Harmonic Optimization of Multilevel Converters Using Genetic Algorithms

Burak Ozpineci[1]
[1]Oak Ridge National Laboratory
Knoxville, TN USA
Email: burak@ieee.org

Leon M. Tolbert[1,2], John N. Chiasson[2]
[2]The University of Tennessee
Knoxville, TN USA
Email: tolbert@utk.edu, chiasson@utk.edu

Abstract— **In this paper, a genetic algorithm (GA) optimization technique is applied to multilevel inverter to determine optimum switching angles for cascaded multilevel inverters for eliminating some higher order harmonics while maintaining the required fundamental voltage. This technique can be applied to multilevel inverters with any number of levels; as an example in this paper, a 7-level inverter is considered, and the optimum switching angles are calculated offline to eliminate the 5th and the 7th harmonics. Then, these angles are used in an experimental setup to validate the results.**

I. Introduction

Multilevel inverters have been drawing growing attention in the recent years especially in the distributed energy resources area due to the fact that several batteries, fuel cells, solar cell, wind, and microturbines can be connected through a multilevel inverter to feed a load or the ac grid without voltage balancing problems. Another major advantage of multilevel inverters is that their switching frequency is lower than a traditional inverter, which means they have reduced switching losses.

The output waveforms of multilevel inverters are in a stepped form; therefore they have reduced harmonics compared to a square wave inverter. To reduce the harmonics further, different multilevel sinusoidal PWM and space-vector PWM schemes are suggested in the literature [1,2]; however, PWM techniques increase the control complexity and the switching frequency. Another approach to reduce the harmonics is to calculate the switching angles in order to eliminate certain order harmonics. Chiasson et. al. [3-5] derived analytical expressions using the mathematical Resultant Theory to compute the optimum switching angles. These expressions were polynomials of 22^{nd} degree, are difficult and time consuming to derive, and for any change of levels or voltage inputs, new expressions are required.

In this paper, a general genetic algorithm (GA) approach will be presented, which solves the same problem with a simpler formulation and with any number of levels without extensive derivation of analytical expressions.

GA is a search method to find the maximum of functions by mimicking the biological evolutionary processes. There are only a few examples of GA applications for power electronics in the literature [6-8], but none on GA applied to multilevel inverters.

Prepared by the Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, managed by UT-Battelle for the U.S. Department of Energy under contract DE-AC05-00OR22725.

II. Cascaded Multilevel Inverters

The cascaded multilevel inverter is one of several multilevel configurations. It is formed by connecting several single-phase H-bridge converters in series as shown in Fig. 1 for a 7-level inverter. Each converter generates a square wave voltage waveform with different duty ratios, which together form the output voltage waveform as in Fig. 2. A three-phase configuration can be obtained by connecting three of these converters in Y or Δ.

For harmonic optimization, the switching angles θ_1, θ_2, and θ_3 (for a 7-level inverter) shown in Fig. 2, have to be selected so that certain order harmonics are eliminated.

III. Genetic Algorithm (GA)

Genetic algorithm is a computational model that solves optimization problems by imitating genetic processes and the theory of evolution. It imitates biological evolution by using genetic operators like reproduction, crossover,

