GSLetterNeo vol.142

2020 年 5 月

シミュレーションで始める
量子コンピューティング

熊澤 努 kumazawa @ sra.co.jp

はじめに

最近、量子コンピュータという言葉をよく聞くようになりました。量子コンピュータはいまだ発展途上とはいうものの、プログラミングの敷居は以前より低くなってきています。本稿では、量子コンピュータのシミュレータを使って、量子コンピューティングを体験してみたいと思います。

量子コンピュータにはいくつかの方式が開発されていますが、ここでは量子アニーリング方式[1]を体験することにしましょう。量子アニーリング方式の量子コンピュータについては、D-Wave Systems 社が提供する商用のクラウドサービス Leap[2]が有名です。ユーザー登録をすると、だれでも 1 カ月間無償で量子コンピュータを使うことができます（以降は有償です）。今回の記事で使うシミュレータは無償で、しかもユーザーの登録などの手続きは不

[1]量子アニーリング方式の量子コンピュータは、アナログ計算をするコンピュータです。量子アニーリングについては、例えば、西森秀稔，大関真之：量子アニーリングの基礎，共立出版，2018．を参照してください。最近の研究開発の動向を知るには、雑誌 数理科学 2019 年 7 月号（サイエンス社）の特集「量子コンピュータの進展」を見てください。この分野のくだけた解説は、雑誌 現代思想 2020 年 2 月号（青土社）の特集「量子コンピュータ」にあります。

[2] https://cloud.dwavesys.com/leap/login/?next=/leap/

要ですので、まずはシミュレータで体験し、興味を持ったら商用サービスを使ってみるという進み方があります。

PyQUBO をインストールする

量子コンピュータ用プログラムは、Python 言語で書くことができます。PyQUBO[3](パイキューボ)は，株式会社リクルートコミュニケーションズで開発された、量子アニーリング方式に基づく量子コンピュータ用の Python パッケージです。無償のオープンソースソフトウェアとして公開されています。量子コンピュータのシミュレータも搭載しているため、気軽に量子コンピューティングを（疑似）体験することができます。今回の記事は PyQUBO のシミュレータを使ってみることにしましょう。
まず、Python を実行する環境を用意してください（Python の実行環境の構築方法の説明は省略します）[4]。次に、次のコマンドを実行して、PyQUBO をインストールしてください。インストールに失敗する場合には、公式ページからソースコードをダウンロードしてインストールする方法があるので、参照してください。

```
pip install pyqubo
```

Python プログラムを書いてみよう

準備ができたので、実際にプログラムを書いてみましょう。
簡単な例題として、0 以上の整数を 2 進数表現に変換するプログラムを作ることにします。
先に、「普通」のコンピュータで動作する Python プログラムの一例を示します。関数 `digit_to_binary` は引数 `n` に正の整数をとり、2 進数表現をリストで返します（エラーチェック等は省きます）。

```
def digit_to_binary(n):
    bits = [n & 1]
    while 1 < n:
        n >>= 1
        bits.insert(0, n & 1)
    return bits
```

[3] https://github.com/recruit-communications/pyqubo
[4] この記事で紹介するプログラムは全て Python 3.7.7 で動作を確認しています。

例えば、10 進数 2 と 10 に対する実行結果は次の通りになります。

```
digit_to_binary(2) # 10
>> [1, 0]

digit_to_binary(10) # 1010
>> [1, 0, 1, 0]
```

次に、量子コンピュータで同じ計算を実行するプログラムを書いてみます。残念ながら上のプログラムを量子コンピュータでそのまま実行することはできないため、一から書き直さなければなりません。その際、プログラムの一部は普通のコンピュータで実行し、量子コンピュータが得意とする部分のみ量子コンピュータで実行するようにプログラムを書く必要があります。ちょっと複雑なので、簡単のため、10 進数 2 を 2 進数に変換するプログラムを示します。

```
import pyqubo

def q_digit2_to_binary():
    # Variabels
    qbit0 = pyqubo.Binary('qbit0')
    qbit1 = pyqubo.Binary('qbit1')
    # Hamiltonian
    hamiltonian = (qbit0 + 2*qbit1 - 2) ** 2
    # Find the best sample with Simulated Annealing
    model = hamiltonian.compile()
    qubo, _ = model.to_qubo()
    sample = pyqubo.solve_qubo(qubo)
    solution, _, objective = model.decode_solution(sample, vartype="BINARY")
    return [solution['qbit1'], solution['qbit0']], objective
```

実行してみると、次のような結果になります。結果の最初のリスト[1, 0]が、10 進数 2 の 2 進数表現 10 になっています。

```
q_digit2_to_binary() # 10
>> ([1, 0], 0.0)
```

