Moderne numerische Methoden der Festkörperphysik

Aufgabenblatt 4: Exakte Diagonalisierung der AF Heisenberg-Kette

Programmcode gen_matrix4ED.c

Bedienung des Programms

Berechnung der Grundzustandsenergien und NN-Korrelationen

Extrapolation zum thermodynamischen Limes

Ergebnisse für offene Randbedingungen

Ergebnisse für das XXZ-Modell

Programmcode ED_jacobi.c

Programmcode gen_matrix4ED.c

```
#define PROGNAME "gen_matrix4ED"
#define VERSION "0.4"
#define DATE  "25.06.2007"
#define AUTHOR "Nils Bluemer"

/* generates full Hamilton matrix of Heisenberg chain (for ED)
    in specified m_z sector
    NEW (0.2): option for open boundary conditions added
    NEW (0.3): sign error corrected (offdiagonal elements
               relative to diagonal), AF model now default
               error for N=2 with periodic bc's corrected */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pointer_utils.c"
```

```
void error(char error_text[])
/* standard error handler */
{
  fprintf(stderr,"\n %s run-time error\n", PROGNAME);
    fprintf(stderr,"--%s--\n",error_text);
    fprintf(stderr,"for general help use option -h\n");
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

long comp_mag(long i, long N)
{
  long mag;
  mag=0;
  while (i>0){
    mag+=i%2;
    i/=2;
  }
  return(2*mag-N);
}
```

```
void gen_conf(int *conf, long i, long N)
{
  long n;
  n=1;
  for(n=1;n<=N;n++){
    conf[n]=i%2;
    i/=2;
  }
}

void erase(int *v, long size)
{
  long i;
  for (i=0;i<size;i++)
    v[i]=0;
}

void print_ivector(int *v, long N)
{
  long i;
  for (i=1;i≤N;i++)
```

```
    printf("%6d",v[i]);
  printf("\n");
}

long E_Ising(int *v, long N, int periodic)
{
  long n, E;
  E=0;
  for (n=1;n<=N-1;n++)
    E+=(2*v[n]-1)*(2*v[n+1]-1);        /* sign in v0.3 corrected */
  E+=periodic*(2*v[N]-1)*(2*v[1]-1); /* sign in v0.3 corrected */
  return(E);
}

void printhelp ()
{
printf("*****************************************\n");
printf("%s:  prints matrix for 1D Heisenberg model\n",PROGNAME);
printf("Version:  %s, %s by %s\n",VERSION,DATE,AUTHOR);
printf("options:  -N# number of spins\n");
```

```
printf("          -m# mz sector (0<=m<=N)\n");
printf("          -o open boundary conditions\n");
printf("          -F ferromagnetic model\n");
printf("          -v# adjust verbosity (default:  1)\n");
printf("          -h this help\n");
}


int main (int argc, char *argv[])
{
  char c;
  long i, j, n, size, base;
  int N, mag, verbosity, periodic, sign;
  int *v, *conf;


  verbosity=1;
  periodic=1;
  sign=1;
```

```
while (--argc > 0 && (*++argv)[0] == '-')
  while (c= *++argv[0])
  switch (c) {
  case 'N':
     sscanf(++argv[0],"%d\n",&N);
           break;
  case 'm':
     sscanf(++argv[0],"%d\n",&mag);
           break;
  case 'o':
    periodic=0;
    break;
  case 'F':
    sign=-1;
    break;
  case 'v':
     sscanf(++argv[0],"%d\n",&verbosity);
           break;
  case 'h':
    printhelp();
    exit(0);
```

```
/*        default:   */
/*           error("No valid choice");   */
    }

  size=1;
  for (i=1;i<=N;i++)
    size*=2;

  v=ivector(0,size-1);
  conf=ivector(1,N);

  if (verbosity>0)
    if (sign>0)
      printf("# Hamiltonian for %d spin AF Heisenberg model
(mz=%d)\n",N,mag);
    else
      printf("# Hamiltonian for %d spin FM Heisenberg model
(mz=%d)\n",N,mag);

  j=0;
```

```
  for (i=0;i<size;i++)
    if (comp_mag(i,N)==mag)
      j+=1;
  printf("%d\n",j);

  for (i=0;i<size;i++)
    if (comp_mag(i,N)==mag){
      erase(v,size);
      gen_conf(conf,i,N);
      if (verbosity>1){
printf("i=%3d:  ",i);
print_ivector(conf,N);
      }

      v[i]=E_Ising(conf,N,periodic);   /* diagonal part of hamiltonian */
      base=1;
      for (n=1;n<=N-1;n++){
if (conf[n]!=conf[n+1])   /* spin flip possible */
   v[i + base*(2*conf[n]-1)]+=2;
base*=2;
```

```
        }
        if ((periodic>0)&&(conf[N]!=conf[1]))
v[i - (base-1)*(2*conf[n]-1)]+=2;

        for (j=0;j<size;j++)
if(comp_mag(j,N)==mag)
    printf("%4d",sign*v[j]);
        printf("\n");
     }

  free_ivector(v,0,size-1);
  free_ivector(conf,1,N);
}
```