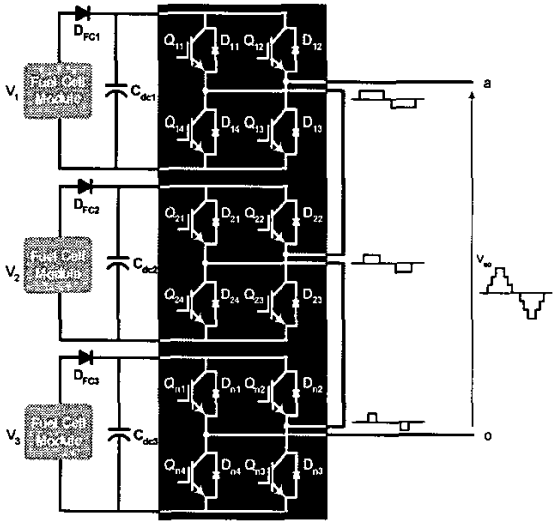

Fig. 1. 7-level cascaded multilevel inverter

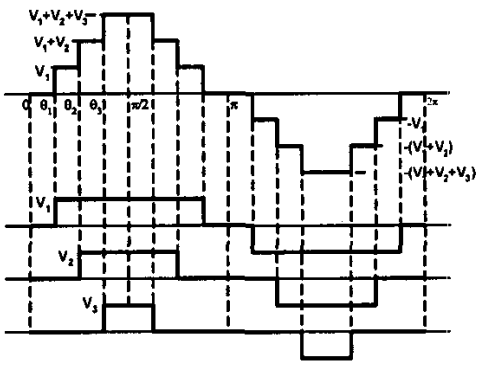

Fig. 2. 7-level cascaded multilevel inverter waveform generation

mutation, etc.

Optimization in GA means maximization. In cases where *minimization is required, the negative or the inverse of the* function to be optimized is used.

To minimize a function, $f(x_1,x_2,...,x_k)$ using GA, first, each x_i is coded as a binary or floating-point string of length m. In this paper, a binary string is preferred, e.g.

$$\begin{aligned} x_1 &= [10001...01001] \\ x_2 &= [00101...11110] \\ &\ldots \ \ldots \ \ldots \ \ldots \ \ldots \\ x_k &= [1110...01011] \end{aligned} \quad (1)$$

The set of $\{x_1, x_2,...,x_k\}$ is called a *chromosome* and x_i are called *genes*. The algorithm works as follows:

B. Initialize population

Set a population size, N, i.e. the number of chromosomes in a population. Then initialize the chromosome values randomly. If known, the range of the genes should be considered for initialization. The narrower the range, the faster GA converges.

$$\text{Population, } P=\begin{Bmatrix} x_{1,1},x_{2,1},...,x_{k,1} \\ x_{1,2},x_{2,2},...,x_{k,2} \\ \ldots \ \ldots \ \ldots \ \ldots \\ x_{1,N},x_{2,N},...,x_{k,N} \end{Bmatrix} \quad (2)$$

C. Evaluate each chromosome

Use a cost function specific to the problem at hand to evaluate the fitness value (FV) of each chromosome,

$$FV=\frac{1}{f(x_1,x_2,...,x_k)} \quad \text{or} \quad FV=-f(x_1,x_2,...,x_k) \quad (3)$$

Add all the FVs to get the total fitness. Divide each FV by the total FV and find the weight/probability of selection, p_i, for each chromosome. The integer part of the product, p_iN gives the number of descendents (offspring) from each chromosome. At the end, there should be N descendent chromosomes. If the number of descendents calculated is less then N, the rest of the descendents are found randomly considering the reproduction probabilities, p_i of each chromosome.

D. Crossover Operation

A floating number (between 0 and 1) for each chromosome is assigned randomly. If this number is smaller than a pre-selected *crossover probability*, this chromosome goes into *crossover*. The chromosomes undergoing crossover are paired randomly. In this case assume x_1 and x_2 are paired. The crossing point is randomly selected, assume 3 in this case.

Then, before crossover,

$$\begin{aligned} x_1 &= [10001...01001] \\ x_2 &= [00101...11110] \end{aligned} \quad (4)$$

and after crossover,

$$\begin{aligned} x_1 &= [10001...11110] \\ x_2 &= [00101...01001] \end{aligned} \quad (5)$$

As seen above, the bits after the 3rd one are exchanged.

E. Mutation Operation:

A floating number (between 0 and 1) for each bit is assigned randomly. If this number is smaller than a pre-selected *mutation probability*, this bit mutates. Assume that the 2nd and 4th bits of x_1 and 2nd, 3rd and 5th bits of x_2 need to be mutated.

Then, before mutation and after crossover,

$$\begin{aligned} x_1 &= [10001...11110] \\ x_2 &= [00101...01001] \end{aligned} \quad (6)$$

and after mutation,

$$\begin{aligned} x_1 &= [11011...11110] \\ x_2 &= [01000...01001] \end{aligned} \quad (7)$$

Finally, the new population is ready for another cycle of genetic algorithm. The algorithm runs a certain number of times as required by the user. At the end, the chromosome with the maximum FV is the answer.

IV. Formulating the Problem

GA algorithm explained in the previous section is the same for any application. There are only a few parameters to be set for GA to work. The steps for formulating a problem and applying GA are as follows:

1- Select binary or floating point strings.

