Übung 06: Quanten Isingmodell

Tobias Blesgen und Leonardo Thome

21.07.2021

Einführung

Das Isingmodell beschreibt die Gitterinteraktionen für Wellenausbreitungen. Um die Ausbreitung einer transversalen Welle auf einem eindimensionalen Gitter zu beobachten, schreiben wir den Hamilton-Operator cer N Gitterpunkte als:

$$H = \sum_{i=0}^{N-2} \sigma_i^x \otimes \sigma_{i+1}^x + g \sum_{i=0}^{N-1} \sigma_i^z. \tag{1}$$

Wir werden ihn auf Zustände basierend auf der Basis des Tensorprodukts aller Einspin-Zustände ($|00..00\rangle$ bis $|11..11\rangle$) anwenden um ihn auf seine Grundzustandsenergie und -wellenfunktion, zu untersuchen. Das g stellt hierbei die Kopplung an ein externes Feld.

Anschließend werden wir für $g = 1$ den Phasenübergang für die Magnetisierung

$$M = \frac{1}{N} \langle\psi| \sum_{i=0}^{N-1} \sigma_i^z |\psi\rangle \tag{2}$$

betrachten.

Implementation des numerischen Verfahrens

```
#include<Rcpp.h>
#include<vector>
#include<algorithm>
#include<math.h>
#include<random>

using namespace Rcpp;
using namespace std;

// Berechnungsschritt des Skalierungsvektors: m = 0, pos = 0, dim(vektor) = n
void Hg_Rekursiv(const int n, int m, int l, int pos, std::vector<double>& vektor){
  vektor[pos] = (double)(n - l);
  for (int i = m; i < n; i++){
    Hg_Rekursiv(n, (i + 1), l+2 , pos + pow(2,i), vektor);
  }
```

```
}

// Skalierungsvektor (rechtes H)
std::vector<double> g_Vektor(const int n){
  std::vector<double> vektor(pow(2,n));
  Hg_Rekursiv(n, 0, 0 , 0, vektor);
  return vektor;
}

// Normierungsbestimmung
double v_Norm(vector<double> const& u) {
    double sum = 0.;
    for (int i = 0; i < u.size(); ++i) {
        sum += u[i] * u[i];
    }
    return sqrt(sum);
}

// Zufälliger Startvektor
std::vector<double> random_V(const int l){
  std::vector<double> vektor(l);
  random_device rd;
  mt19937 gen(rd());
  uniform_real_distribution<> dis(0,1.0);
  for (int i = 0; i < l; ++i) {
    vektor[i] = dis(gen);
  }
  double norm = v_Norm(vektor);
  for (int i = 0; i < l; i++){
    vektor[i] /= norm;
  }
  return vektor;
}

// Anwendung von H
std::vector<double> H(std::vector<double> vektor, std::vector<double> gv, const double g, const int n){
  int size = pow(2,n);
  std::vector<double> neu(size);
  for (int i = 0; i < size; i++){
    neu[i] = gv[i]*g*vektor[i];
  }
  for (int i = 0; i < pow(2,n-2); i++){
    for (int j = 0; j < 4; j++){
      neu[i*4+j] += vektor[(i+1)*4-j-1];
      //Rprintf("- %d -\n",(i+1)*4-j-1 );
    }
  }
  for (int i = 0; i < (n-2); i++){
    int mittel = pow(2,n-i-2);
    for (int grob = 0; grob < pow(2,i); grob++){
      for (int medium = 0; medium < 4; medium++){
        for (int j = 0; j < pow(2,n-i-2); j++){
          neu[(grob*4+medium)*mittel+j] += vektor[j+(4*(grob)+3-medium)*mittel];
```

```
          //Rprintf("i=%d j=%d: %d; %d \n", i, j, (grob*4+medium)*mittel+j, j+(4*(grob)+3-medium)*mitte
        }
      }
    }
  }
  return neu;
}

// Hauptfunktion zur Eigenwertbestimmung
//[[Rcpp::export]]
std::vector<double> eigen(const int n, const double g, const int max){
  const int l = pow(2,n);
  std::vector<double> vektor = random_V(l);
  std::vector<double> g_vektor = g_Vektor(n);
  double norm;

  for(int i=0; i<max; i++){
    vektor = H(vektor, g_vektor, g, n);
    norm = v_Norm(vektor);
    for (int j = 0; j < l; j++){
      vektor[j] /= norm;
    }
  }
  return vektor;
}

//[[Rcpp::export]]
double eigenwert(std::vector<double> eigenvektor, const double g, const int n){
  const int l = pow(2,n);
  std::vector<double> g_vektor = g_Vektor(n);
  std::vector<double> h_vektor = H(eigenvektor, g_vektor, g, n);
  double norm = v_Norm(eigenvektor);
  double sum = 0;
  for(int j=0;j<l;j++){
      sum += eigenvektor[j]*h_vektor[j];
    }
  sum /= norm;
  return sum;
}

// Matrixausgabe
//[[Rcpp::export]]
void matrixausgabe(const int n, const double g){
  int l = pow(2,n);
  std::vector<double> vektor(l);
  std::vector<double> g_vektor = g_Vektor(n);
  for (int i = 0; i<l; i++){
    for (int j = 0; j<l; j++){
      vektor[j] = 0;
    }
    vektor[i] = 1;
    vektor = H(vektor, g_vektor, g, n);
    for (int j = 0; j<l; j++){
```

```
      Rprintf("  %d  ", (int)(vektor[j]));
    }
    Rprintf("\n");
  }
  Rprintf("\n");
}

// Anwendung von H2
std::vector<double> h_shift(std::vector<double> vektor, const double g, const int n, std::vector<double>
  std::vector<double> g_vektor = g_Vektor(n);
  std::vector<double> h_vektor = H(vektor, g_vektor, g, n);
  const int l = pow(2,n);
  std::vector<double> neu(l);
  const double eigenw = eigenwert(eigenvektor, g, n);
  for (int i = 0; i < l; i++){
    neu[i] = h_vektor[i];
    for (int j = 0; j < l; j++){
      neu[i] -= (eigenw*eigenvektor[i]*eigenvektor[j]);
    }
  }
  return neu;
}

// erogn
//[[Rcpp::export]]
std::vector<double> wellenfunktion(const int n, const double g, const int max){
  // bestimmung des größten eigenwert-vektors
  std::vector<double> grosser_vektor = eigen(n, g, max);
  // bestimmung des kleinstenm eigenwert-vektors
  const int l = pow(2,n);
  std::vector<double> vektor = random_V(l);
  std::vector<double> g_vektor = g_Vektor(n);
  double norm;
  for(int i=0; i<max; i++){
    vektor = h_shift(vektor, g, n, grosser_vektor);
    norm = v_Norm(vektor);
    for (int j = 0; j < l; j++){
      vektor[j] /= norm;
    }
  }
  return vektor;
}
```

