}^{N} \tilde{y}(n)\cos(kn\frac{\pi}{N}) \tag{6}$$

where $\tilde{y}(0) = y(0)$, $\tilde{y}(k) = y(k-1) + y(k)$ for $1 \le k \le N-1$, and $\tilde{y}(N) = y(N-1)$.

Y(k) in equation (6) is now in the form of the equations (1) and (2). The efficiency of the computation can be enhanced further if we use the property $\cos(x) = \cos(k\pi - x)$ when k is even and $\cos(x) = -\cos(k\pi - x)$ when k is odd.

Since $\cos(kn\frac{\pi}{N}) = \pm\cos(k(N-n)\frac{\pi}{N})$ for $n = 0,1,...,\frac{N}{2}$, we have

$\tilde{y}(n)\cos(kn\frac{\pi}{N}) + \tilde{y}(N-n)\cos(k(N-n)\frac{\pi}{N}) = (\tilde{y}(n) \pm \tilde{y}(N-n))\cos(kn\frac{\pi}{N})$ for $n = 0,1,...,\frac{N}{2}$. Therefore, we can combine the n^{th} element with the $(N+1-n)^{th}$ element to reduce the number of terms in equation (6) from $(N+1)$ to $\frac{(N+1)}{2}$.

The forward DCT now can be rewritten as:

$$Y(k) = \frac{2}{N}\gamma_k \sum_{n=0}^{N-1} y(n)\cos[k(n+\frac{1}{2})\frac{\pi}{N}]$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \sum_{n=0}^{\frac{N-1}{2}} (\tilde{y}(n) + \tilde{y}(N-n))\cos(kn\frac{\pi}{N}) \text{ if N is odd} \tag{7}$$

and

$$Y(k) = \frac{2}{N}\gamma_k \sum_{n=0}^{N-1} y(n)\cos[k(n+\frac{1}{2})\frac{\pi}{N}]$$

$$= \frac{1}{N\cos(k\frac{\pi}{2N})}\gamma_k \{(\sum_{n=0}^{\frac{N-2}{2}} (\tilde{y}(n) + \tilde{y}(N-n))\cos(kn\frac{\pi}{N})) + \tilde{y}(\frac{N}{2})\cos(k\frac{\pi}{2})\} \text{ if N is even} \tag{8}$$

This symmetry of the SCT leads to a more efficient computation since the computation for the cosine function or the multiplication is more expensive than the computation for additions.

A similar analysis for the computation of the IDCT requires one less summation in equation (6) than the forward DCT and will not be repeated here.

Since $T_0(x) = \cos(0) = 1$, $T_1(x) = \cos(x)$ and $T_k(x) = \cos(kx) = 2\cos(x)\cos((k-1)x) - \cos((k-2)x)$, we have $T_n(x) = \cos n(nx)$ for all n. $T_n(x)$ is the n^{th} Chebyshev polynomial, the SCT can be seen as a particular value of a Chebyshev sum, $\sum_{k=0}^{n} b_k T_k(x)$ [1]. Hence, any algorithm used to compute the value of the Chebyshev sum can be applied to evaluate the SCT. Clenshaw in 1955 and Forsythe in 1957 developed algorithms to compute the sum, $\sum_{k=0}^{n} b_k T_k(x)$.

2. ALGORITHM FOR EVALUATING POLYNOMIALS

Clenshaw and Forsyth presented methods for the evaluation of a polynomial, $P_N(x)$, represented by

$P_n(x) = \sum_{k=0}^{n} b_k T_k(x)$, where $T_k(x)$ is the Chebyshev polynomial, defined by the recurrence

$T_0(x) = 1$

$T_1(x) = x$

$T_k(x) = 2x\,T_{k-1}(x) - T_{k-2}(x)$

for k = 2,3, …n.

