Computer simulations in statistical physics
Homework 4: Ising model

Code of program mc_Ising

Raw data: traces, covariance and autocorrelation time

Simple observables: $|M|$, M^2, E, E^2

Smart observables: susceptibility, specific heat & Binder's cumulant

Finite-size scaling plots

Code of program mc_Ising

```
#define PROGNAME "mc_Ising_2D"
#define VERSION "0.2"
#define DATE  "22.11.2006"
#define AUTHOR "Nils Bluemer"

/* Metropolis Monte Carlo for 2D-Ising Model with pbc */
/* adapted from http://gold.cchem.berkeley.edu/~acpan/220A/2DIsing.c */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/*************************
 * Constant Declarations *
 *************************/
#define NMAX 40                 /* Lattice size (i.e. NMAX x NMAX) */
#define J -1.0                  /* Ising lattice coupling constant
```

```
        J < 0: Ferromagnetic
        J > 0: Antiferromagnetic */
#define H 0.0                       /* Ising lattice field strength */

void printhelp ()
{
printf("*************************************************\n");
printf("%s:  Metropolis Monte Carlo for 2D Ising
model\n",PROGNAME);
printf("Version:  %s, %s by %s\n",VERSION,DATE,AUTHOR);
printf("based on
http://gold.cchem.berkeley.edu/~acpan/220A/2DIsing.c\n");
printf("options:  -T temperature (really:  k_B T/|J|)\n");
printf("          -L linear dimension of lattice (L<=%d)\n",NMAX);
printf("          -w# number of initialization sweeps\n");
printf("          -n# number of measurement sweeps\n");
printf("          -c print configurations\n");
printf("          -h this help\n");
}
```

```
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 random_init(int mag[NMAX][NMAX], int L){
  int i,j;
    for(i = 0; i < L; i++)
    for(j = 0; j < L; j++){
if(drand48() > 0.5)
   mag[i][j] = 1;
else
   mag[i][j] = -1;
      }
}
```

```
void outputmag(int mag[NMAX][NMAX], int L){
  int i,j;
  printf("\n");
  for(i = 0; i < L; i++){
      for(j = 0; j < L; j++)
{
   if(mag[i][j] == -1)
     printf("* ");
   else
     printf("  ");
}
      printf("\n");
    }
}

double mag_av(int mag[NMAX][NMAX], int L){
/* computes average magnetization */
  int i,j;
  double x;
```

```
  x=0.0;

  for(i = 0; i < L; i++)
    for(j = 0; j < L; j++)
      x+=1.0*mag[i][j];
  x=x/(L*L);
  return(x);
}

/*****************************************************************/
int main(int argc, char *argv[]){

  /*************************
   * Variable Declarations *
   *************************/
  int mag[NMAX][NMAX];              /* 2D Ising Lattice */
  int i, j, k;                      /* Loop counters */
  int s,d;                          /* Lattice spin variables */
  double Energy;                    /* Total lattice energy */
  int Inn[4] = {1,-1,0,0};          /* Nearest neighbor array I */
  int Jnn[4] = {0,0,1,-1};          /* Nearest neighbor array J */
```

```
  int Inew, Jnew;                    /* Nearest neighbot indices */
  double Etemp, deltaE;              /* Temp energy variables for MC moves
*/
  int accept = 0;                    /* Number of accepted moves */
  int move = 0;                      /* Number of moves total */

  char c;
  double T;                          /* temperature (in units of J/k_B) */
  double beta;
  int sweeps;                        /* number of measurement sweeps */
  int warm;                          /* number of warm-up sweeps */
  int print_conf;                    /* flag for printing configurations */
  int L;                             /* lattice dimension */

  T=1.5;
  sweeps=500;
  print_conf=0;
  L=NMAX;

  while (--argc > 0 && (*++argv)[0] == '-')
```

```
while (c= *++argv[0])
switch (c) {
case 'n':
    sscanf(++argv[0],"%d\n",&sweeps);
          break;
case 'w':
    sscanf(++argv[0],"%d\n",&warm);
          break;
case 'L':
    sscanf(++argv[0],"%d\n",&L);
          if (L>NMAX) error ("L too large");
          break;
        case 'T':
    sscanf(++argv[0],"%lf\n",&T);
          break;
        case 'c':
          print_conf=1;
          break;
case 'h':
  printhelp();
  exit(0);
```

```
            break;
    }

  beta=1.0/T;
  printf("# L=%d, T=%f, w=%d, n=%d, print_conf=%d\n", L, T, warm,
sweeps, print_conf);

  /* Seed the random number generator */
  srand48((unsigned int) time(NULL) - (time(NULL)/100000)*100000);

  /* Choose your own initial configuration */
  random_init(mag, L);

  /* Determine the energy initially */
  Energy = 0.0;

  for(i=0;i<L;i++)
    for(j=0;j<L;j++){
/* Loop over nearest neighbors */
for(k=0;k<4;k++){
     Inew = i + Inn[k];
```

```
      Jnew = j + Jnn[k];

      /* Check periodic boundary conditions */
      if(Inew < 0)
        Inew = L-1;
      else if(Inew >= L)
        Inew = 0;

      if(Jnew < 0)
        Jnew = L-1;
      else if(Jnew >= L)
        Jnew = 0;

      /* Update the energy */
      Energy += J * mag[i][j] * mag[Inew][Jnew];
    }
/*Calculate the contribution from the field H */
Energy += 2*H*mag[i][j];
       }
```

```
/* Account for double counting */
Energy /= 2.0;

/* Monte Carlo Initialization */
for(i=0;i<warm;i++){
     for(j=0;j<L*L;j++){
 /* Pick a random site */
 s = (int) (drand48()*L);
 d = (int) (drand48()*L);

 Etemp = 0.0;

 /* Loop over nearest neighbors */
 for(k=0;k<4;k++){
      Inew = s + Inn[k];
      Jnew = d + Jnn[k];

      /* Check periodic boundary conditions */
      if(Inew < 0)
Inew = L-1;
```

```
      else if(Inew >= L)
Inew = 0;

      if(Jnew < 0)
Jnew = L-1;
      else if(Jnew >= L)
Jnew = 0;

      Etemp += -J*mag[s][d]*mag[Inew][Jnew];
    }

 /* Calculate change in energy */
 deltaE = 2.0*Etemp - 2*H*mag[s][d];

 /* Metropolis criteria */
 if(deltaE < 0){
      mag[s][d] = -mag[s][d];
      Energy += deltaE;
      accept++;
      move++;
```

```
    }
  else if(drand48()<exp(-beta*deltaE)){
      mag[s][d] = -mag[s][d];
      Energy += deltaE;
      accept++;
      move++;
    }
  else
    move++;
}
   }

  /*************************
   * The Run               *
   *************************/
  for(i=0;i<sweeps;i++){
     for(j=0;j<L*L;j++){
   s = (int) (drand48()*L);
   d = (int) (drand48()*L);
```

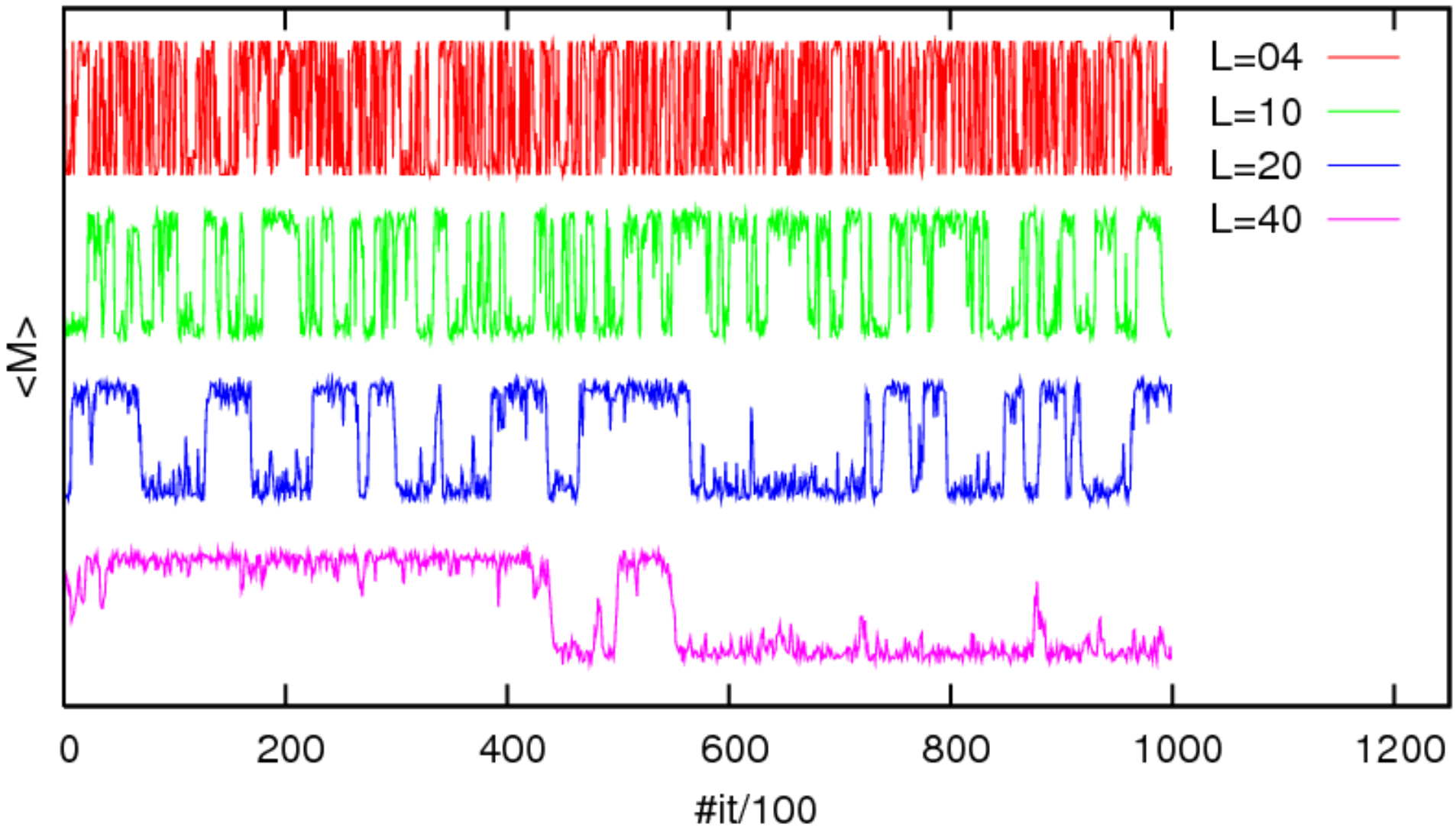
```
 Etemp = 0.0;

 for(k=0;k<4;k++){
      Inew = s + Inn[k];
      Jnew = d + Jnn[k];

      if(Inew < 0)
Inew = L-1;
      else if(Inew >= L)
Inew = 0;

      if(Jnew < 0)
Jnew = L-1;
      else if(Jnew >= L)
Jnew = 0;

      Etemp += -J*mag[s][d]*mag[Inew][Jnew];
   }

 deltaE = 2.0*Etemp - 2*H*mag[s][d];
```

```
    if(deltaE < 0){
        mag[s][d] = -mag[s][d];
        Energy += deltaE;
        accept++;
        move++;
      }
    else if(drand48()<exp(-beta*deltaE)){
        mag[s][d] = -mag[s][d];
        Energy += deltaE;
        accept++;
        move++;
      }
    else
      move++;
}
      /* COLLECT DATA HERE */
      printf ("%9.6lf  %9.6lf\n", mag_av(mag,L), Energy/(L*L));
    }

  /*****************************
```

