Quanten Isingmodell

Silas Müller, Wayne Ströbel

10 7 2021

Das Quanten-Isingmodell

Beim Quanten Isingmodell mit transversalem Feld sind N quantenmechanische Spins auf einem eindimensionalen Gitter angeordnet. Der Hamilton-Operator, der die Dynamik dieser Spins mit offenen Randbedingungen beschreibt, hat folgende Form:

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

Der erste Term des Hamiltonoperators ist der Wechselwirkungsterm zwischen den Spins an den Gitterpunkten i und $i+1$. Der zweite Term beschreibt ein externes magnetisches Feld mit Kopplung g. Mit den bekannten Paulimatrizen σ_i^x und σ_i^z jeweils am Gitterpunkt (Zustand) i.
Als Basis der Zustände an einem Gitterpunkt i wählen wir die Eigenbasis von σ^z, also:

$$|\Phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

Dabei gilt auf Grund der Normierung für die Amplituden α_i: $\sum_i \alpha_i^2 = 1$. Der Zustand für alle Spins ergibt sich dann aus dem Tesnorprodukt aller Einspinzustände an den Gitterpunkten 0 bis N-1 ,also $|00\ldots00\rangle$,$|00\ldots01\rangle$,$|00\ldots10\rangle$,.... Es gibt 2^N solcher Basiszustände und dementsprechend 2^N Amplituden.
Das Modell hat bei $g = 1$ einen Quanten-Phasenübergang, der in der transversalen Magnetisierung

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

sichtbar wird, wobei $|\psi\rangle$ ein Zustand des N-Spin-Systems ist.

Aufgabenstellung

Wir sollen die Grundzustandenergie, Grundzustandwellenfunktion und Magnetisierung M im Grunszustand für N=2, N=4, N=6, N=8 und N=10 für $g \in [0, 2]$ bestimmen.
Außerdem soll die Energie und die Magnetisierung als Funktion von g dargestellt werden.
Abschließend vergleichen wir die Kurven für verschiedene N, interpretieren die Ergebnisse und diskutieren das Konvergenzverhalten des Algorithmus für N = 4 und g = 0.5.

Vorüberlegungen

Es sei hier zuerst explizit auf die Vorlesung vom 7.7.2021 verwiesen, in welcher die Theorie und genauere Erläuterung zu finden ist. Hier soll noch einmal das wichtigste für den Algorithmus hervorgehoben und unsere Vorgehensweise erläutert werden.

Wir wollen zuerst den Hamiltonoperator als Matrix explizit darstellen und anschließend die Eigenwerte (also die Energien bestimmen). Für den niedrigsten Eigenwert soll dann der normierte Eigenvektor gefunden werden, was dem Grundzustand entspricht. Dadurch sollen die gewünschten Ergebnisse und Auftragungen erreicht werden.
Dafür verwenden wir eineseits obige Definitionen, benötigen jedoch auch das Tensorprodukt um σ_i^z und $\sigma_i^z \otimes \sigma_{i+1}^z$ zu berechnen.
So befindet sich σ_i an der i-ten Stelle (mit 1 als 2x2-Einheitsmatrix):

$$1_0 \otimes 1_2 \otimes ... \otimes 1_{i-1} \otimes \sigma^z \otimes 1_{i+1} \otimes ... \otimes 1_{N-1} = \sigma_i^z$$

bzw.

$$1_0 \otimes 1_2 \otimes ... \otimes 1_{i-1} \otimes \sigma^x \otimes \sigma^x \otimes 1_{i+2} \otimes ... \otimes 1_{N-1} = \sigma_i^x \otimes \sigma_{i+1}^x$$

Grundzustandenergie, Grundzustandswellenfunktion

Wir gehen nun wie oben vor und wollen den Hamiltonoperator, sowie seine Eigenwerte und entsprechenden Eigenvektoren abhängig von g konstruieren und letzlich auftragen.