Bedienung des Programms

Abruf der Hilfe-Seite mit Kommandozeilenparameter -h:

```
nils/NumMeth> ./gen_matrix4ED -h
*********************************************************
gen_matrix4ED: prints matrix for 1D Heisenberg model
Version: 0.4, 25.06.2007 by Nils Bluemer
options: -N# number of spins
         -m# mz sector (0<=m<=N)
         -o open boundary conditions
         -F ferromagnetic model
         -v# adjust verbosity (default: 1)
         -h this help
```

Erzeugung der Hamilton-Matrizen für $N = 4$:

```
nils/NumMeth> ./gen_matrix4ED -N4 -m4
# Hamiltonian for 4 spin AF Heisenberg model (mz=4)
1
   4
```

```
nils/NumMeth> ./gen_matrix4ED -N4 -m2
# Hamiltonian for 4 spin AF Heisenberg model (mz=2)
4
   0   2   0   2
   2   0   2   0
   0   2   0   2
   2   0   2   0

nils/NumMeth> ./gen_matrix4ED -N4 -m0
# Hamiltonian for 4 spin AF Heisenberg model (mz=0)
6
   0   2   0   0   2   0
   2  -4   2   2   0   2
   0   2   0   0   2   0
   0   2   0   0   2   0
   2   0   2   2  -4   2
   0   2   0   0   2   0
```

Erzeugung der Hamilton-Matrix für $N = 4$, $m_z = 2$ mit Angabe der Ising-Basis ($v = 2$) bzw. ohne Header (v=0):

```
nils/NumMeth> ./gen_matrix4ED -N4 -m2 -v2
# Hamiltonian for 4 spin AF Heisenberg model (mz=2)
4
i=  7:        1     1     1     0
   0   2   0   2
i= 11:        1     1     0     1
   2   0   2   0
i= 13:        1     0     1     1
   0   2   0   2
i= 14:        0     1     1     1
   2   0   2   0

nils/NumMeth> ./gen_matrix4ED -N4 -m2 -v0
4
   0   2   0   2
   2   0   2   0
   0   2   0   2
   2   0   2   0
```

Erzeugung der Hamilton-Matrizen für ferromagnetisches Modell bzw. offene Randbedingungen:

```
nils/NumMeth> ./gen_matrix4ED -N4 -m2 -F
# Hamiltonian for 4 spin FM Heisenberg model (mz=2)
4
   0  -2   0  -2
  -2   0  -2   0
   0  -2   0  -2
  -2   0  -2   0

nils/NumMeth> ./gen_matrix4ED -N4 -m2 -o
# Hamiltonian for 4 spin AF Heisenberg model (mz=2)
4
   1   2   0   0
   2  -1   2   0
   0   2  -1   2
   0   0   2   1
```

Berechnung der Grundzustandsenergien und NN-Korrelationen

```
nils/NumMeth> ./gen_matrix4ED -N4 -m0 -v0 | ./ED_jacobi -v1
Original matrix A:
  0.00000  2.00000  0.00000  0.00000  2.00000  0.00000
  2.00000 -4.00000  2.00000  2.00000  0.00000  2.00000
  0.00000  2.00000  0.00000  0.00000  2.00000  0.00000
  0.00000  2.00000  0.00000  0.00000  2.00000  0.00000
  2.00000  0.00000  2.00000  2.00000 -4.00000  2.00000
  0.00000  2.00000  0.00000  0.00000  2.00000  0.00000
Eigenvalues:
Eigen values:
  4.00000   0.00000   0.00000  -0.00000  -4.00000  -8.00000
final Eigenvectors:
  0.40825 -0.47689 -0.72280 -0.01140  0.00000 -0.28868
  0.40825 -0.00000 -0.00000  0.00000 -0.70711  0.57735
  0.40825  0.79857 -0.18540  0.27914  0.00000 -0.28868
  0.40825  0.04304  0.33009 -0.79949 -0.00000 -0.28868
  0.40825  0.00000  0.00000 -0.00000  0.70711  0.57735
  0.40825 -0.36471  0.57812  0.53175 -0.00000 -0.28868
```

Kurzversion:

```
nils/NumMeth> ./gen_matrix4ED -N4 -m0 -v0 | ./ED_jacobi -v0
Eigen values:
  4.00000   0.00000   0.00000  -0.00000  -4.00000  -8.00000
```