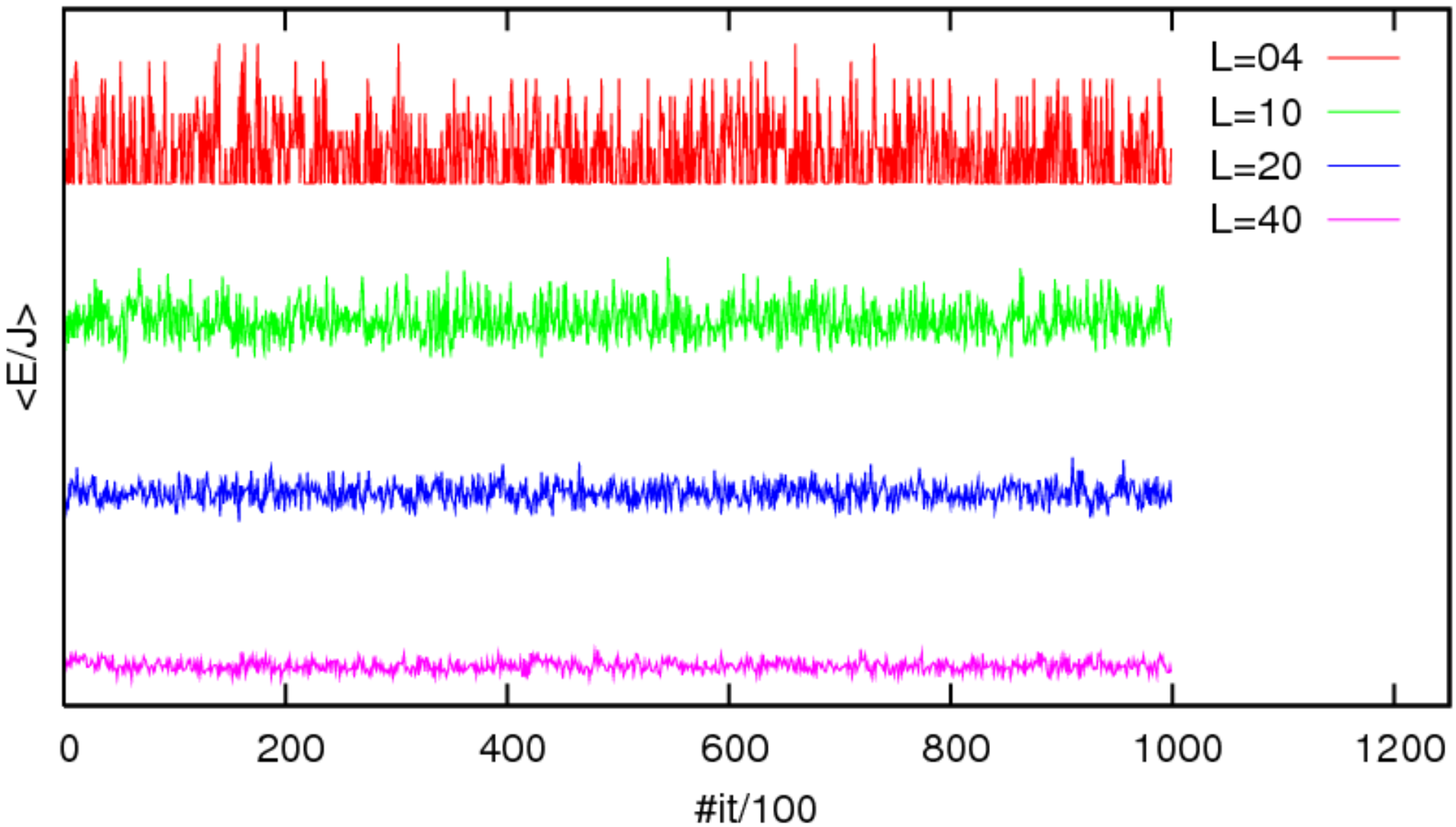
```
    * Output data and shut down *
    ****************************/

  /* OUTPUT DATA HERE */
  if (print_conf>0)
    outputmag(mag, L);

  /*Ratio of accepted spin flips compared to total attempted flips*/
  printf("# Acceptance rate:  %2.1f %%\n", 100.0*accept/move);
}
```

Raw data: traces, covariance and autocorrelation time

Trace: magnetization for $T = 2.27 J/k_B \approx T_C$ (10^5 sweeps)

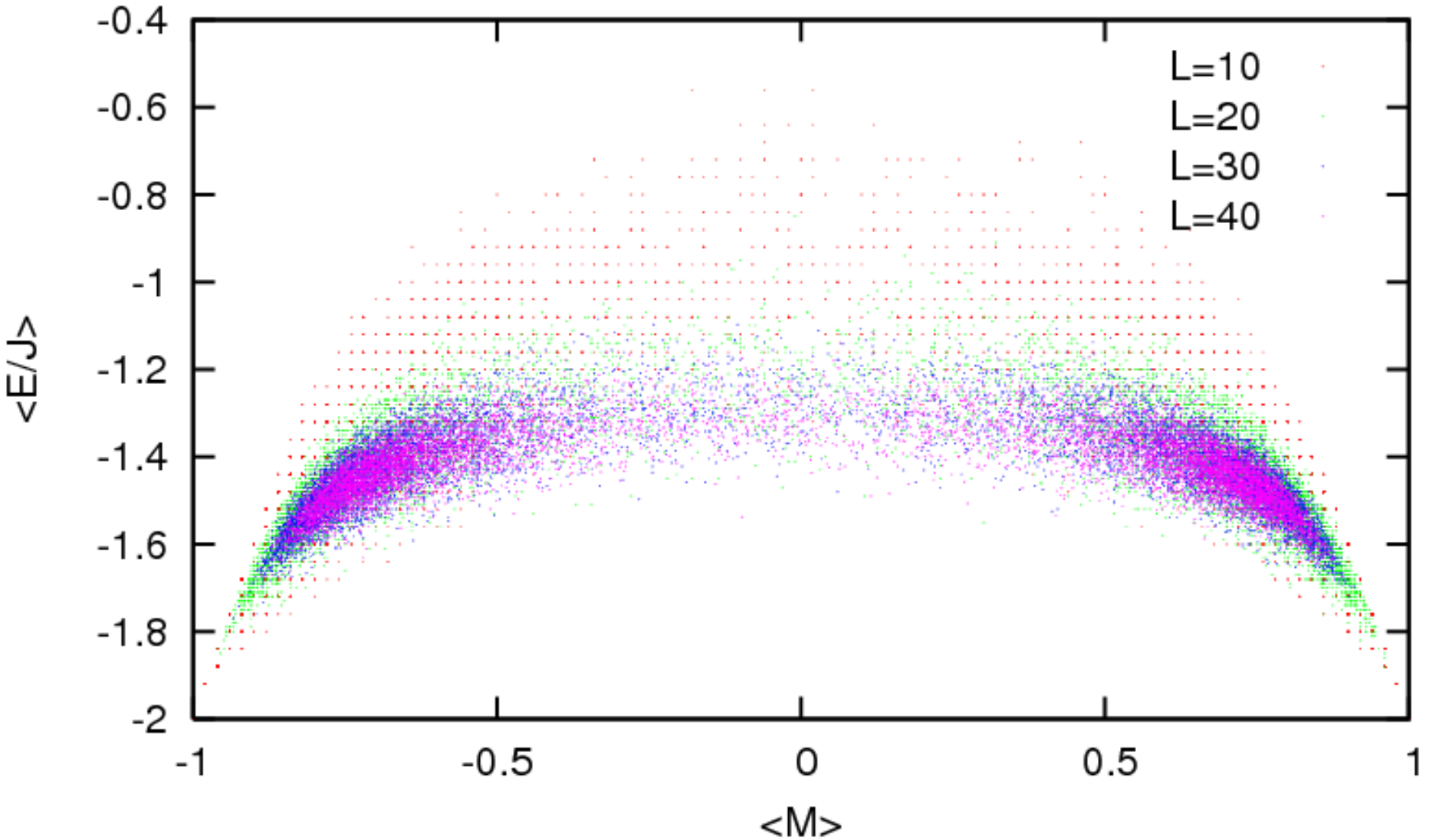


Trace: energy for $T = 2.27 J/k_B \approx T_C$ (10^5 sweeps)