Clenshaw's front–to-back (forward) algorithm is shown in Figure 3 and Forsyth's algorithm is shown in Figure 4.

$$Y := 2 \times x$$
$$V[0] := b[0]$$
$$V[1] = Y \times V[0] + b[1]$$
$$V[k] := [Y \times V[k-1] - V[k-2] + b[k]](k = 2 \rightarrow n)$$
$$polyvalue = V[n-1] \times T_{n-1}(x) + (b[n] - V[n-2]) \times T_n(x)$$

Figure 1: Clenshaw's Algorithm [3]

$$Y = 2 \times x$$
$$T[0] := 1, V[0] := b[0]$$
$$T[1] := x, V[1] := V[0] + b[1] \times T[1]$$
$$\begin{bmatrix} T[k] := Y \times T[k-1] - T[k-2] \\ V[k] := V[k-1] + b[k] \times T[k] \end{bmatrix} (k = 2 \rightarrow n)$$
$$polyvalue := V[n]$$

Figure 2: Forsythe's Algorithm [1]

The recursive filter structures of the Clenshaw and Forsythe algorithms are shown in Figures 3 and 4. These algorithms can be easily implemented in parallel. In Appendix A, we show a parallel programming language program for the computation of Clenshaw and Forsythe's algorithms. The program is implemented aCe C, a parallel programming language [6].

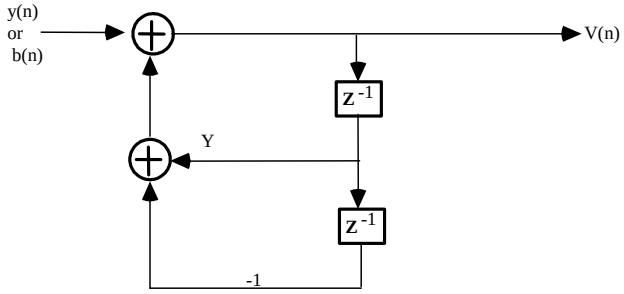


Figure 3: Recursive structure of Clenshaw's algorithm

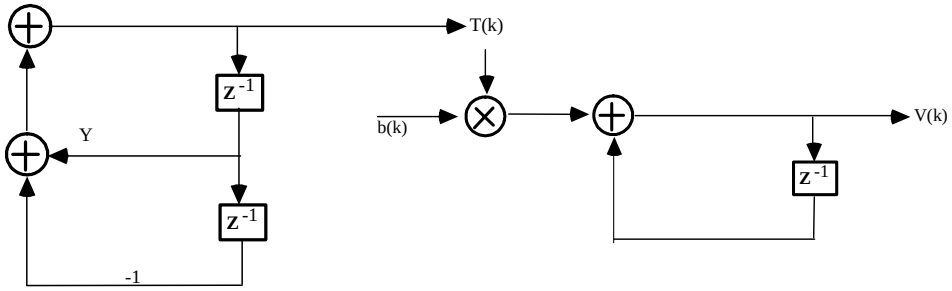


Figure 4: Recursive structure of Forsythe's algorithm

3. NUMERICAL ACCURACY COMPARISONS

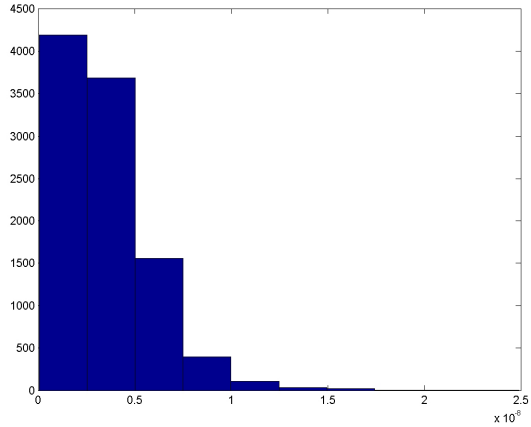
Although Clenshaw's algorithm is more efficient than Forsythe's algorithm in terms of computational complexity, it is not necessarily the more accurate one. The accuracy of these algorithms depends on the input data. The computation of the direct implementation of equation (2) was done in double precision and was used as the reference for numerical accuracy. The mean square errors of the transform components (MSE) for both Forsyth's and Clenshaw's methods were calculated using a C program. Both Forsythe's and Clenshaw's algorithms for the computation of the SCT were done single precision.