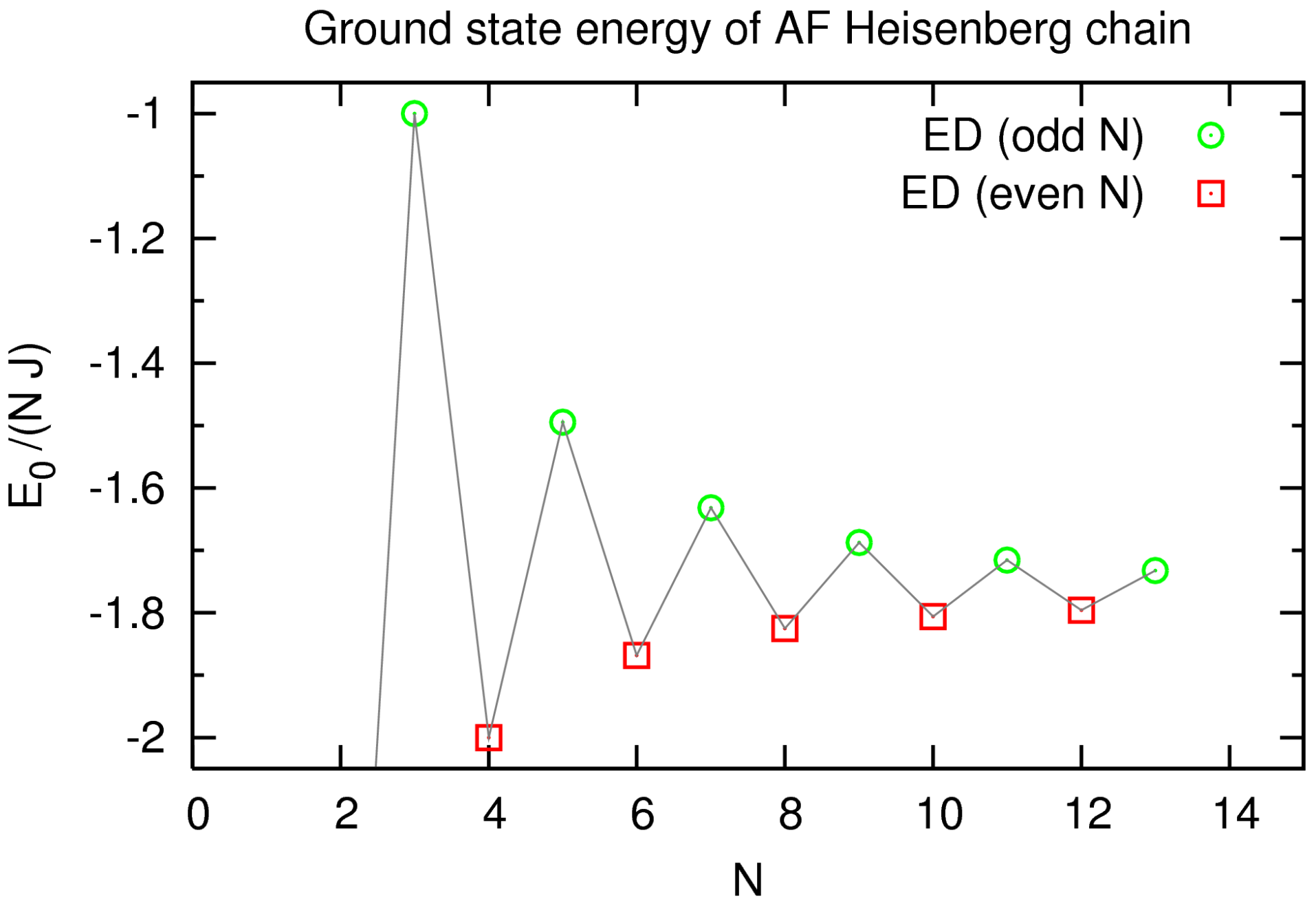
Ausgabe inklusive NN-Korrelationen:

```
nils/NumMeth> ./gen_matrix4ED -N4 -m0 -v0 | ./ED_jacobi -v0 -c
Eigen values:
  4.00000   0.00000   0.00000  -0.00000  -4.00000  -8.00000
# N * nearest-neighbor correlation (E_Ising): -2.666667
```