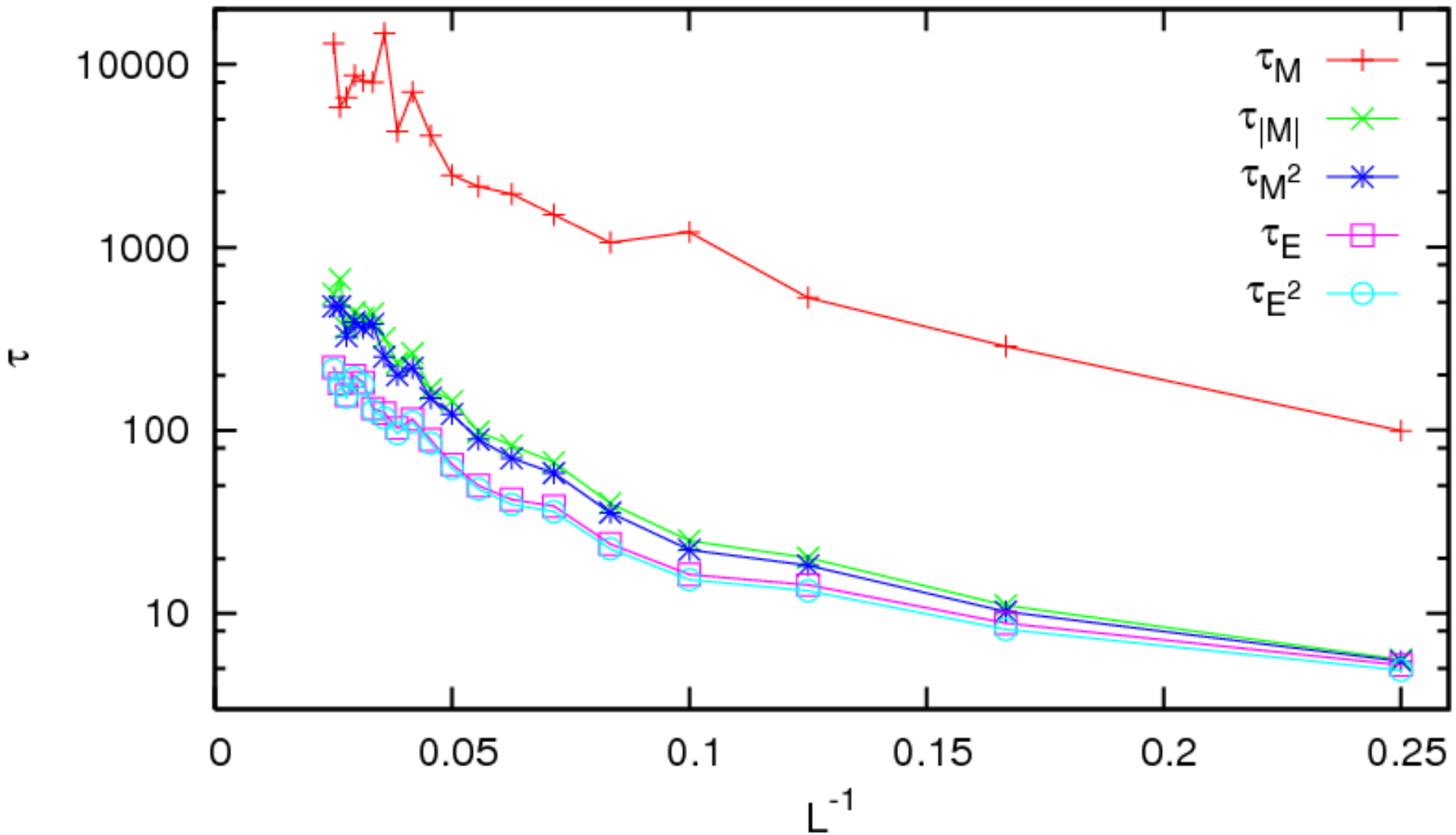
Very different shapes for traces of M and E; here less variation for larger L.

Distribution of data and covariance for $T = 2.27 J/k_B \approx T_C$ (10^6 sweeps)



Low energy $\rightsquigarrow$ large M, but flat region for $|< M >| \lesssim 0.6$

Autocorrelation times for $T = 2.27 J/k_B \approx T_C$ (10^5 sweeps)



Autocorrelation times important, specific to observable, increase with L.

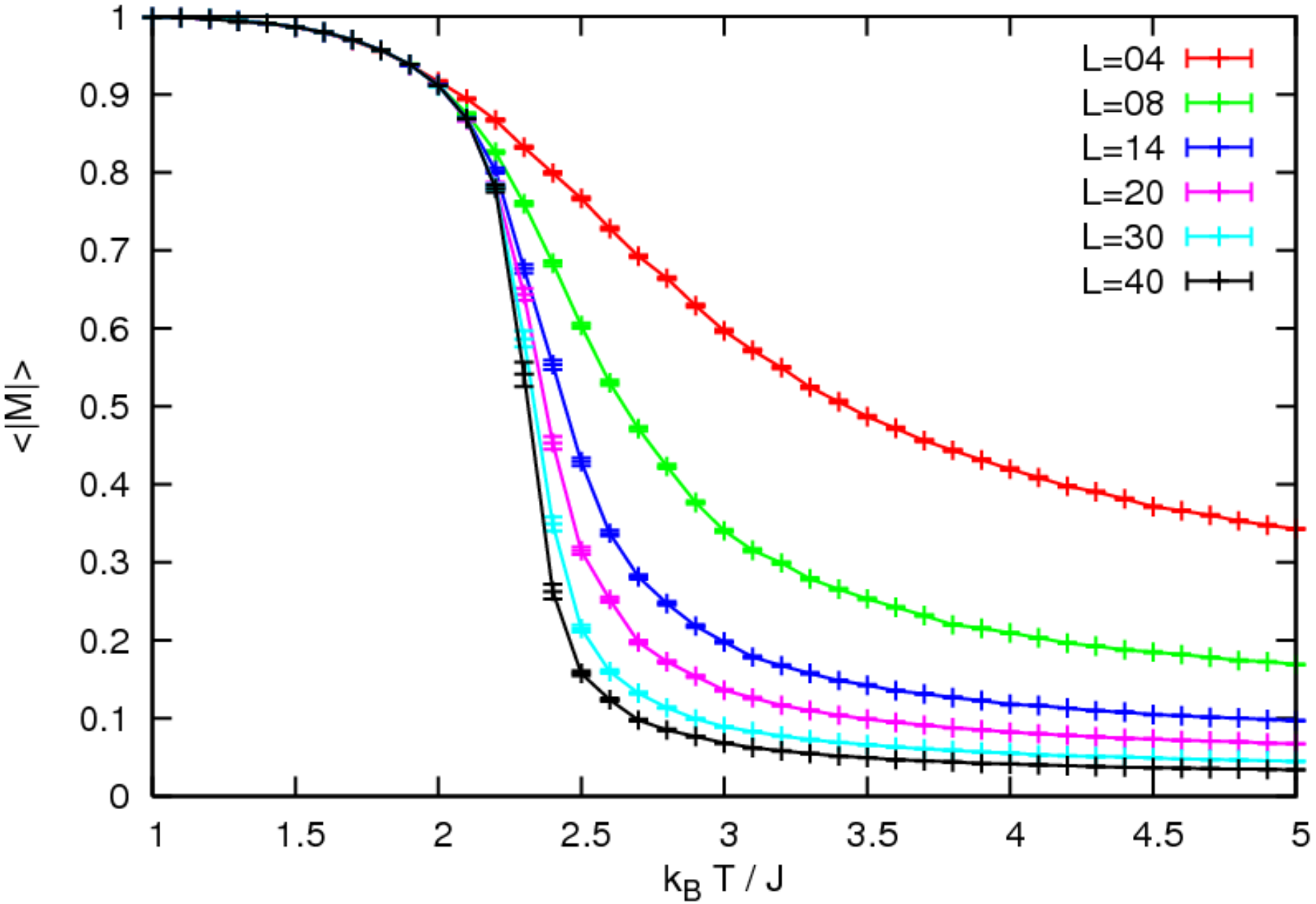
Simple observables: $|M|$, M^2, E, E^2

The following results have been obtained from Monte Carlo simulations of the 2d square lattice Ising model using single-spin-flip Metropolis updates.

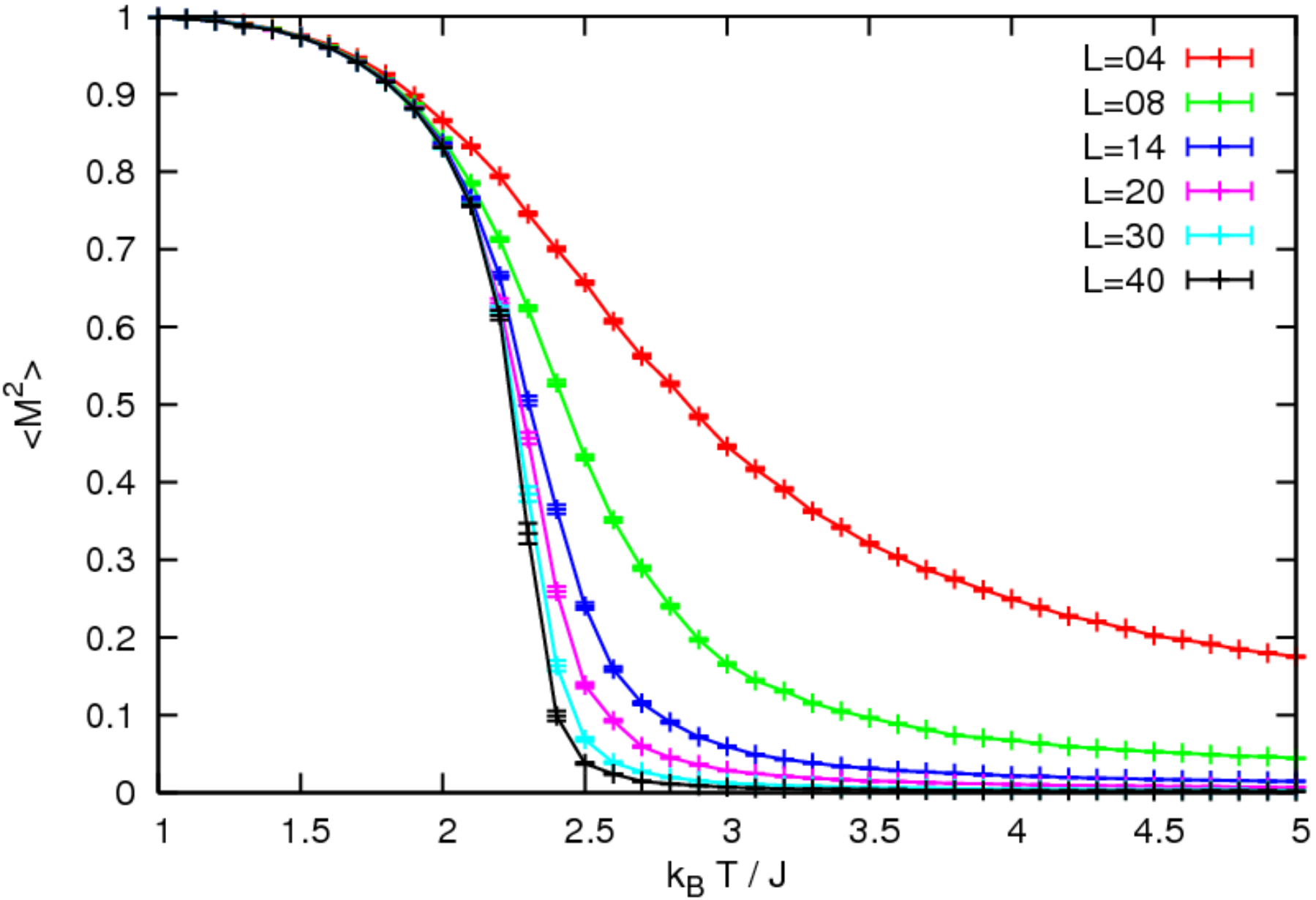
Let us first look at the obvious observables: magnetization and energy (plus their squares).