Trials (10,000) were run on input data sequences of length 8, 16, 32, 64, 128, 512 and 1024 using uniformly distributed random integer values (0-255). The cosine values were pre-computed to speed up the program's run -time. Table 1 shows the mean square error of the transform components for both methods. As can be seen, our results show that Forsythe's method has a lower MSE except for the trials with sequence length 16.

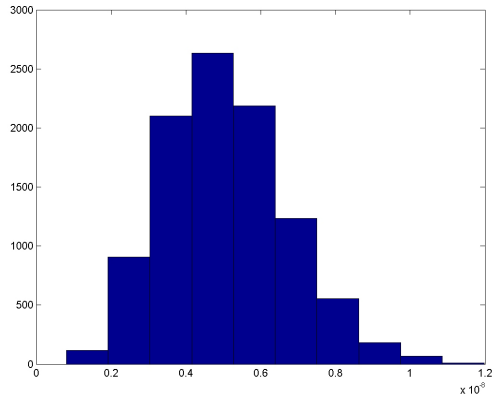
Table 1: Mean Square Errors for random input sequences.

Number of data points	Clenshaw's error	Forsythe's error
8	5.5421e-010	3.6397e-010
16	3.4568e-009	5.0782e-009
32	4.0893e-009	2.1883e-009
64	1.0247e-007	1.3114e-008
128	4.4432e-008	2.9541e-008
256	4.6080e-006	3.9177e-006
512	1.4232e-005	1.2274e-005
1024	2.4604e-005	1.8927e-005

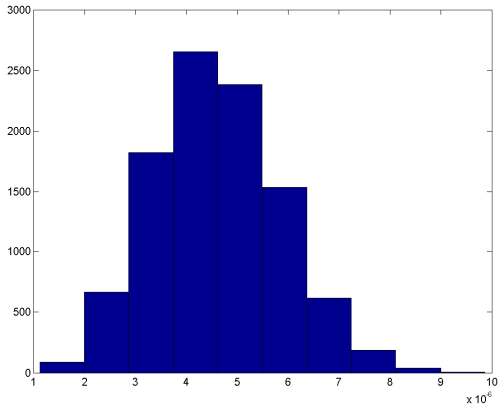
Figure 5 shows the histograms of MSE for Clenshaw's method and Forsythe's methods using 16, 256, and 1024 data points.