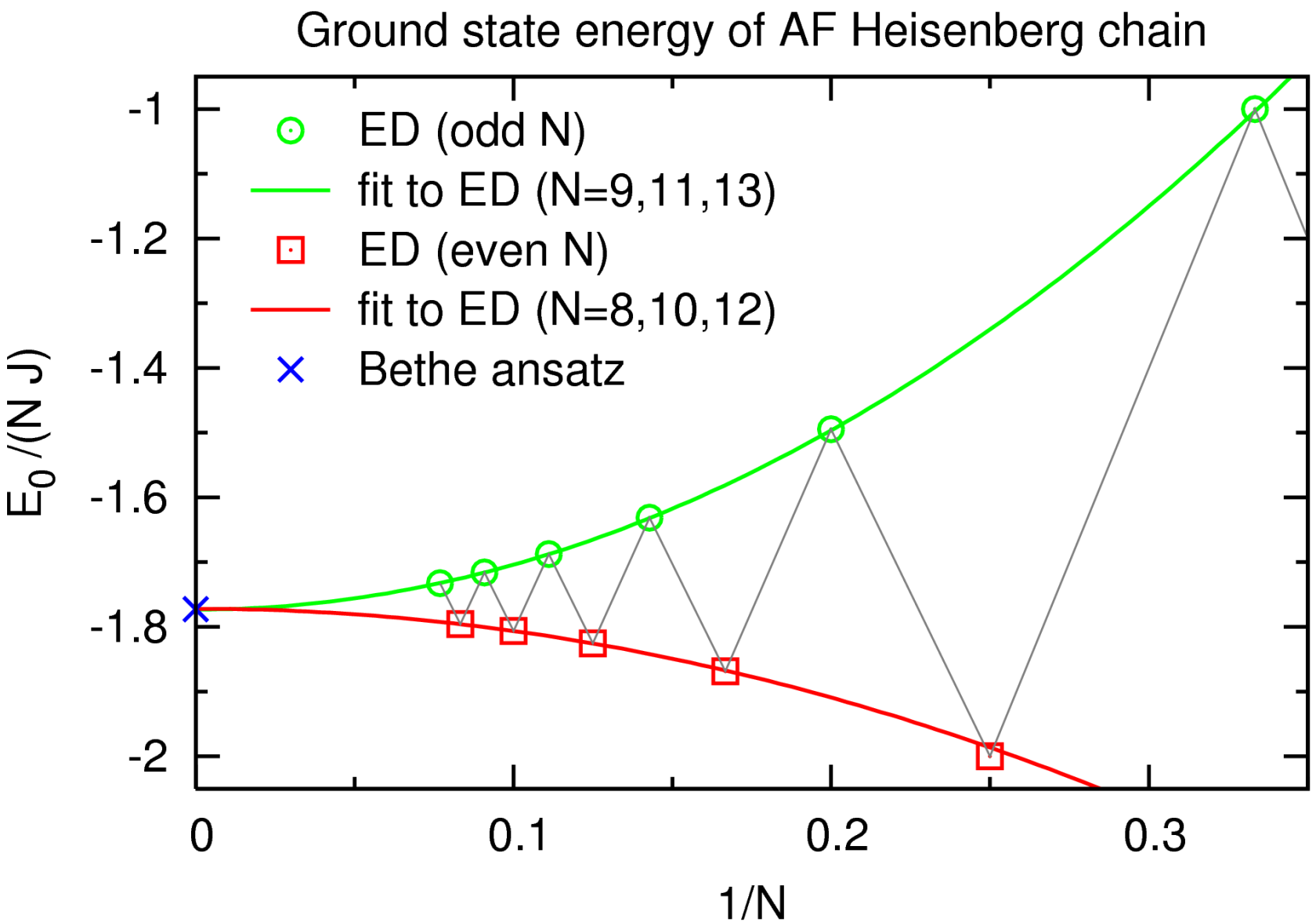
Ergebnisse (AF Heisenberg, periodische Randbedingungen):

```
# energies and NN correlations of AF Heisenberg model
2       -6.00000        -2.000000
3       -3.00000        -1.000000
4       -8.00000        -2.666667
5       -7.47214        -2.490712
6       -11.21110       -3.737034
7       -11.42072       -3.806906
8       -14.60437       -4.868125
9       -15.18920       -5.063066
10      -18.06179       -6.020595
11      -18.87575       -6.291915
12      -21.54956       -7.183188
13      -22.51834       -7.506112
```

1. Spalte: Anzahl Spins, 2. Spalte: Gesamtenergie, 3. Spalte: Ising-Anteil der Gesamtenergie.


Ground state energy of AF Heisenberg chain
ED (odd N)
ED (even N)
$E_0/(N\,J)$
-1
-1.2
-1.4
-1.6
-1.8
-2
0
2
4
6
8
10
12
14
N

Extrapolation zum thermodynamischen Limes



Linien sind jeweils least-squares fits an Funktionen $f(x) = f_0 + f_2 x^2$ mit $x = 1/N$:

```
nils/NumMeth> tail +2 Heisenberg_E_corr.dat | awk '{if ($1%2==1)
              print 1.0/$1, 0, $2/$1}' | tail -3
0.111111  0 -1.68769
0.0909091 0 -1.71598
0.0769231 0 -1.73218

nils/NumMeth> tail +2 Heisenberg_E_corr.dat | awk '{if ($1%2==1)
              print 1.0/$1, 0, $2/$1}' | tail -3 | polyfit -fe -lll
-1.7731571   0   6.9219847   0.77   3   0.0001003   0   0.01087


nils/NumMeth> tail +2 Heisenberg_E_corr.dat | awk '{if ($1%2==0)
              print 1.0/$1, 0, $2/$1}' | tail -3 | polyfit -fe -lll
-1.7719473   0   -3.4292301   0.33  3   9.183e-05  0     0.008029
```

Wir erhalten also die Ergebnisse: $E_0^{\text{odd}}/N = -1.7732 \pm 0.0001$,
$E_0^{\text{even}}/N = -1.7719 \pm 0.0001$.

Zum Vergleich: das exakte Ergebnis ist $E_0/N = 1 - 4\ln(2) \approx -1.7725887$.