All observables are scaled per number of lattice sites, e.g. $\langle E \rangle \equiv \langle H \rangle / N; \quad N = L^2$

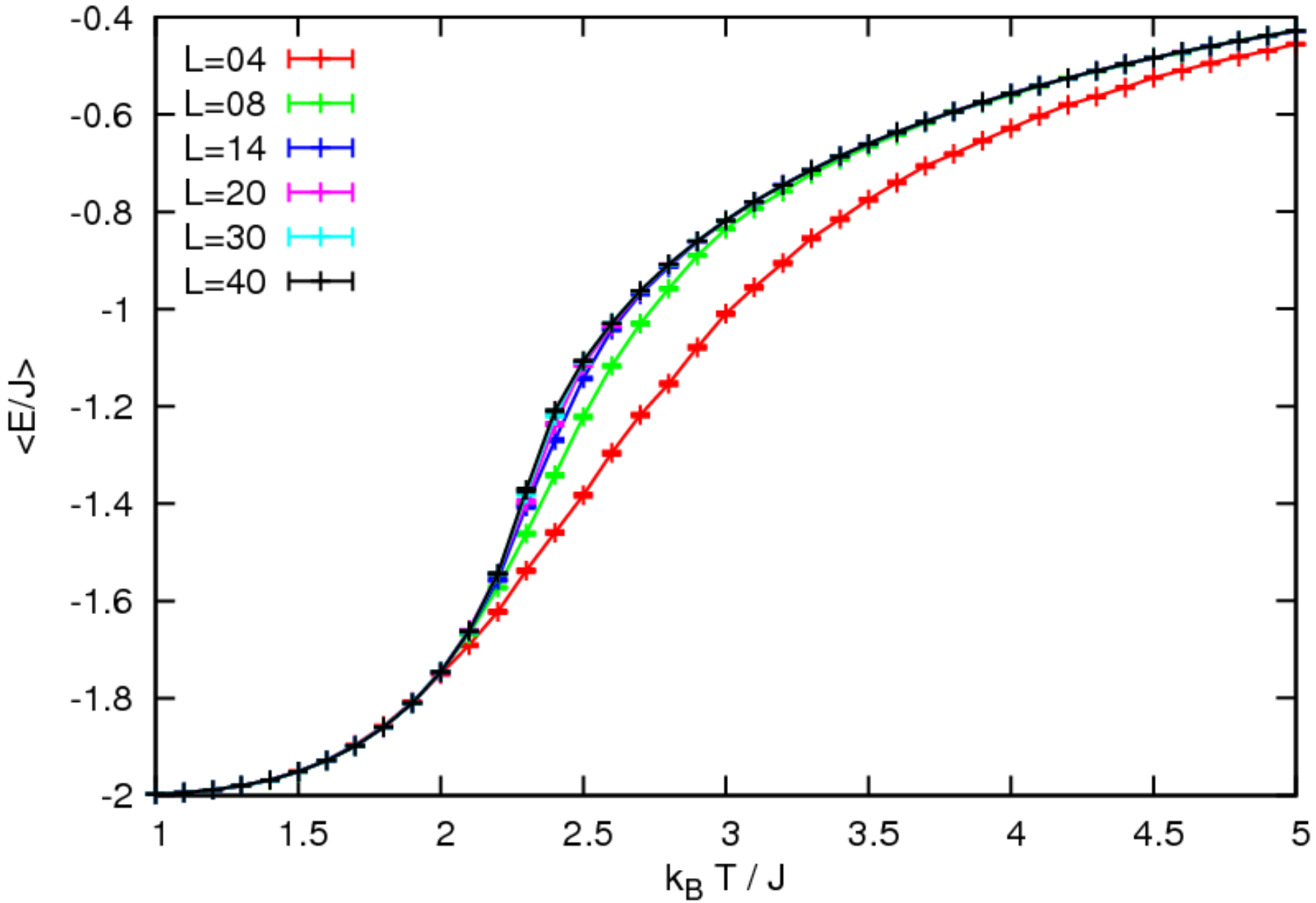
Magnetization (10^5 sweeps)



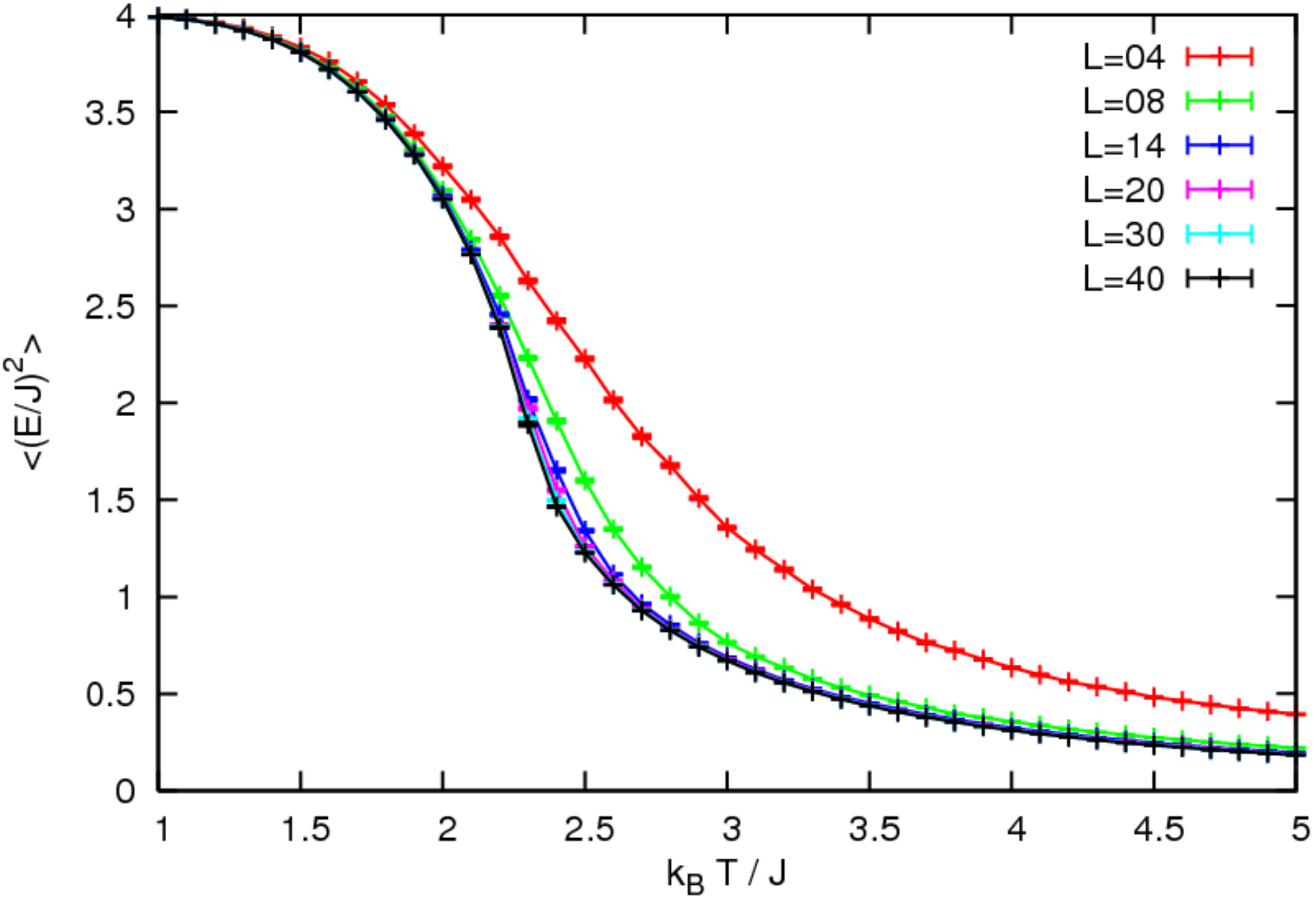
Squared magnetization (10^5 sweeps)



Energy (10^5 sweeps)



Squared energy (10^5 sweeps)



Smart observables: susceptibility, specific heat & Binder's cumulant

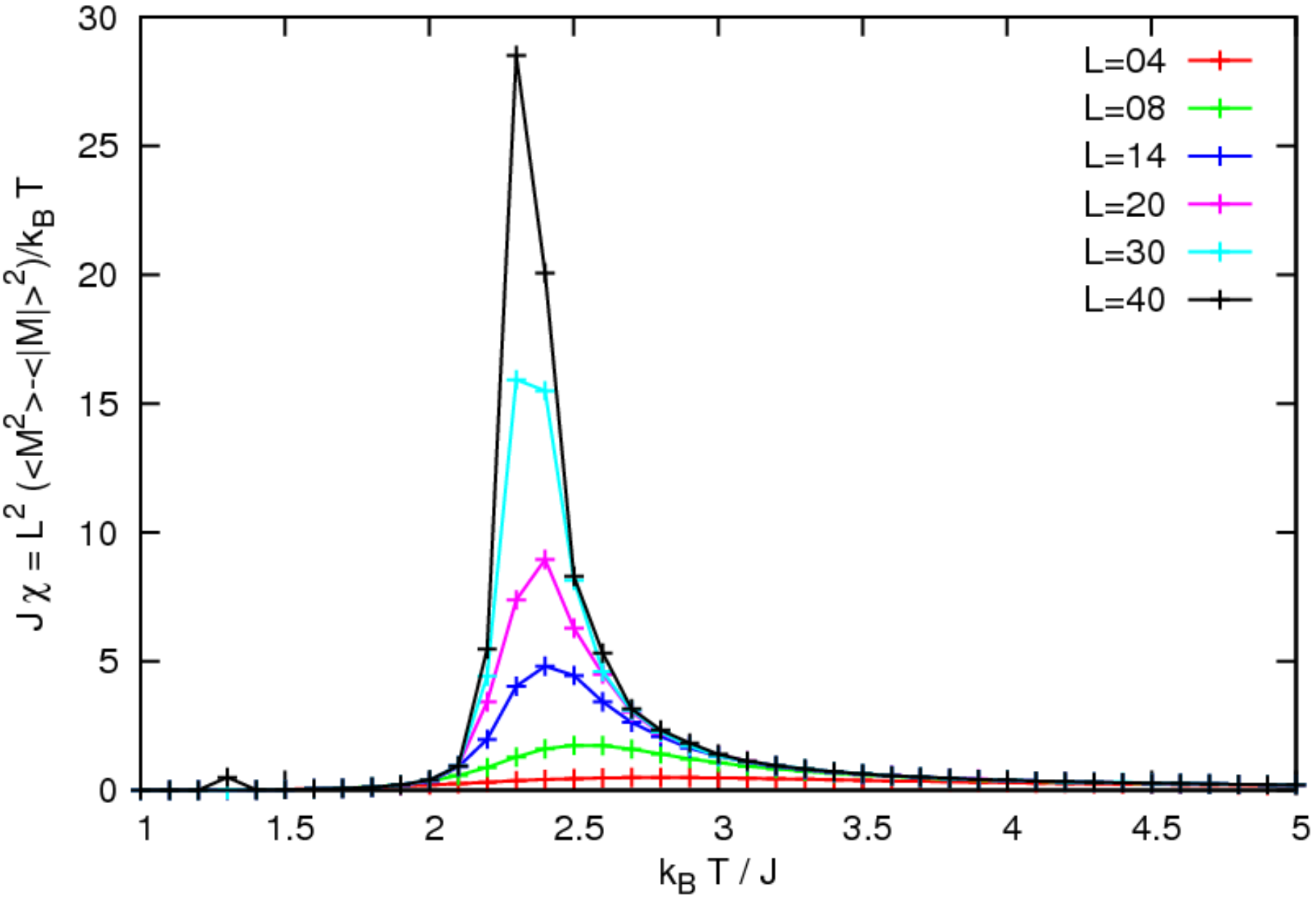
Now we turn to observables which should be particularly suited for determining the Curie temperature T_c.