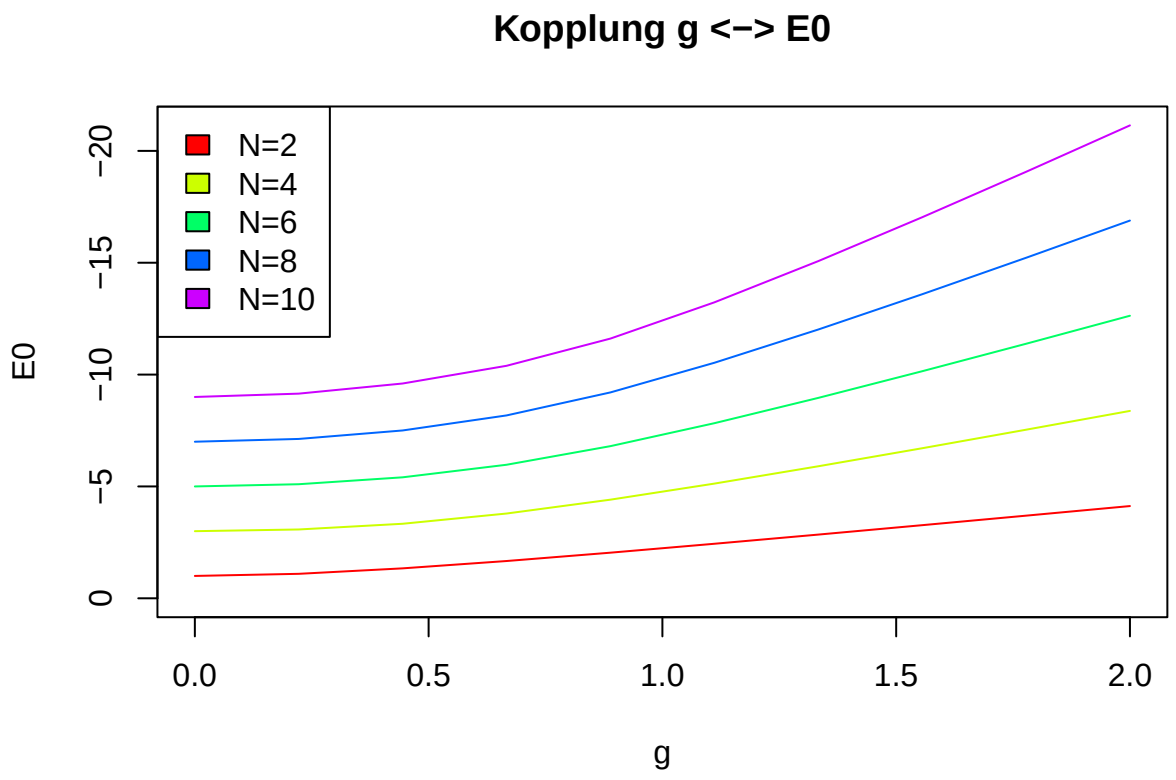
Code

```
#Hamiltonfunktion
Hamilton <- function(g, SIGMA_X_SUM, SIGMA_Z_SUM){

  ham <- SIGMA_X_SUM + g*SIGMA_Z_SUM

}

#Definition des mehrfachen Tensorprodukts
TENSORPRODUKT <- function(x1, ...){
  r <- x1
  for (vi in list(...)){
    r <- kronecker(r,vi)
  }
  r
}


#Definitionen
EINHEITSMATRIX = diag(2)
SIGMAZ = matrix(c(1,0,0,-1), nrow=2, ncol=2)
SIGMAX = matrix(c(0,1,1,0), nrow=2, ncol=2)


#Berechnung der g komponente der Hamiltonfunktion
SIGMA_Z_SUM <- function(n, EINHEITSMATRIX, SIGMAZ){

  value <- 0
  l <- 0

  for (i in c(1:n)) {
```

```
    for (k in 1:n) {
      l[k] <- list(EINHEITSMATRIX)
    }

    l[i] <- list(SIGMAZ)

    for (a in c(1:(n-1))) {
      l[1] <- list(TENSORPRODUKT(l[[1]], l[[a+1]]))
    }

    value <- value + l[[1]]
  }
  value

}


#Berechnung der SigmaX für Hamilton
SIGMA_X_SUM <- function(n, EINHEITSMATRIX, SIGMAX){

  value <- 0

  for (i in c(1:(n-1))) {

    l <- 0

    for (k in 1:n) {
      l[k] <- list(EINHEITSMATRIX)
    }

    l[i] <- list(SIGMAX)
    l[i+1] <- list(SIGMAX)

    for (a in c(1:(n-1))) {
      l[1] <- list(TENSORPRODUKT(l[[1]], l[[a+1]]))
    }

    value <- value + l[[1]]
  }
  value

}

#Vektoriteration
Eigenvalue1 <- function( x, A, steps){

  lambda <- 0

  for (k in (1:steps)) {
    w <- A %*% x
    x <- w/sqrt(sum(w*Conj(w)))
    u <- A %*% x
    lambda <- sqrt(sum(u*Conj(u)/sum(x*Conj(x))))
```

```
  }
  lambda
}

#Inverse Vektoriteration
Eigenvalue2 <- function(x, sigm, A, steps){

  lamb_k <- rep(0, 2^n)
  Identity <- diag(2^n)

  w <- x/sqrt(sum(x*Conj(x)))
  lamb <- sigm

  for (k in (1:steps)) {
    x <- inv(A - sigm*Identity) %*% w
    w <- x/sqrt(sum(x*Conj(x)))
    lamb_k[k] <- as.vector(w) %*% as.vector(A %*% w)
  }
  lamb_k
}

#Zufallsvektor
randomvec <- function(n, range){

  sample(-range:range, 2^n)

}


#Eingabewerte

g <- seq(0,2, length.out=10)
n <- c(2,4,6,8,10)
tmp <- rep(0, length(g))
l_ham <- list(tmp,tmp,tmp,tmp,tmp)

b <- 1

for (i in n) {
  a <- 1
  for (j in g) {
    tmp[a] <- tail(unname(unlist(eigen(Hamilton(j,
              SIGMA_X_SUM(i, EINHEITSMATRIX, SIGMAX), SIGMA_Z_SUM(i,
              EINHEITSMATRIX, SIGMAZ)))[1])),1)
    a <- a+1
  }
  l_ham[[b]] <- tmp
  b <- b+1
}

#Plots

plot(0,0, ylim=c(0, min(unlist(l_ham))), xlim=c(0,max(g)),
```

```
     main="Kopplung g <-> E0", xlab="g", ylab="E0", "n")


for (l in 1:length(n)){
  lines(g, l_ham[[l]], col=rainbow(length(n))[l])
}
legend("topleft", c("N=2","N=4","N=6","N=8","N=10"), fill=rainbow(length(n)))
```