2- Find the number of variables specific to the problem; this number will be the number of genes in a chromosome. In this application, the number of variables is the number of controllable switching angles, which is the number of H-bridges in a cascaded multilevel inverter.

A 7-level inverter requires three H-bridges; thus, each chromosome for this application will have three switching angles, i.e. $\{\theta_1, \theta_2, \theta_3\}$.

3- Set a population size and initialize the population. In this application a population size of 20 is selected. Higher population might increase the rate of convergence but also increases the execution time. The selection of optimum sized population requires some experience in GA.

The population in this paper has 20 chromosomes, each containing three switching angles. The population is initialized with random angles between 0 and 90 degrees

taking into consideration the quarter-wave symmetry of the output voltage waveform.

4-The most important item for the GA to evaluate the fitness of each chromosome is the cost function. The objective of this study is to minimize some harmonics; therefore the cost function has to be related to these harmonics. As an example assume that the 5th and 7th harmonics at the output of a 7-level inverter have to be minimized. Then, the cost function, f can be selected as the sum of these two harmonics normalized to the fundamental,

$$f(\theta_1,\theta_2,\theta_3)=100\times\frac{|V_5|+|V_7|}{|V_1|}. \quad (8)$$

where θ_i are the switching angles and V_n are the nth order voltage harmonics.

For each chromosome, a multilevel output voltage waveform (Fig. 2) is created using the switching angles in the chromosome and the required harmonic magnitudes are calculated using FFT techniques.

The fitness value, FV is calculated for each chromosome inserting (8) in (3). In this case,

$$FV(\theta_1,\theta_2,\theta_3)=-100\times\frac{|V_5|+|V_7|}{|V_1|} \quad (9)$$

is used. The switching angle set producing the max FV is the best solution of the first iteration.

5- GA is usually set to run for a certain number of iterations (100 in this case) to find an answer. After the first iteration, FV's are used to determine new offspring as explained in Section II. These go through crossover and mutation operations and a new population is created which goes through the same cycle starting from FV evaluation.

Sometimes, GA can converge to a solution much before 100 iterations are completed. To save time, in this paper, the iterations have been stopped when the cost function goes below 1 in which case the sum of the 5th and the 7th harmonics is negligible compared to the fundamental. As seen in Fig. 3, GA resulted in cost functions even smaller than 0.4.

Note that after these iterations, GA finds one solution; therefore, it has to be run as many times as the number of solutions required to cover the whole modulation index range.

The MATLAB source code required to solve this problem for any number of levels and up to any number of harmonics is given in the Appendix at the end of the paper. The MATLAB GA Optimization Toolbox still needs to be downloaded from [9].

V. Results

For the 7-level inverter, switching angles, which minimize the 5th and 7th harmonics, are shown in Fig. 3.

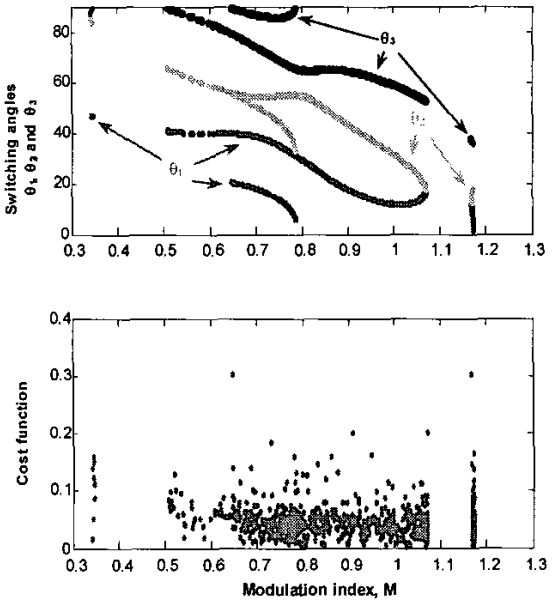

Fig. 3. Solutions for θ_1, θ_2, and θ_3 and the cost function

Note that this plot is similar to the one in [3] but has more solutions. The reason for this is that in [3], the solution only includes angles that result in zero 5th and 7th order harmonics. In this paper, however, as seen in the bottom plot of Fig. 3, any solution that yields a cost function less than 1 is accepted. This means that if low harmonics are tolerable, then a wider solution space is available.