Magnetic susceptibility: $k_B T \chi = \langle M^2 \rangle - \langle M \rangle^2$

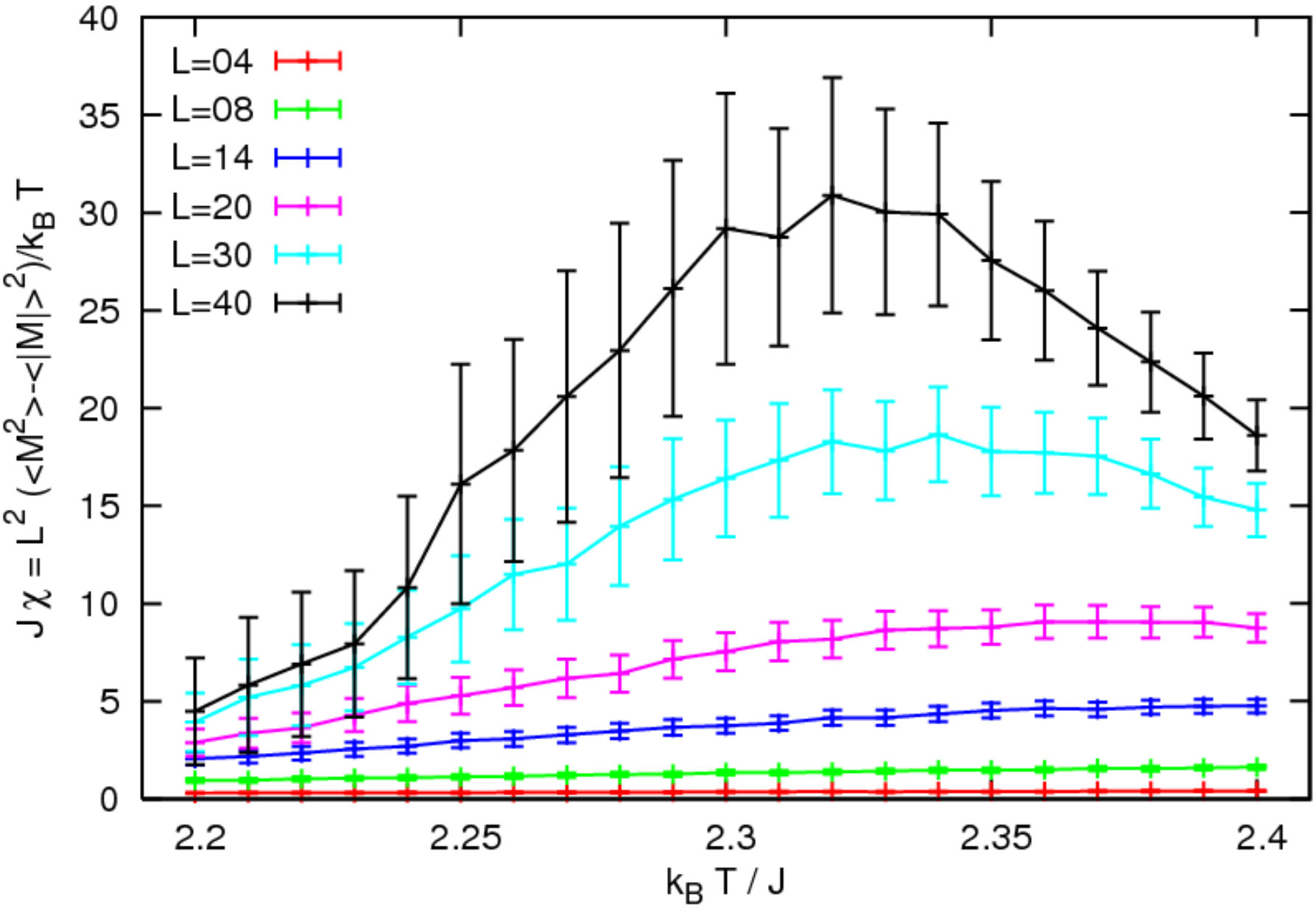
Specific heat: $k_B T^2 C_V = \langle H^2 \rangle - \langle H \rangle^2$

For the moment, only the data is used, no additional input like critical exponents.

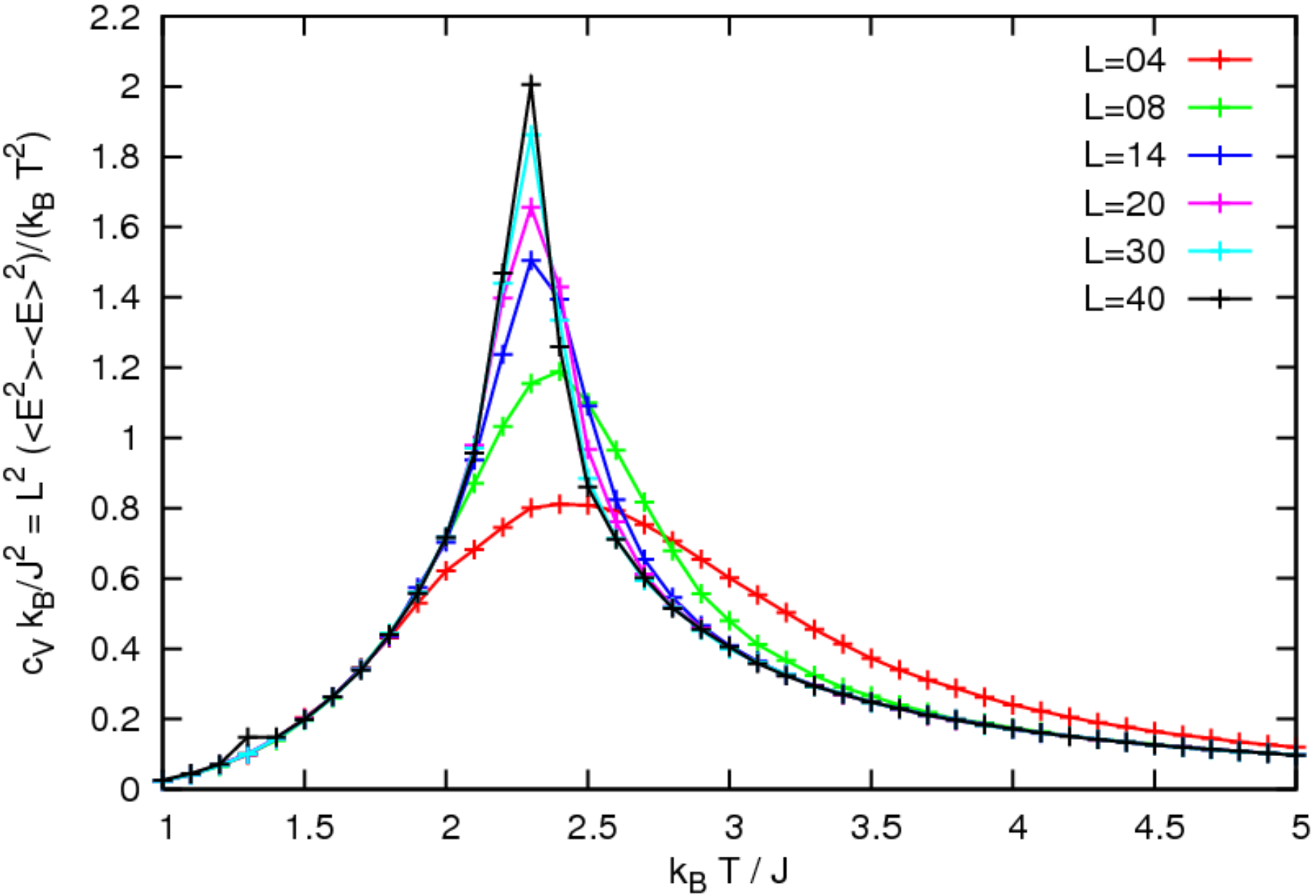
Magnetic susceptibility (10^5 sweeps)



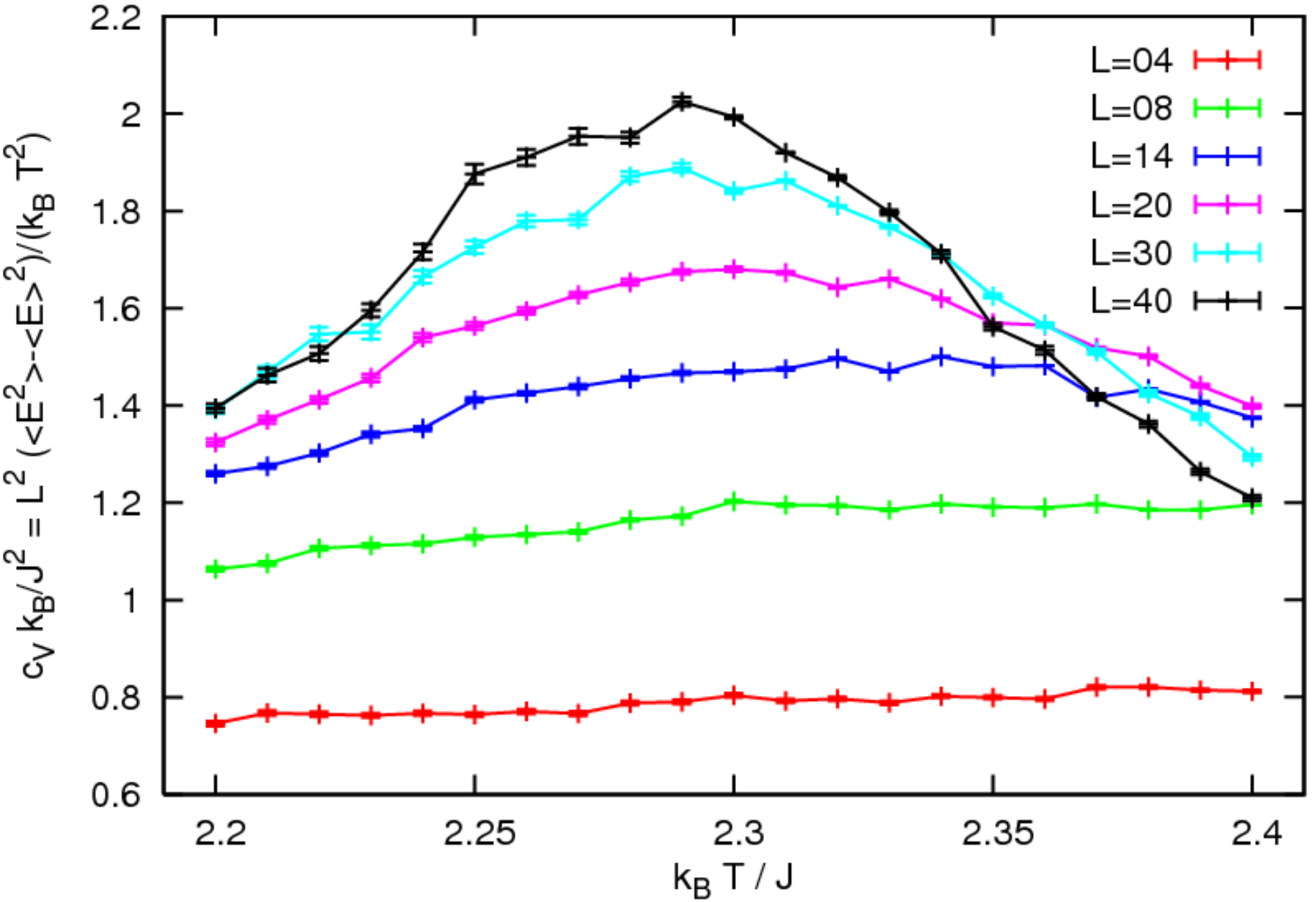
Magnetic susceptibility near T_c (10^6 sweeps)