Wieso Abweichung außerhalb der Fehlerbalken?

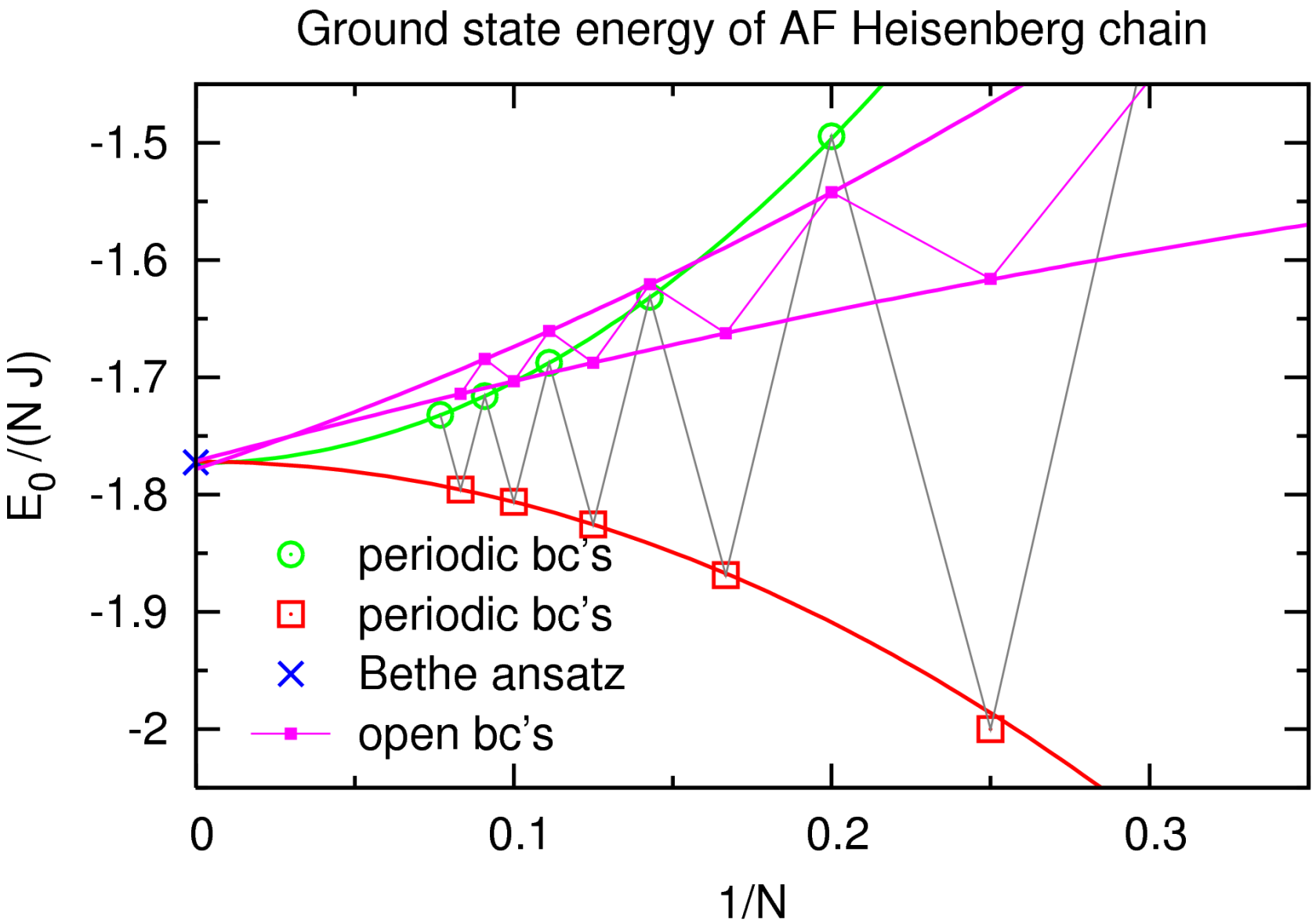
Hier nur heuristischer Fehler! Bessere Ergebnisse mit Fit höherer Ordnung:

```
nils/NumMeth> tail +2 Heisenberg_E_corr.dat | awk '{if ($1%2==0)
        print 1.0/$1, 0, $2/$1}' | tail -4 | polyfit -fe -o4 -llll
-1.772565 0 -3.3094966 0 -5.2153711  1 4  7.586e-05 0 0.01039 0 0.288
```

Neues Ergebnis: $E_0^{\text{even}}/N = -1.77257 \pm 0.00008$.