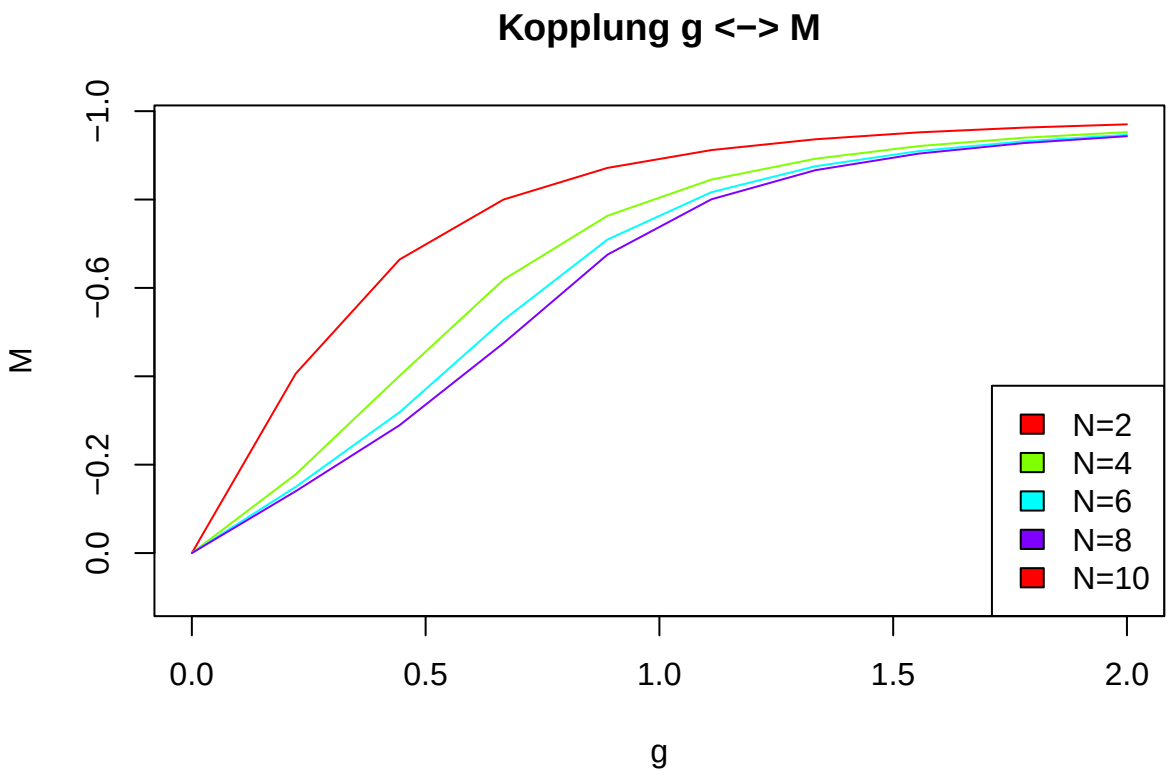
Magnetisierung als Funktion von g

Wir wollen nun die Magnetisierung mit Hilfe der gefundenen Eigenvektoren mit obiger Formel abhängig von g und N auftragen. Das Verfahren benötigt jedoch für N=8 schon recht lange und deutlich länger für N=10, sodass dieser Fall in der PDF explizit gezeigt, aber im Code auskommentiert wird.

Code

```
#Funktion zur Magnetisierung
Magnet <- function(n, SIGMA_Z_SUM, eigenvect){

  const <- 1/n
  sum <- (eigenvect) %*% (SIGMA_Z_SUM %*% eigenvect)
```

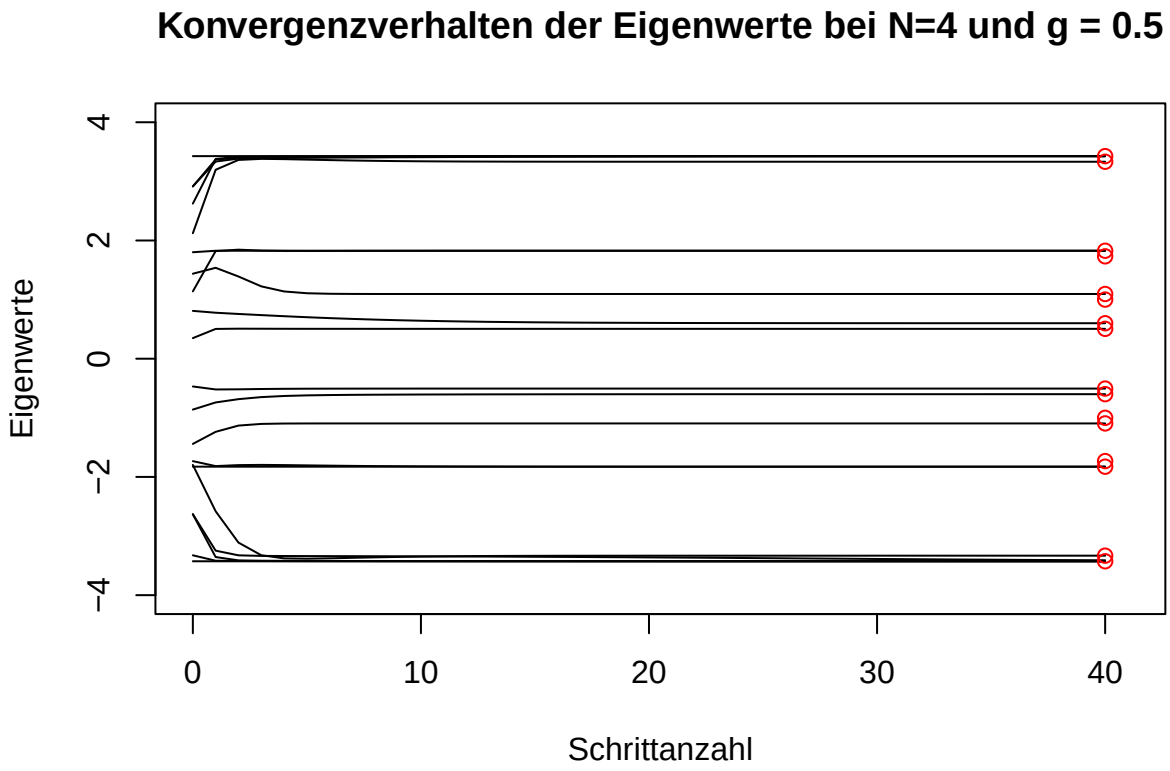
```
  mag <- const * sum
  mag
}

#Nimmt die Eigenvektoren der niedrigsten Eigenwerte und addiert diese.
Eigenvect <- function(n, ham){
  summ <- rep(0, 2^n)
  for (i in 1:length(eigen(ham)[[1]])) {
    if (tail(eigen(ham)[[1]], 1) == eigen(ham)[[1]][i]) {
      summ <- summ + as.vector(eigen(ham)[[2]][,i])
    }
  }
  summ
}

#Eingabewerte

g <- seq(0,2, length.out=10)
n <- c(2,4,6,8)
tmp <- rep(0, length(g))
l_ham <- list(tmp,tmp,tmp,tmp,tmp)

b <- 1

for (i in n) {
  a <- 1
  for (j in g) {
    tmp[a] <- unlist(Magnet(i, SIGMA_Z_SUM(i, EINHEITSMATRIX, SIGMAZ),
                            Eigenvect(i, Hamilton(j, SIGMA_X_SUM(i, EINHEITSMATRIX, SIGMAX),
                            SIGMA_Z_SUM(i, EINHEITSMATRIX, SIGMAZ)))))
    a <- a+1
  }
  l_ham[[b]] <- tmp
  b <- b+1
}

#Plots


plot(0,0, ylim=c(0.1, min(unlist(l_ham))), xlim=c(0,max(g)),
     main="Kopplung g <-> M", xlab="g", ylab="M", "n")


for (l in 1:length(n)){
  lines(g, l_ham[[l]], col=rainbow(length(n))[l])
}

legend("bottomright", c("N=2","N=4","N=6","N=8","N=10"), fill=rainbow(length(n)))
```