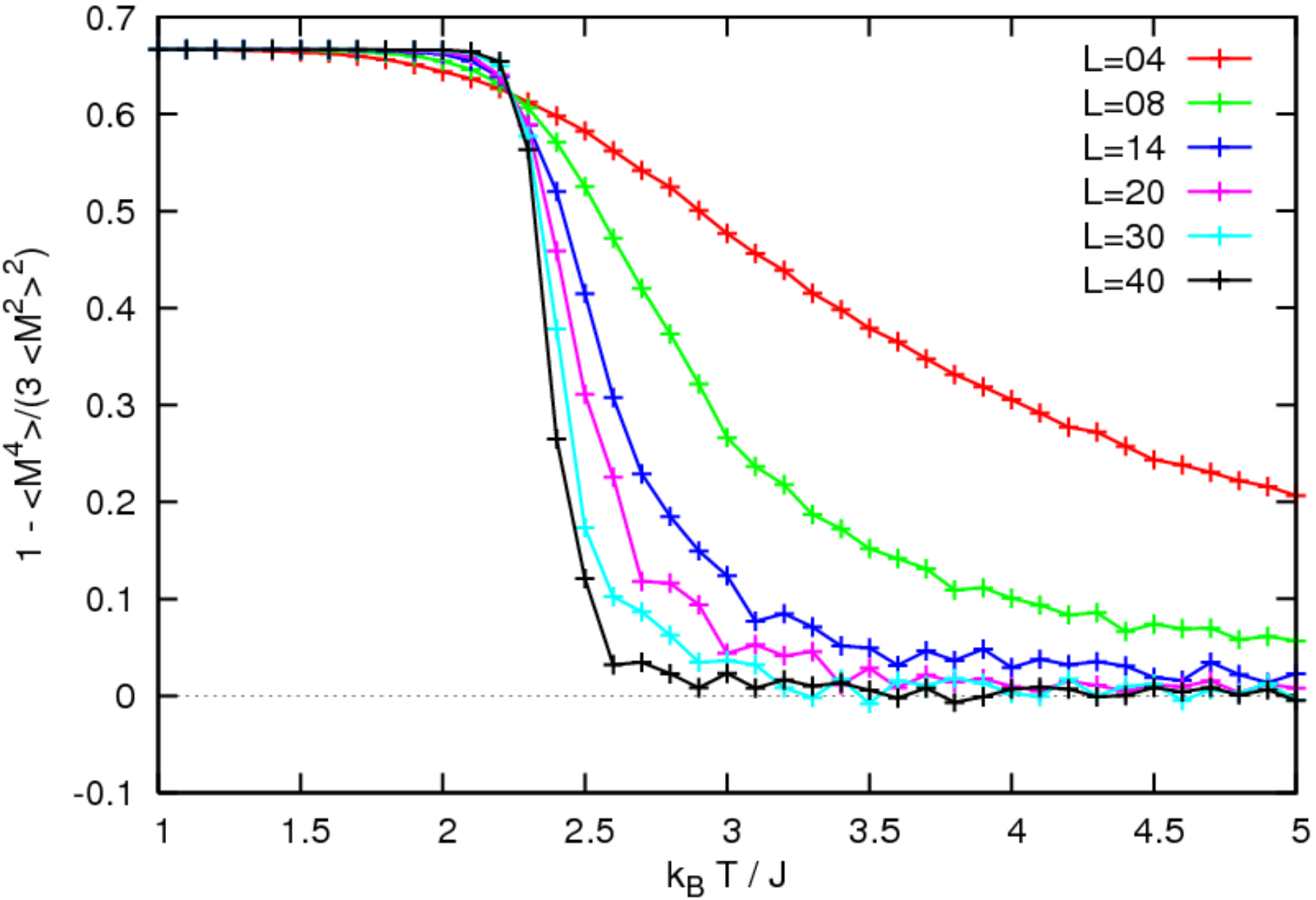
Specific heat (10^5 sweeps)



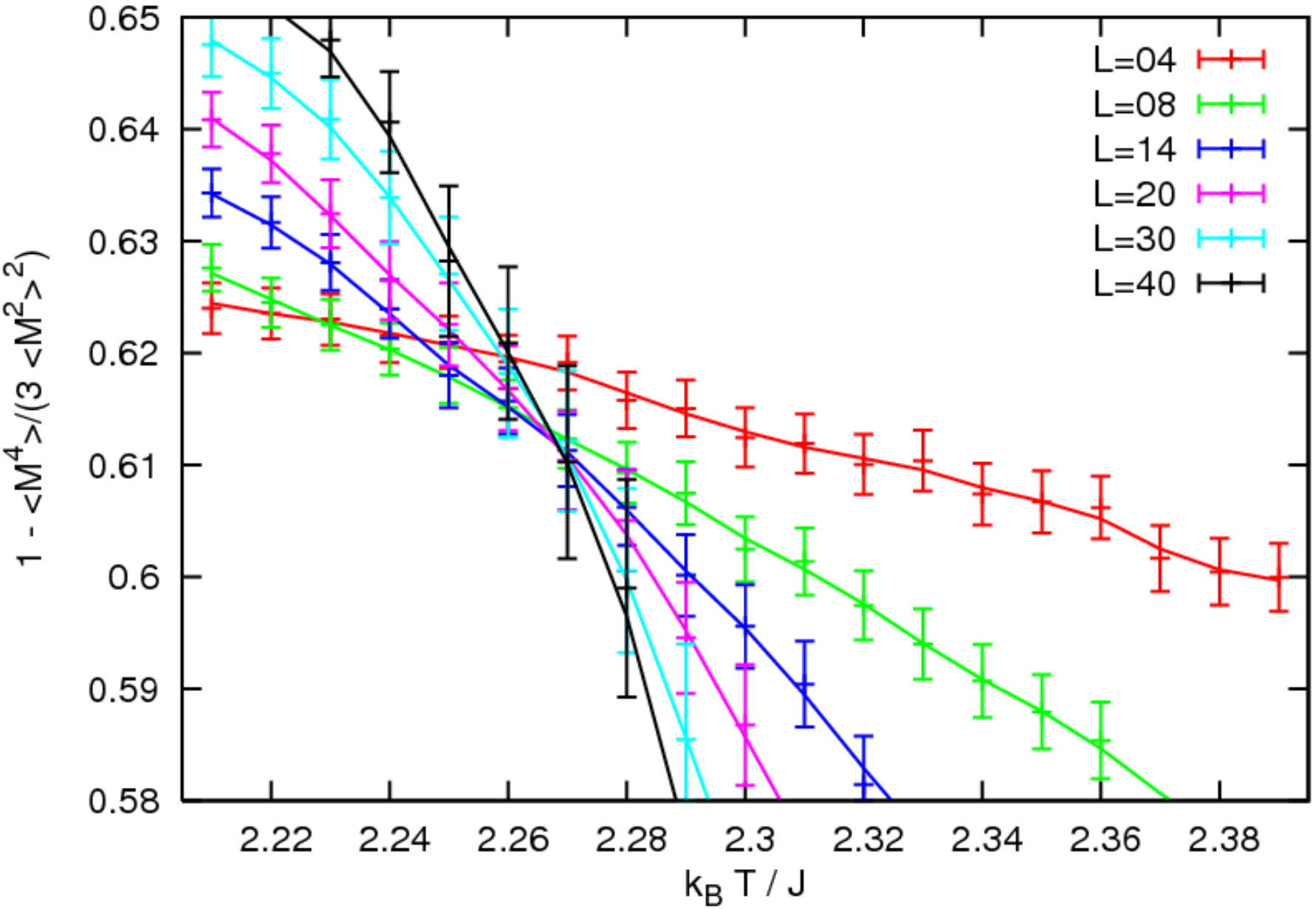
Specific heat near T_c (10^6 sweeps)



Binder's cumulant (10^5 sweeps)



Binder's cumulant near T_c (10^6 sweeps)