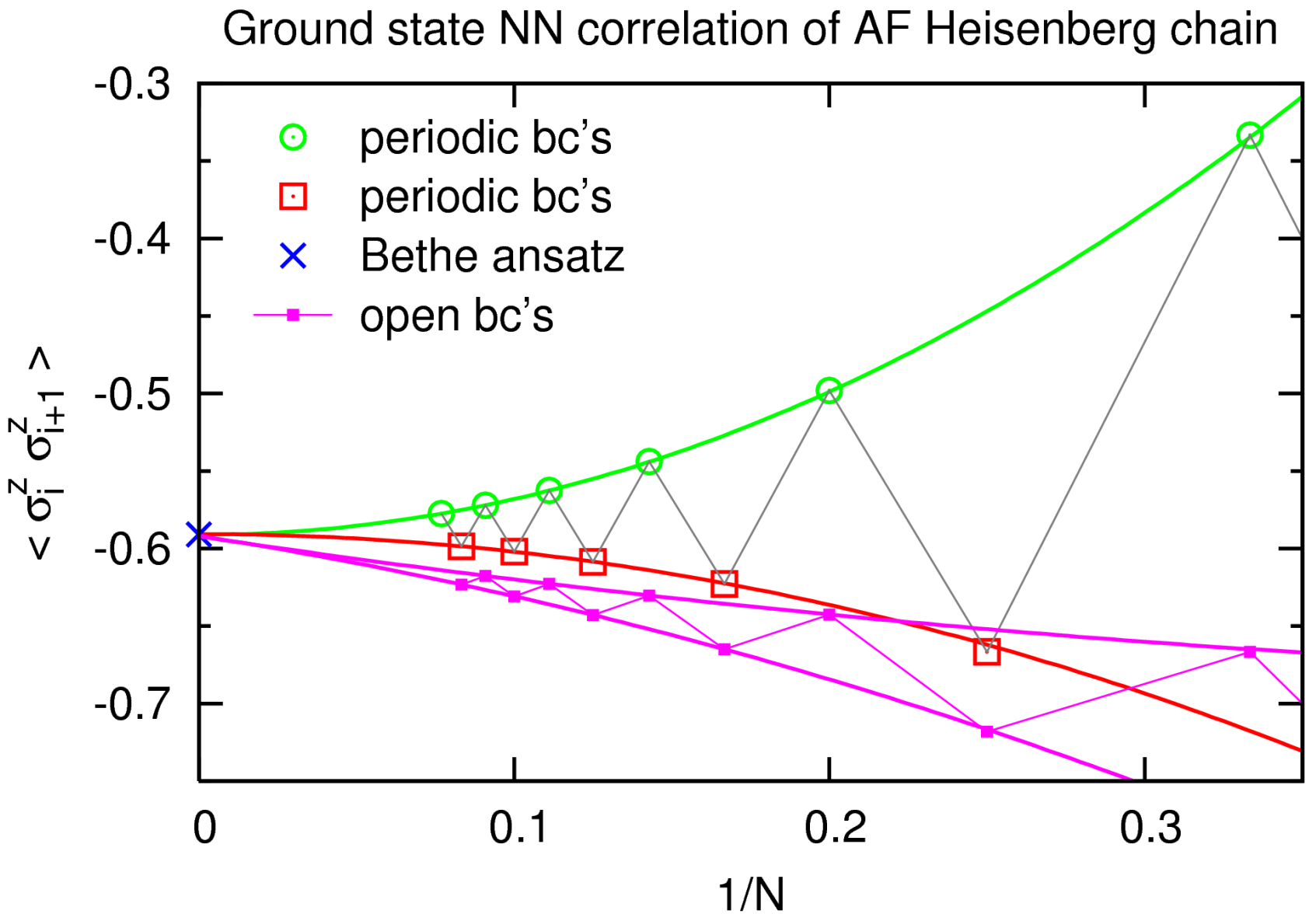
Ergebnisse für offene Randbedingungen

```
# AF Heisenberg chain with open boundary conditions
2       -3.00000        -1.00000
3       -4.00000        -1.333333
4       -6.46410        -2.154701
5       -7.71155        -2.570515
6       -9.97431        -3.324770
7       -11.34496       -3.781653
8       -13.49973       -4.499910
9       -14.94529       -4.981762
10      -17.03214       -5.677380
11      -18.52837       -6.176124
12      -20.56836       -6.856121
```



Offene Randbedingungen: schwächerer Even-Odd-Effekt, aber linearer Term in $1/N$

Nächst-Nachbar-Korrelationen: Vorfaktor $1/(N-1)$ für offene Randbedingungen:



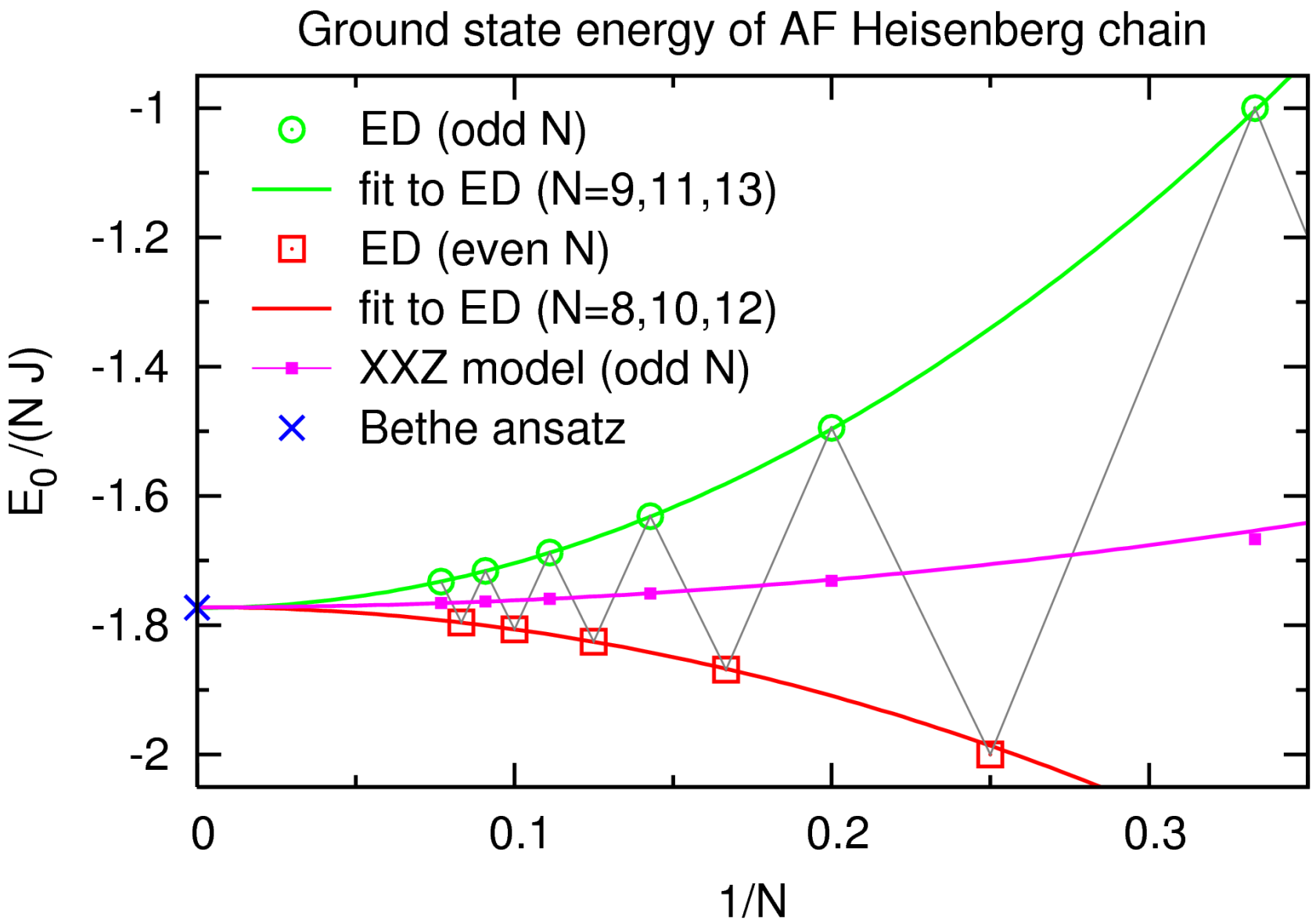
Ergebnisse für das XXZ-Modell

$$\hat{H} = \sum_{i=1}^{N} \left[\sigma_i^z \sigma_{i+1}^z - \sigma_i^x \sigma_{i+1}^x - \sigma_i^y \sigma_{i+1}^y \right] \quad ; \quad \vec{\sigma}_{N+1} \equiv \vec{\sigma}_1$$

Beachte: gegenüber dem AF Heisenberg-Modell sind die Vorzeichen von J_x und J_y vertauscht.

Beziehung zum AF-Heisenberg-Modell?

Äquivalent für offene Randbedingungen (N beliebig) und für gerades N bei periodischen Randbedingungen!



Viel kleinere Finite-Size-Fehler für XXZ-Modell!