Vergleich für verschiedene N und Interpretation

Zuerst zur Energieverteilung: Eine höhere Anzahl an Gitterpunkten (N) sorgt hier erkennbar für eine stärkere Bindungsenergie des Grundzustands. Die Abstände sind zwischen den einzelnen Energieniveaus gleich, wobei diese mit zunehmender Kopplung zunehmen. Außerdem lässt sich erkennen, dass die Energie jeweils mit zunehmender Kopplung weiter sinkt, wobei hohe N davon stärker als kleine N betroffen sind.
Die Magnetisierung liefert folgendes Bild: Alle N besitzen für eine Kopplung von 0 erwartungsgemäß keine Magnetisierung. Sie alle streben gegen $M = -1$, was einem Diamagneten entspricht. Des Weiteren lässt sich erkennen, dass geringere N schneller $M = -1$ laufen.

Diskussion des Konvergenzverhaltens



Wie im Code schon zu erkennen, haben wir für die Endergebnisse nicht unserern Code der Inversenvektoriteration verwendet, sonder die eigen-Funktion von R selbst. Dies lässt sich durch das geplottete Kovergenzverhalten erklären:
R (rote Punkte) liefert erwartungsgemäß 16 Eigenwerte und auch Eigenvektoren für N=4. Unser Code liefert (durch Variation von sigm) nur 12 Eigenwerte. Es lässt sich dabei erkennen das das Ergebnis dabei maßgeblich von der Wahl sigm abhängt und so gerade Eigenwerte, welche nah beieinander liegen einfach fehlen. Auf Grund dieses instabilen Verhaltens und ‚da es uns nicht gelang dieses Problem zu beheben entschlossen wir uns die in R vorhandene Funktion zu nutzen. Dieses bietet auch einige weitere bedeutende Vorteile:
Die Ergebnisse sind nach Größe der Eigenwerte sortiert, sodass wir gezielt auf den kleinsten Eigenwert zugreifen konnten. Außerdem sind die Eigenvektoren und somit die Amplituden bereits normiert, was ebenfalls eine wichtige Voraussetzung war.

Fazit

Letzlich ließen sich Grundzustände, die jeweiligen Energien und die Magnetisierung gut für die verschiedenen Kopplungen und Anzahl an Gitterpunkten N darstellen. Für N=10 benötigt unser Verfahren jedoch zu lange. Leider mussten wir auf die Eigenfunktion von R zurückgreifen, um den niedrigsten Eigenwert stabil zu bestimmen (analog die Eigenvektoren).

Literatur

Tensor Product, https://stackoverflow.com/questions/8764954/outer-tensor-product-in-r, Stand: 17.07.2021

Vorlesungen vom 30.6.2021 (Eigenwerte und Eigenvektoren) und 7.7.2021 (Quanten-Ising-Modell)