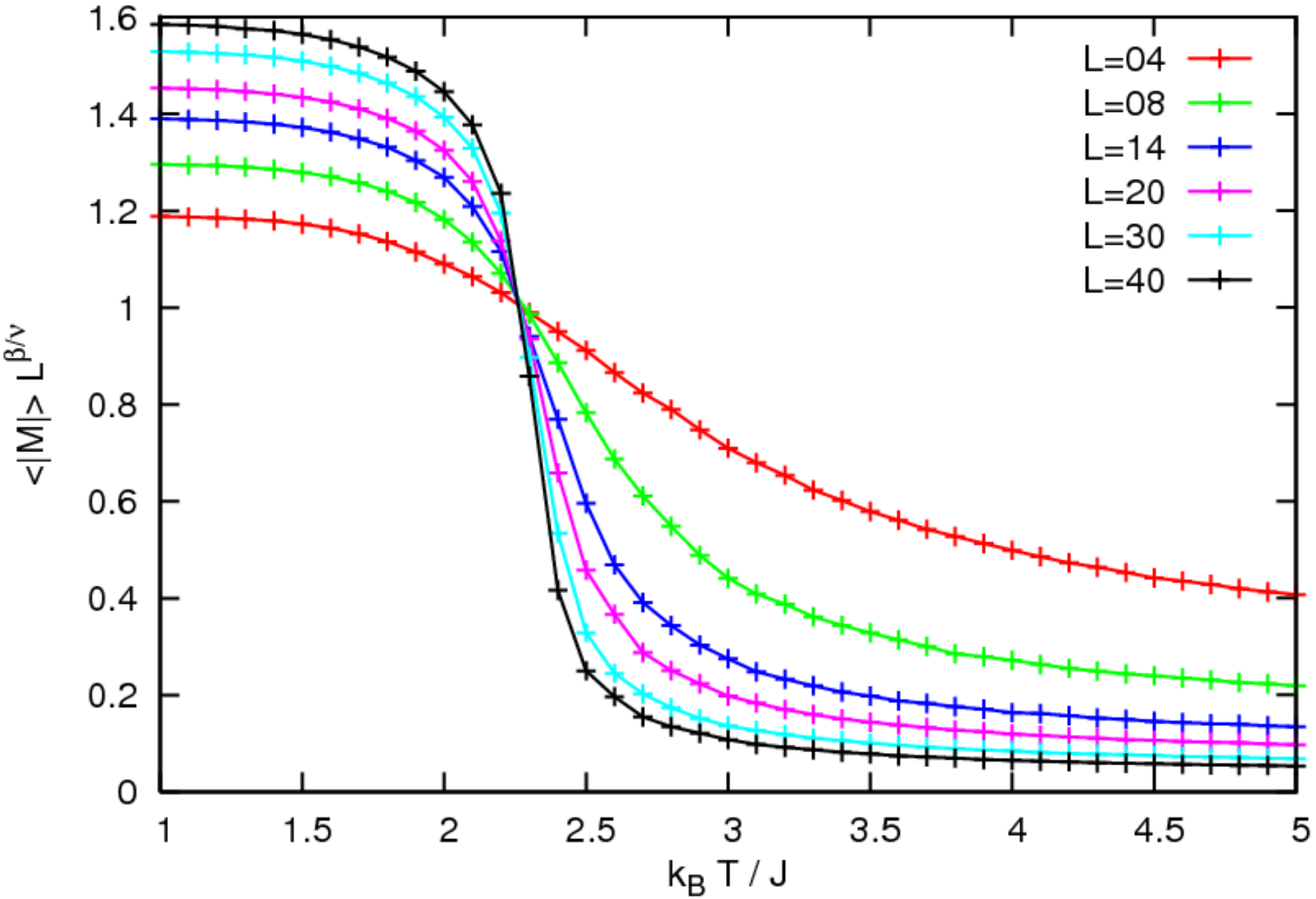
Finite-size scaling

Up to now we have only used the data and no specific analytic insight (note that Binder's cumulant and similar constructions would be useful in any dimension and for many models including the Heisenberg model).

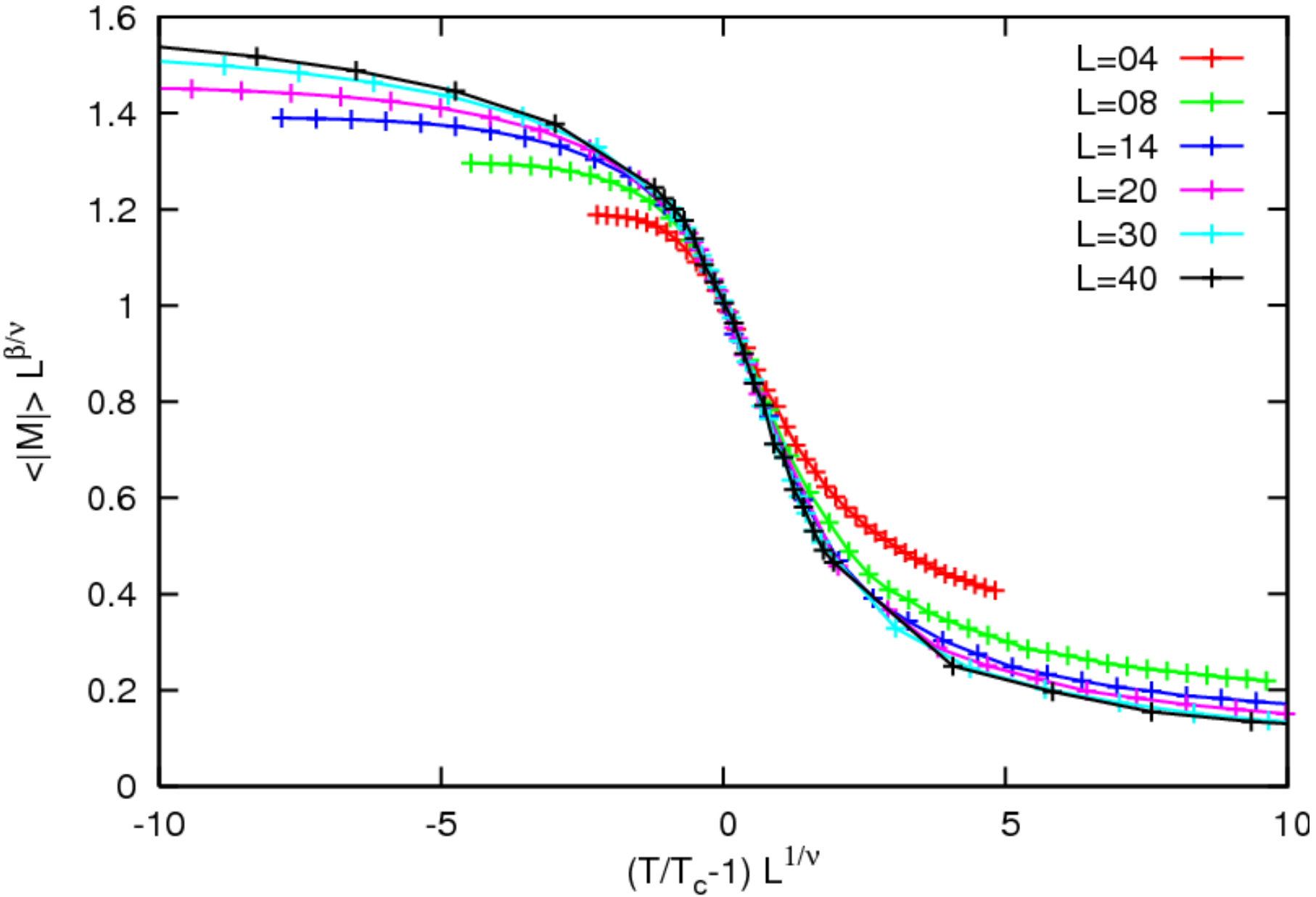
Let us now use our knowledge about scaling properties of the Ising model in 2 dimensions (valid also for honeycomb and triangular lattice):

dynamical exponents: $\nu = 1; \quad \beta = 1/8; \quad \gamma = 7/4$

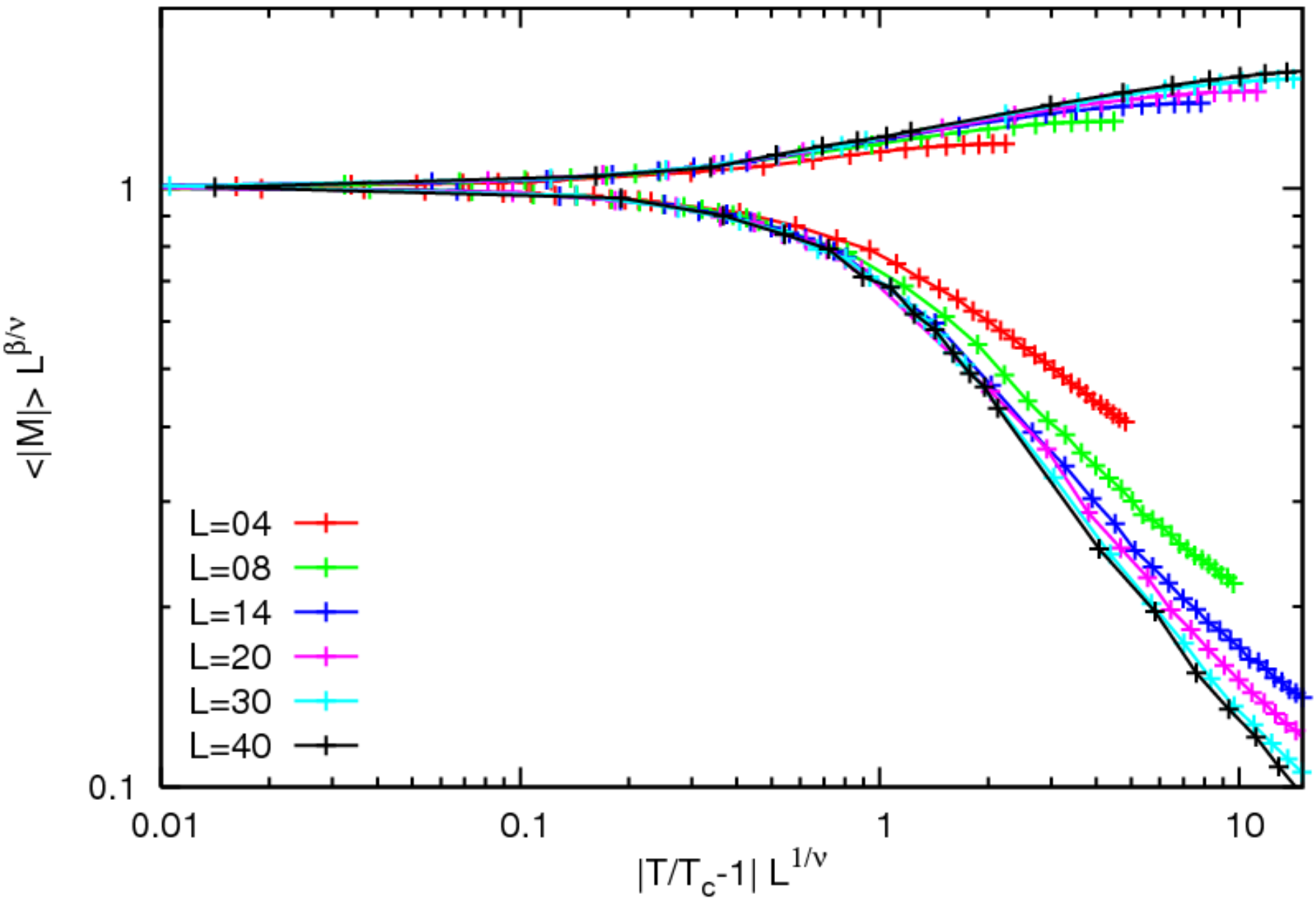
Scaled magnetization vs unscaled T (10^5 sweeps)