Programmcode ED_jacobi.c

```
#define PROGNAME "ED_jacobi"
#define VERSION "0.3"
#define DATE  "14.06.2007"
#define AUTHOR "Nils Bluemer"

/* performs Jacobi sweep (for determination of eigenvalues)
    NEW (0.2): only output of energies at verbosity=0
    NEW (0.3): uses pointer_utils.c, comments added */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pointer_utils.c"

void error(char error_text[])
/* standard error handler */
{
```

```
  fprintf(stderr,"\n %s run-time error\n", PROGNAME);
    fprintf(stderr,"--%s--\n",error_text);
    fprintf(stderr,"for general help use option -h\n");
    fprintf(stderr,"...now exiting to system...\n");
    exit(1);
}

void printhelp ()
{
printf("*************************************************\n");
printf("%s:  eigenvalue utility\n",PROGNAME);
printf("Version:  %s, %s by %s\n",VERSION,DATE,AUTHOR);
printf("Input:  symmetric matrix on stdin, preceded by
dimension,\n");
printf("e.g.:  3\n      1 2 3\n      2 4 5\n      3 5 6\n");
printf("options:  -d# digits on output\n");
printf("          -v# verbosity (default:1)\n");
printf("          -c compute NN correlation for ground state\n");
printf("          -h this help\n");
}
```

```
void print_comp_matrix(double **a, double *d, int size, int digits)
/* prints matrix after some Jacobi rotations (reconstructed from a and d) */
{
  int i,j;
  char buffer [20];

  sprintf(buffer,"%%%d.%dlf",digits+4,digits);
/*    printf("%s\n",buffer); */
  for (i=1;i<=size;i++){
    for (j=1;j<=size;j++)
      if (i<j)
printf(buffer,a[i][j]);
      else
if (i>j)
   printf(buffer,a[j][i]);
else
   printf(buffer,d[i]);
    printf("\n");
  }
}
```

```
void print_vector(double *v, int size, int digits)
{
  int i;
  char buffer [20];

  sprintf(buffer,"%%%d.%dlf ",digits+4,digits);
  for (i=1;i<=size;i++)
    printf(buffer,v[i]);
  printf("\n");
}

void jacobi(double **a, int n, double d[], double **v, int *nrot,
int *sweeps, int verbosity, int digits)
/* standard routine for diagonalization, destroys input matrix a on output
    d: vector of eigenvalues, v: array of eigenvectors, n: dimension,
    nrot: number of jacobi rotations, sweeps: number of iterations (sweeps),
    verbosity: determines output, digits: determines precision on output */
{

...

   ++(*nrot);
```

```
    if (verbosity>=4){
      printf("Matrix A after %d Jacobi sweeps (%d
rotations):\n",*sweeps,*nrot);
      print_comp_matrix(a,d,n,digits);

...

      (*sweeps)++;
      if (verbosity>=2){
        printf("Matrix A after %d Jacobi sweeps (%d
rotations):\n",*sweeps,*nrot);
        print_comp_matrix(a,d,n,digits);
      }
      if (verbosity>=3){
        printf("Matrix v after %d Jacobi sweeps (%d
rotations):\n",*sweeps,*nrot);
        print_matrix(v,n,digits);
      }
    }
    error("Too many iterations in routine jacobi");
}
```

```
void eigsrt(double d[], double **v, int n)
/* sorts eigenvalues and eigenvectors by eigenvalues (largest first) */
{
int k,j,i;
double p;

for (i=1;i<n;i++) {
  p=d[k=i];
  for (j=i+1;j<=n;j++)
   if (d[j] >= p) p=d[k=j];
  if (k != i) {
   d[k]=d[i];
   d[i]=p;
   for (j=1;j<=n;j++) {
    p=v[j][i];
    v[j][i]=v[j][k];
    v[j][k]=p;
   }
  }
}
```

```
}


void read_matrix(double **a, int size)
{
  int i,j;
  double dummy;

  for (i=1;i<=size;i++)
    for (j=1;j<=size;j++){
      scanf("%lf", &dummy);
      a[i][j]=dummy;
    }
}

int main (int argc, char *argv[])
{
  char c;
  int i;
  int digits, size, nrot, sweeps, verbosity, corr;
```

```
double **a, **v, *d, *a_diag, x, prob;


digits=5;
verbosity=2;
corr=0;

while (--argc > 0 && (*++argv)[0] == '-')
  while (c= *++argv[0])
  switch (c) {
  case 'd':
     sscanf(++argv[0],"%d\n",&digits);
           break;
  case 'c':
     corr=1;
           break;
  case 'v':
     sscanf(++argv[0],"%d\n",&verbosity);
           break;
  case 'h':
    printhelp();
```

```
      exit(0);
/*       default:   */
/*          error("No valid choice");   */
    }

  scanf("%d",&size);
  a=dmatrix(1,size,1,size);
  v=dmatrix(1,size,1,size);
  d=dvector(1,size);

  read_matrix(a,size);

  if (verbosity>0){
    printf("Original matrix A:\n");
    print_matrix(a,size,digits);
    printf("Eigenvalues:\n");
  }

  if (corr>0){ /* copy diagonal elements of a */
    a_diag=dvector(1,size);
    for (i=1;i<=size;i++)
```

```
    a_diag[i]=a[i][i];
}


nrot=0; sweeps=0;

jacobi(a,size,d,v,&nrot,&sweeps,verbosity,digits);

eigsrt(d,v,size);

printf("Eigen values:\n");
print_vector(d,size,digits);

if (verbosity>=1){
  printf("final Eigenvectors:\n");
  print_matrix(v,size,digits);
}

if (corr>0){
  x=0;
```

```
    for (i=1;i<=size;i++){
      prob=v[i][size]*v[i][size];
      x+=a_diag[i]*prob;
    }
    printf("# N * nearest-neighbor correlation (E_Ising):
%8.6lf\n", x);
  }

  free_dmatrix(a,1,size,1,size);
  free_dmatrix(v,1,size,1,size);
  free_dvector(d,1,size);
}
```