In this figure for certain modulation indices, several sets of solutions are available. Either of these solutions can be used to minimize the selected harmonics. Another possibility [3] is to calculate THDs for each solution set and use the set that gives the lowest THD.

As can be observed in Fig. 3, for some modulation indices, no solution sets are available. This means that for those modulation indices, either there is not a solution or GA could not find one. The former reason is more of a possibility than the latter.

Fig. 4 shows the experimental 7-level voltage waveform for M=0.42. Fig. 5, on the other hand, shows the first 15 harmonics of the waveform in Fig. 4. As seen in this figure, the 5th and the 7th harmonics of the voltage waveform are negligible.

Fig. 6 shows the optimum switching angles when this technique is applied to a 9-level inverter (4 H-bridges, 4 switching angles) to minimize the 5th, 7th, 11th, and 13th harmonics. In this case, for the angles GA selected, the cost function is bound at 0.2, which much lower than the previous case.

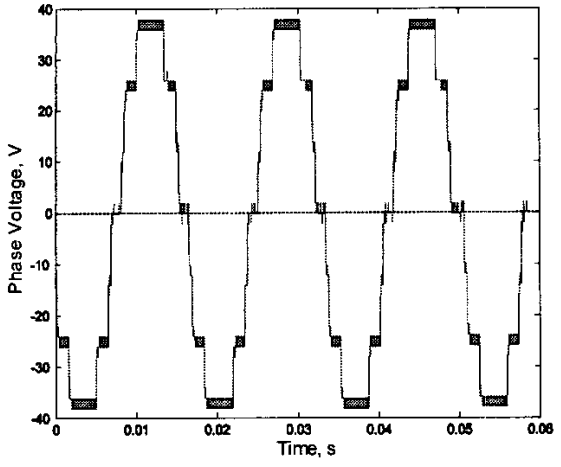

Fig. 4. Experimental output voltage waveform

VI. CONCLUSIONS

The comparison of the results in this paper to similar work in the literature shows that the GA approach for the harmonic optimization of multilevel inverters works properly. As in this approach, GA can be applied to any problem where optimization is required; therefore, it can be used in many applications in power electronics. A freely available MATLAB GA optimization toolbox [9] can be used for any optimization needs. When the toolbox is used, the only programming required for the GA application is given in the appendix below.

REFERENCES

[1] L.M Tolbert, F. Z. Peng, T.G. Habetler, "Multilevel PWM methods at low modulation indices," *IEEE Transactions on Power Electronics*, 15(4), July 2000, pp. 719 – 725.

[2] L.M. Tolbert, T.G. Habetler, "Novel multilevel inverter carrier-based PWM method," *IEEE Transactions on Industry Applications*, 35(5), Sept.-Oct. 1999, pp. 1098 – 1107.

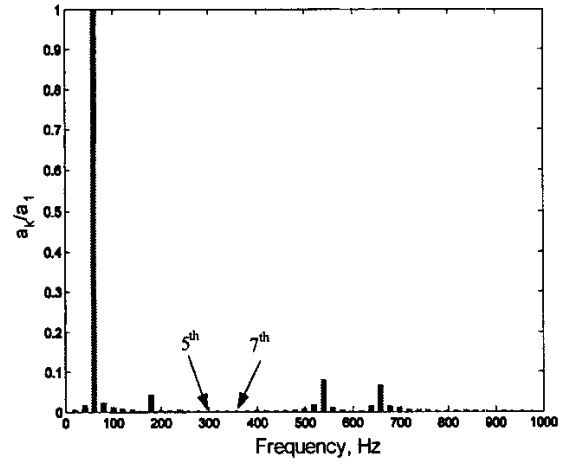

Fig. 5. Normalized (with respect to the fundamental) FFT vs frequency.