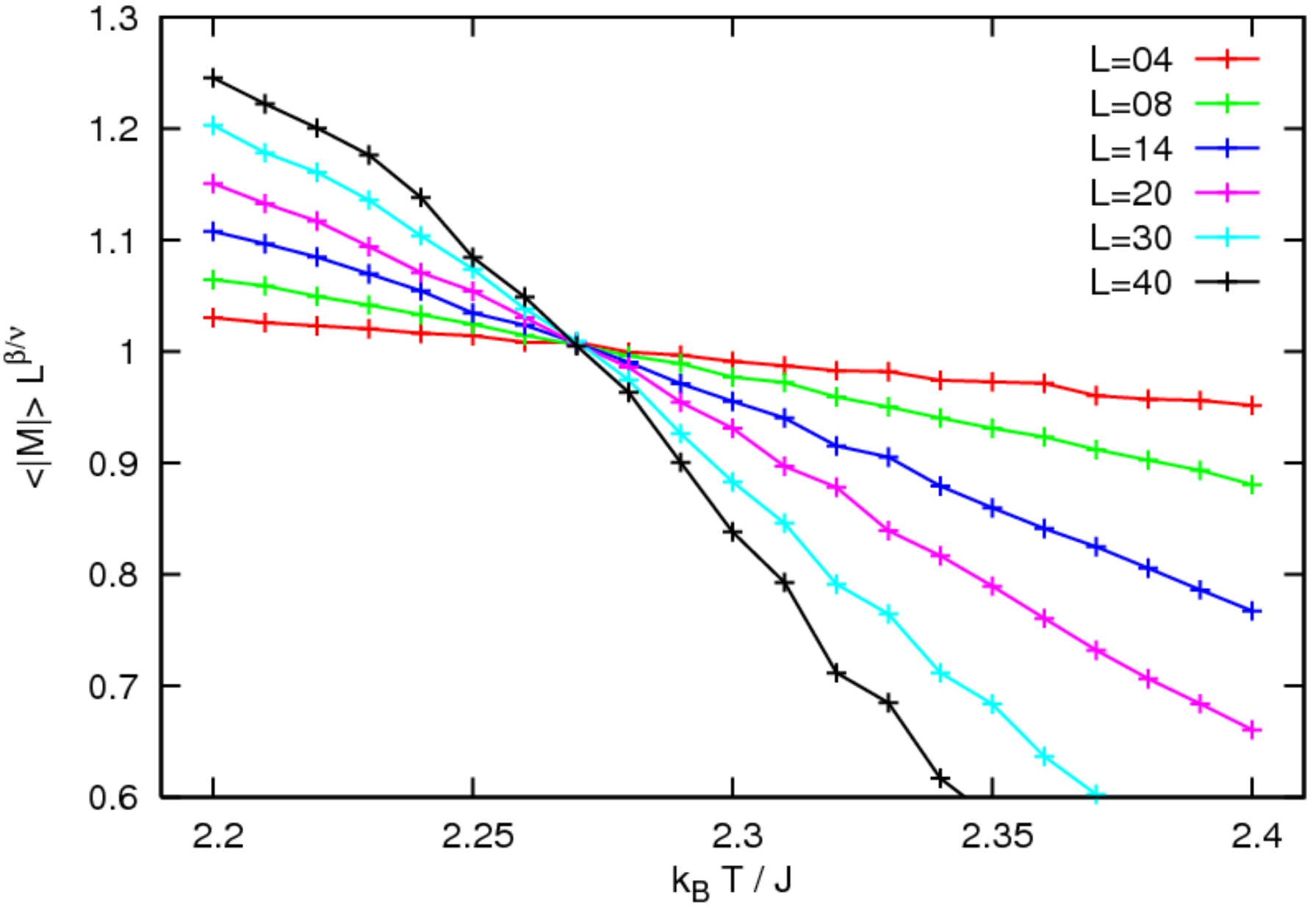
Scaled magnetization vs scaled T (10^5 -10^6 sweeps)



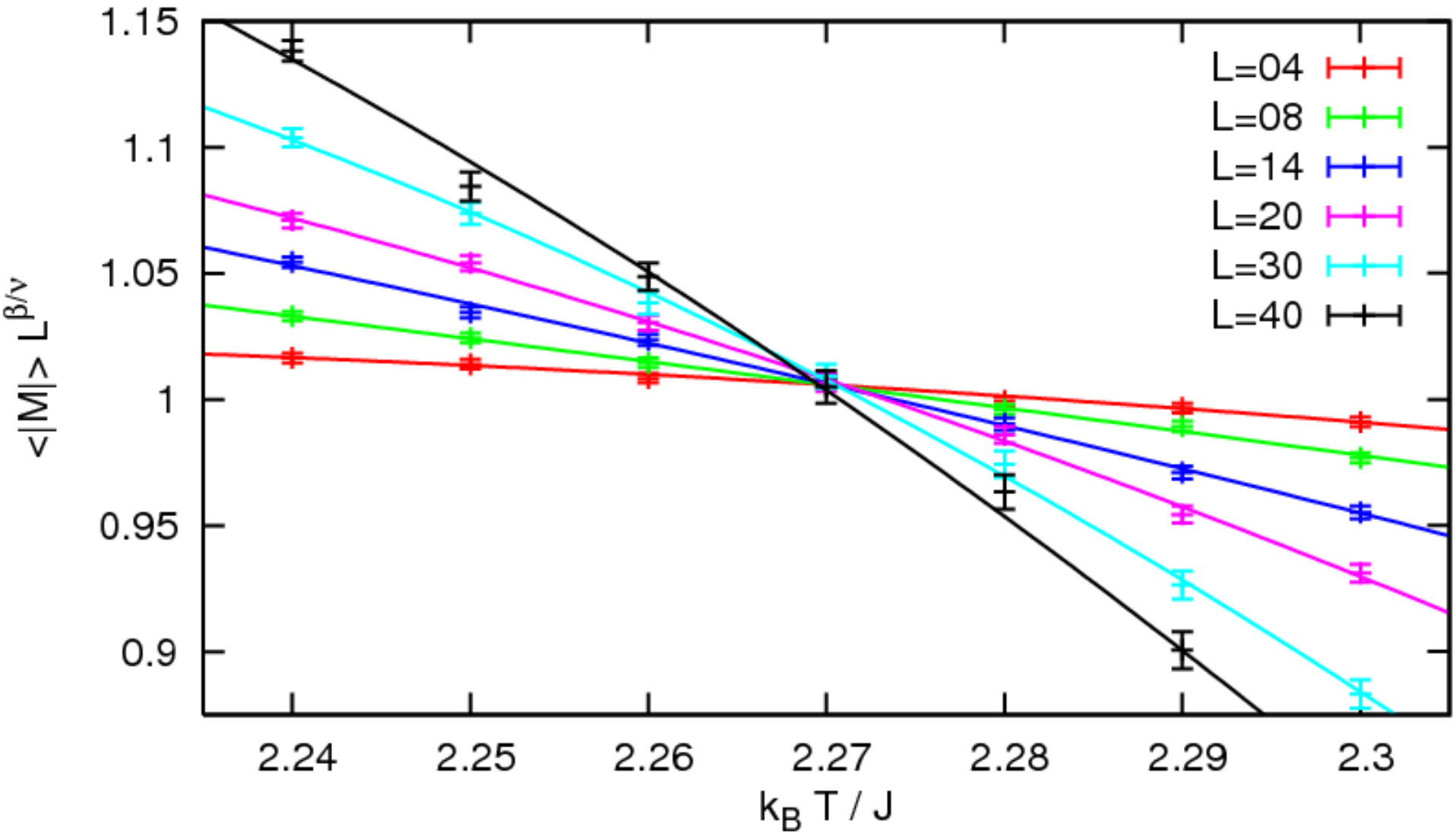
Scaled magnetization vs scaled T (10^5 -10^6 sweeps) - logarithmic



Scaled magnetization vs unscaled T (10^6 sweeps)

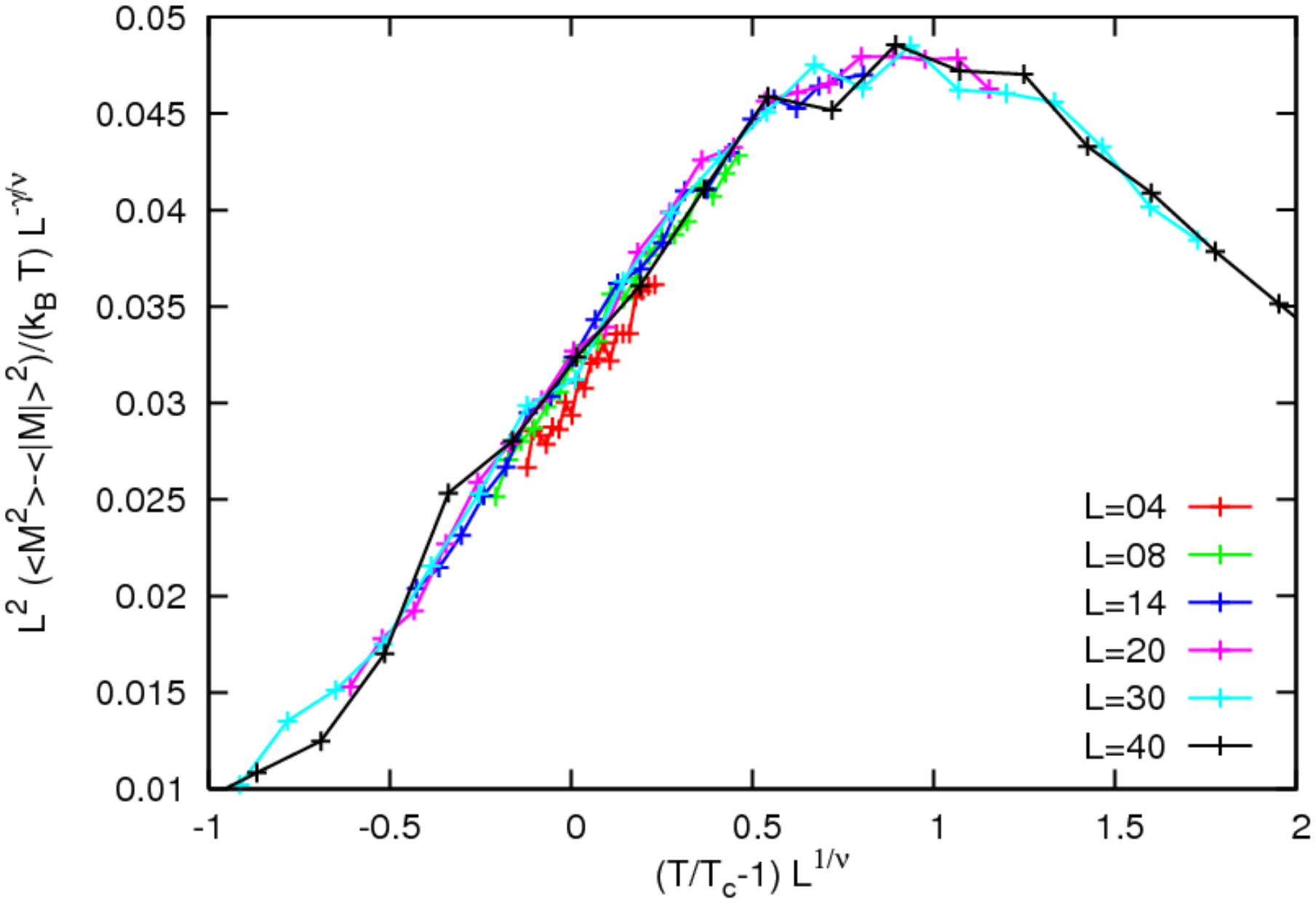


Scaled magnetization vs unscaled T, smoothed (10^6 sweeps)



Incredibly accurate determination of T_c (on relative 10^{-3} level even for $L \leq 30$)!

Scaled susceptibility vs scaled T (10^6 sweeps)



Scaled susceptibility vs scaled T (10^6 sweeps) - logarithmic