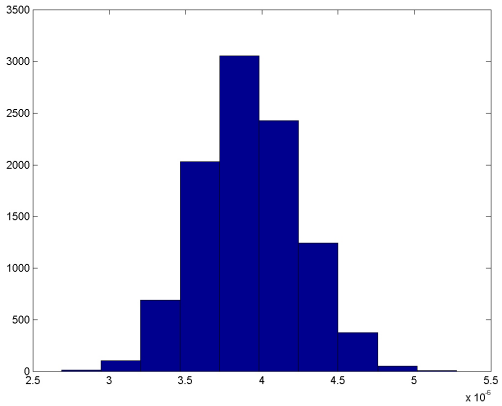
(a) Histogram of errors: Clenshaw 16



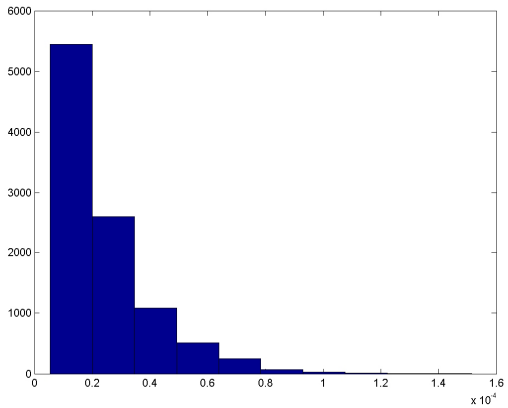
(b) Histogram of errors: Forsythe 16



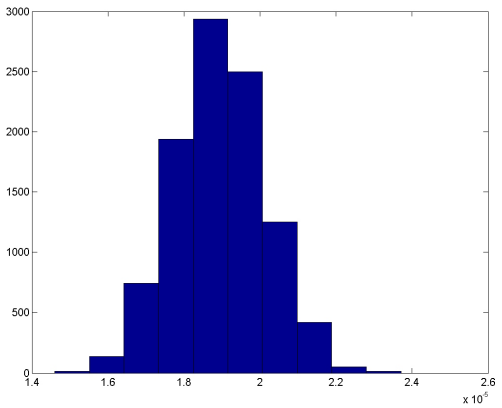
(c) Histogram of errors: Clenshaw 256



(d) Histogram of errors: Forsythe 256



(e) Histogram of errors: Clenshaw's 10 24



(f) Histogram of error: Forsythe's 10 24

Figure 5: Histogram of mean square errors in the computation of the SCT

4. SUMMARY

We compared the numerical accuracy of Clenshaw's and Forsyth's methods for the computation of the recursive symmetric discrete cosine transform. Both computational methods are front-to-back and are easily implemented on parallel platforms and hardware. When compared with Clenshaw's method, the recursive part of Forsyth's algorithm is more complex. However, we found that Forsythe's algorithm resulted in lower mean square errors than Clenshaw's method except for input sequences of length 16. Other DCT forms can be computed using the SCT transform as an intermediate step.

5. REFERENCES

1. R. Barrio and J. Sabadell, "A parallel algorithm to evaluate Chebyshev series on message passing environment", SIAM Journal of Scientific Computing, Vol. 20, No. 3, 1999, pp. 964-969.
2. M.F. Aburdene, J. Zheng and R. Kozick, "Computation of Discrete Cosine Transform Using Clenshaw's Recurrence Formula", IEEE Signal Processing Letters, Vol. 2, No. 8, August 1995, pp. 155-156.
3 W. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd Edition, Cambridge University Press, 1992.
4. L. Kronsjo, *Algorithms: Their Complexity and Accuracy*, Second Edition, John Wiley & Sons, 1987.
5. H.Kitajima, "A Symmetric Cosine Transform", IEEE Transaction on Computers, Vol. C-29, No. 4, April 1980, pp.317-323.
6. K.R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Inc., San Diego, 1990.
7. J.E. Dorband and M.F. Aburdene, "Architecture-adaptive computing environment: a tool for teaching parallel programming", Proceedings of 32nd Annual, Frontiers in Education, Volume 3, 6-9 Nov. 2002, pp.S2F-7 - S2F-12.

6. ACKNOWLEDGEMETS

Partial support for this work was provided by NASA Grant NNG04GA29G.