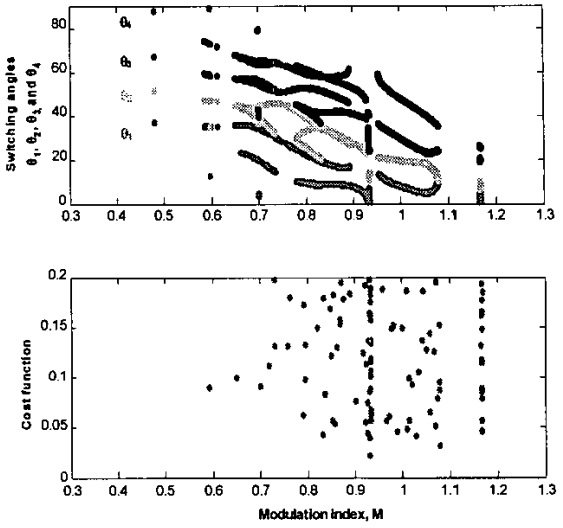

Fig. 6. Solutions for θ_1, θ_2, θ_3, and θ_4 and the cost function.

[3] J. Chiasson, L. M. Tolbert, K. McKenzie, Z. Du, "Eliminating harmonics in a multilevel converter using resultant theory," *Conference Proceedings of IEEE Power Electronics Specialists Conference*, 2002, vol. 2, 503– 508.

[4] J. N. Chiasson, L. M. Tolbert, K. J. McKenzie, Z. Du, "A Complete Solution to the Harmonic Elimination Problem," *IEEE Transactions on Power Electronics*, 19(2), March 2004, pp. 491 – 499.

[5] J. N. Chiasson, L. M. Tolbert, K. J. McKenzie, Z. Du, "A Unified Approach to Solving the Harmonic Elimination Equations in Multilevel Converters," *IEEE Transactions on Power Electronics*, 19(2), March 2004, pp. 478 – 490.

[6] B. Ozpineci, J. O. P. Pinto, L. M. Tolbert, "Pulse-width optimization in a pulse density modulated high frequency AC-AC converter using genetic algorithms," *Conference Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 2001, pp. 1924 – 1929.

[7] A. I. Maswood, S. Wei, M. A. Rahman, "A Flexible Way to Generate PWM-SHE Switching Patterns Using Genetic Algorithms," *Conference Proceedings of IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2001, pp. 1130– 1134.

[8] M. J. Schutten, D. A. Torrey, "Genetic Algorithms for Control of Power Converters," *Conference Proceedings of IEEE Power Electronics Specialists Conference*, 1995, pp. 1321– 1326.

[9] C. Houck, J. Joines, M. Kay, *The Genetic Algorithm Optimization Toolbox (GAOT) for Matlab 5*, http://www.ie.ncsu.edu/mirage/GAToolBox/gaot.

APPENDIX

A. Matlab source code for the menu

```
% Main menu to select the number of levels of the inverter,
% the number of solution sets required, the maximum
% number of iterations, the minimum error tolerable, and
% up to what number of harmonics will be minimized.
function [p] = mainmenu(px)
clc
disp(sprintf('1-Change the number of levels                :     %g',px(1)));
disp(sprintf('2-Change number_of_points                    :     %g',px(2)));
disp(sprintf('3-Change the maximum number of iterations:     %g',px(3)));
disp(sprintf('4-Change the minimum error (percent)         :     %g',px(4)));
disp(sprintf('5-eliminate (6n+1) harmonics. n              :     %g',px(5)));

p=input('Make a selection (1-4) or Press enter to run :  ');
if (isempty(p)==1),p=0;end
```