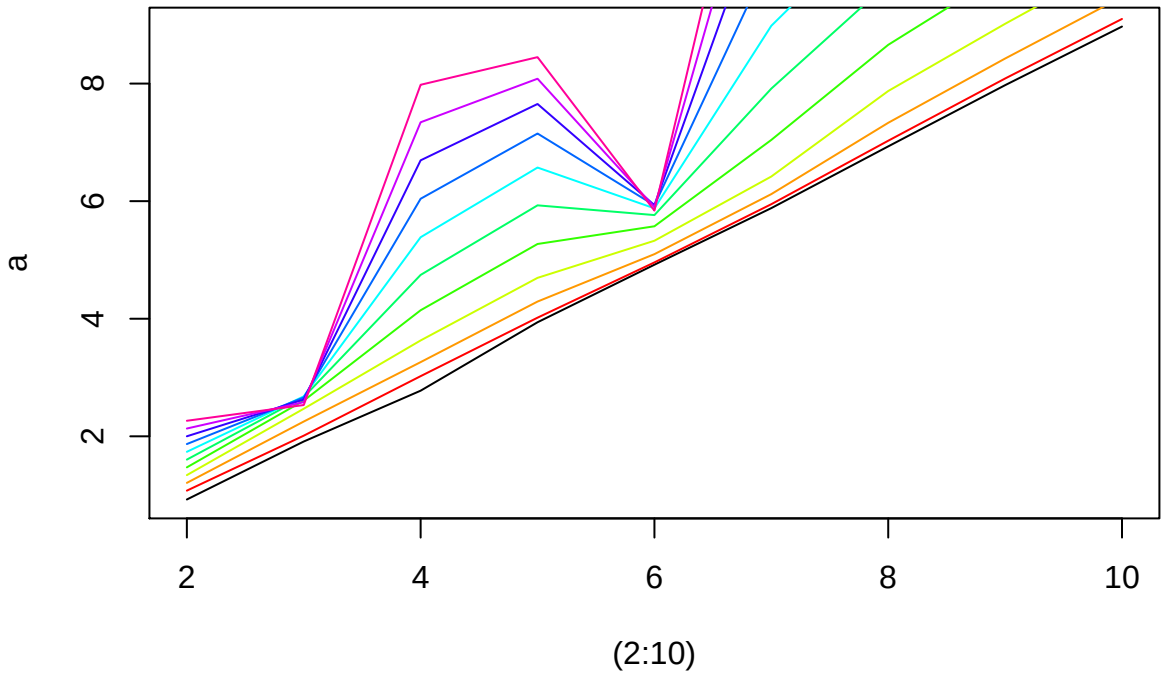
Stabilitätsüberprüfung

```
ca = 2:10
a = 2:10
d = 1:10

for (element in ca){
  b = eigen(element, 0, 1001)
  a[element-1] = eigenwert(b, 0,  element)
```

```
}
plot((2:10),a,"l")

for (color in d){
  for (element in ca){
    b = eigen(element, color/5, 1000)
    k = eigenwert(b, color/5,element)
    a[element-1] = k
  }
  lines((2:10),a,"l", col=rainbow(10)[color])
}
```




```
h =  wellenfunktion(4, 0.5, 1000)
eigenwert(h, 0.5, 4)
```

```
## [1] 3.426629
```

```
k =  eigen(4, 0.5, 1000)
eigenwert(h, 0.5, 4)
```

```
## [1] 3.426629
```

Wir wollen am Beispiel des $\gamma = 0$ Falls überprüfen, ob unsere Methodik stabil ist. Wir würden erwarten, dass die mittlere Amplitude über die Zeit konstant bleibt, und tragen hierzu die Summe aller Funktionspunkte grafisch gegen die Zeit auf:

Wie wir sehen können, bleibt die Amplitudensumme über dem betrachteten Bereich sehr konstant und wir können die Methode im Folgenden auf ein gedämpftes System anwenden.

Fazit

Durch das Umformen der DGL zur Gleichung **??** ließ sich der zeitliche Schwingungsverlauf örtlich bestimmen.

Literatur

[1] Wolfgang Demtröder, *Experimentalphysik 1, Mechanik und Wärme*, SpringerSpektrum, Auflage 8, 2018, Seite 334.