7. APPENDIX A: a Ce C Program

```
/*  1-D Discrete Cosine Transform, SCT
aCe Program

Maurice F. Aburdene
John E. Dorband
_____________________________________________________________________

1D DCT formula for N samples in y[]:


Y(m)= sqrt(2/N) * km * SUM(n=0->N){ kn * y(n) * cos [ m*n*pi/N ] }

        km = 1/sqrt(2) for m=0,N and 1 otherwise.
        kn = 1/sqrt(2) for n=0,N and 1 otherwise.

_____________________________________________________________________
*/

#define real float

# define Direct   1
```

```
# define Clenshaw 1
# define Forsyth  1

#include <stdio.aHr>
#include <math.aHr>

#define N 4
#define PI   (4*atan(1))

threads Z[N+1];

Z real V[N+1], T[N+1], b[N+1] ;

/*initialize bundle variables*/

Z void inputdata( void ) {
        int i;
        /*Generates values*/
        for(i=0;i<=N;i++) { b[i]=1.0; }
}

Z real DCTrow ( real b[N+1] )
{

        real km, kn, value ;
        int j,k;

        km = kn = 1.0/sqrt(2);

        value = 0.0 ;
        for (j=0;j<=N;j++) {
                b[j] = b[j] * cos($$i*j*PI/N);
        }
        for (j=0;j<=N;j++) {
                if(($$i==0)||($$i==N)) b[j] = b[j] * km ;
                if((j==0)||(j==N)) b[j] = b[j] * kn ;
        }
        for (j=0;j<=N;j++) {
                value += b[j];
        }
        value *= sqrt(2)/(sqrt(N));

        return value;
}

Z real DCTclenshaw ( real b[N+1] )
{
        real km, kn ;
        int j,k; real Y, value ;

        km = kn = 1.0/sqrt(2);
        for (j=0;j<=N;j++) {
                if(($$i==0)||($$i==N)) b[j] = b[j] * km ;
                if((j==0)||(j==N)) b[j] = b[j] * kn ;
```

```
	}

	Y = 2*cos($$i*PI/N) ;
	V[0] = b[0] ;
	V[1] = Y * V[0] + b[1] ;
	for(k=2;k<=N;k++) {
		V[k] = ( Y * V[k-1] - V[k-2] + b[k] ) ;
	}

	value = V[N-1] * cos($$i*PI*(N-1)/N) + (b[N]-V[N-2])*cos($$i*PI) ;

	return value*(sqrt(((real)2)/N));
}

Z real DCTforsyth ( real b[N+1] )
{
	real km, kn ;
	int j,k; real Y, value ;

	km = kn = 1.0/sqrt(2);
	for (j=0;j<=N;j++) {
		if(($$i==0)||($$i==N)) b[j] = b[j] * km ;
		if((j==0)||(j==N)) b[j] = b[j] * kn ;
	}

	Y = 2*cos($$i*PI/N) ;
	T[0] = 1;
	V[0] = b[0] ;
	T[1] = cos($$i*PI/N) ;
	V[1] = V[0] + b[1]*T[1] ;
	for(k=2;k<=N;k++) {
		T[k] = Y * T[k-1] - T[k-2] ;
		V[k] = V[k-1] + b[k]*T[k] ;
	}

	value = V[N] ;

	return value*(sqrt(((real)2)/N));
}


void main (){

	Z.{
		int i; real W;

# if Direct
		inputdata();
		W = DCTrow(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
		for(i=0;i<=N;i++) { b[i]=Z[i].W; }
		W = DCTrow(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
```

```
MAIN.{ printf("\n"); }
# endif

# if Clenshaw
                        inputdata();
                        W = DCTclenshaw(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
                        for(i=0;i<=N;i++) { b[i]=Z[i].W; }
                        W = DCTclenshaw(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
# endif

# if Forsyth
                        inputdata();
                        W = DCTforsyth(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
                        for(i=0;i<=N;i++) { b[i]=Z[i].W; }
                        W = DCTforsyth(b);
printf("W[%d]=%e  ",$$i,W); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
# endif

            }


}

/*
for (k=0;k<=N;k++) { printf("X[%d][%d]=%e  ",$$i,k,X[k]); MAIN.{ printf("\n"); } }
MAIN.{ printf("\n"); }
for (k=0;k<=N;k++) { printf("X[%d][%d]=%e  ",$$i,k,X[k]); MAIN.{ printf("\n"); } }
MAIN.{ printf("\n"); }
for (k=0;k<=N;k++) { printf("X[%d][%d]=%e  ",$$i,k,X[k]); MAIN.{ printf("\n"); } }
MAIN.{ printf("\n"); }
printf("Y[%d]=%e  ",$$i,value); MAIN.{ printf("\n"); }
MAIN.{ printf("\n"); }
*/
```