B. Matlab source code for the main program

```
% Finds the optimum angles to minimize up to the 6n+-1th harmonics using
GA.
% Uses functions
%          "fitness.m" to evaluate the fitness of the population
%          "mainmenu.m" to display an options menu
%
% If the percentage of the sum of the selected harmonics is less than
% "minh", the solution is accepted.  if not it is not accepted
%
% If some solutions are not accepted, then "number of points" will be
% different than the "number of iterations", otherwise they will be the
% same.
%
% For n cascaded converters, the number of levels is 2n+1
clear

global harm_n

p=100;
k=7;nop=2000;mnoi=20000;minh=2;harm_n=8;
px=[7 2000 20000 2 8];
while p~=0
   p=mainmenu(px);
   switch p
      case 0
      case 1
         k=(input('number of levels (default 7)=    ')-1)/2;
         if isempty(k),k=3;end
         px(1)=2*k+1;
      case 2
         nop=input('number_of_points (default 2000):    ');
         if isempty(nop),nop=2000;end
         px(2)=nop;
      case 3
         mnoi=input('maximum number of iterations (default 20000):     ');
         if isempty(mnoi),mnoi=20000;end
         px(3)=mnoi;
      case 4
         minh=input('minimum error (default 2):     ');
         if isempty(minh),minh=2;end
         px(4)=minh;
      case 5
         harm_n=input('eliminate (6n+1) harmonics. n (default 8):     ');
         if isempty(harm_n),harm_n=2;end
         px(5)=harm_n;
      otherwise
         disp('Error 01:Enter a value between 0 and 4')
         disp('Press enter to continue')
         pause
      end
   end

k=(px(1)-1)/2;
 nop=px(2);
 mnoi=px(3);
 minh=px(4);

for i=1:k
   pool(i,:)=[.01 .99];
end

file='fitness';
n=100;
out=zeros(1,k+3);
it=1;
it_all=0;
flag1=0;

VV=125*ones(1,k);

while (it<nop)
it_all=it_all+1;
if it_all==mnoi,return,end

if flag1==1
   it=size(out,1)+1;
   flag1=0;
else
   it;
end

%  Create a random initial population of size 20.

initPop=initializega(20,pool,file);

for i=1:20
   initPop(i,1:k)=sort(initPop(i,1:k));
end

[x endPop] = ga(pool,file,[],initPop,[1e-6 1 1],'maxGenTerm',n);

if (-x(k+1)<minh)
   MM=x;
   N=4096;
   [fftvaoo vaoo]=modulation(N,MM,VV);
   vvv=abs(fftvaoo(2));
   sumhh=0;
for g=1:harm_n
   h55=abs((fftvaoo(6*g+1+1)));   %(6n+1)th harmonic
   h77=abs((fftvaoo(6*g-1+1)));   %(6n-1)th harmonic
   sumhh=sumhh+h55+h77;
end
   xx=sort(x(1:k))*90;
   out(it,:)=[xx(1:k) vvv/sum(VV) -x(k+1) 100*(sumhh)/vvv]; %The best found
   it=it+1;
   flag1=0;
else
   flag1=1;
end

if flag1==0
   figure(1)
   subplot(2,1,1),plot(0:N/2-1,abs(fftvaoo(1:N/2)));
   axis([0,50,0,(abs(fftvaoo(2))*1.01)])

   subplot(2,1,2),plot(vaoo)
   hold, plot(abs(fftvaoo(2))*sin(2*pi*(1:N)/N),'r'),hold

   figure(2)
   subplot(3,1,1)
   for i=1:k
      plot(out(:,k+1),out(:,i),'.'),
      if i==1,hold,end
      if i==k,hold off,end
   end
   subplot(3,1,2),
   plot(out(:,k+1),sum(out,2)/(k*90),'.'),hold,
   plot(out(:,k+1),4/pi-out(:,k+1),'r'),
   plot(out(:,k+1),1-out(:,k+1),'g'),hold,ylabel('overall M')
   subplot(3,1,3),
   plot(out(:,k+1),(out(:,k+2)),'.'),hold on
   plot(out(:,k+1),(out(:,k+3)),'.r'),hold off, ylabel('cost function')
end
drawnow
end
```

C. Matlab source code for the fitness evaluation

```
% Evaluates the fitness of the chromosomes

function [sol, val] = fitness(sol,options)

global harm_n
M=sol;
N=4096; %N is a power of 2
kk=max(size(sol))-1;
V=125*ones(1,kk);
[fftvao v]=modulation(N,M,V);
```

```
vv=abs(fftvao(2));
sumh=0;
for g=1:harm_n
   h5=abs((fftvao(6*g+1+1)));   %(6n+1)th harmonic
   h7=abs((fftvao(6*g-1+1)));   %(6n-1)th harmonic
   sumh=sumh+h5+h7;
end

val=-100*(sumh)/vv;
```

D. *Matlab source code for creating output voltage*

```
% Modulation creates the multilevel output voltage.

function [fftv,vao] = modulation(N,M,V)
nn=1:N;
vc(1:N/4)=nn(1:N/4)/(N/4);
vc(N/4+1:N/2)=fliplr(vc(1:N/4));
vc(N/2+1:N)=-(vc(1:N/2));
for i=1:max(size(V))
   v(i,1:N)=(M(i)<=abs(vc))*V(i);
end
vao=sum(v).*sign(vc);
fftv=2*fft(vao)/N;
```