プログラムを読み解く

上で示した量子コンピュータを使用するプログラムは、普通のプログラムとは雰囲気がかなり違います。詳しく見ていきましょう。

1 行目は PyQubo を使うためのモジュールのインポートです。

3

```
import pyqubo
```

関数 q_digit2_to_binary 内では、最初に PyQUBO で使う変数を 2 つ用意しています。

```
    # Variabels
    qbit0 = pyqubo.Binary('qbit0')
    qbit1 = pyqubo.Binary('qbit1')
```

10 進数 2 は 2 ビットで表現できるので、下位ビットと上位ビットに相当するバイナリ変数を、それぞれ qbit0, qbit1 とします。変数に、変数名と同じ名称のラベルを文字列で付けていますが、これは後で計算結果を参照する際に必要です。PyQUBO には、0 と 1 のいずれかをとる pyqubo.Binary と、-1 と 1 のいずれかをとる pyqubo.Spin の 2 種類の変数しかないことに注意が必要です。文字列型や整数型といった変数はありません[5]。

次に、ハミルトニアンと呼ばれる「数式」を用意します。

```
    # Hamiltonian
    hamiltonian = (qbit0 + 2*qbit1 - 2) ** 2
```

ハミルトニアンとは物理学で出てくる概念です[6]。この式は何かを計算した結果得られた値を変数 hamiltonian に代入しているように見えますが、そうではなく、変数 qbit0, qbit1 の値が与えられた場合の計算式そのものを定義しています。一種の関数や演算子と考えた方がわかりやすいかもしれません。実際の応用の場面では、この式のことを、目的関数や評価関数と呼ぶこともあります。

量子コンピュータは、ハミルトニアンの値が最小となるような変数 qbit0, qbit1 の値を求めます。どんな問題でも解ける一般的なハミルトニアンの形式はないので、計算したい内容や解きたい問題に応じて、ユーザーである人間が予め決める必要があります。ハミルトニアンをうまく決めることは、量子コンピュータを使ううえで、最も重要なポイントの一つです。

---

[5] 用意されていないというだけなので、記事の例のように、2 進数で整数を表現しても構いません。

[6] 系の全エネルギーに相当します。例えば、須藤靖：解析力学・量子論，東京大学出版会，2008（最新版は第 2 版）．や、清水明：新版 量子論の基礎，サイエンス社，2004．を参照してください。

上のプログラムではハミルトニアンは次のように決めました。下位と上位のビットから `qbit0 + 2*qbit1` で、元の 10 進数を計算します。次に、そこから 2 を引いて結果を 2 乗します。すると、`qbit0 = 0, qbit1 = 1` のときには、ハミルトニアンの値は 0 になります。そうでない場合、例えば、`qbit0 = 1, qbit1 = 0` のときには、$(1 + 2 \times 0 - 2)^2 = 1$ となり、必ず 0 より大きくなります（最後の 2 乗は、ハミルトニアンの値が負の数にならないようにするための処置です）。つまり、ハミルトニアンの最小値を量子アニーリングで求めると、その値が 0 となり、そのときの変数の値が `qbit0 = 0, qbit1 = 1` となるのです。これは、正に 10 進数 2 の 2 進数表現 10 を表しています。

次は、量子アニーリングのシミュレータへの入力を作成する処理です。上で書いた数式を QUBO（Quadratic Unconstrained Binary Optimization:制約なし 2 値変数 2 次形式最適化）という形式に変換しています。

```
    model = hamiltonian.compile()
    qubo, _ = model.to_qubo()
```

ここまでは普通のコンピュータで実行するプログラムです。

最後に、QUBO を入力として量子アニーリングのシミュレータ[7]を実行します。

```
    sample = pyqubo.solve_qubo(qubo)
    solution, _, objective = model.decode_solution(sample, vartype="BINARY")
    return [solution['qbit1'], solution['qbit0']], objective
```

`pyqubo.solve_qubo` を呼ぶと、シミュレータは、ハミルトニアンを最小化する `qbit0, qbit1` の値と、その時のハミルトニアンの値を返します。この場合、変数 `solution` に `qbit0, qbit1` の値が、変数 `objective` にハミルトニアンの値が格納されます。変数名をキーとして下位ビットと上位ビットの値を取り出してリストにし、ハミルトニアンの値と共に返しています。

---

[7] シミュレータは、焼きなまし法という技術を実装しています。これは、量子アニーリングとは異なる物理現象を基に考案された最適化技法です。焼きなまし法については、例えば、Juraj Hromkovič: Algorithmics for Hard Problems, 2nd ed., Springer, 2004（日本語訳は、J. ホロムコヴィッチ著，和田幸一，増澤利光，元木光雄訳：計算困難問題に対するアルゴリズム理論，丸善出版，2012）を参照してください。

なお、量子コンピュータの実機として D-Wave Systems 社が提供する量子アニーリング機を使用する場合には、pyqubo.solve_qubo の代わりに D-Wave's Ocean Software が提供する Python API を実行します。その場合でも、QUBO を作成する処理までは変更の必要はありません。
今までは 10 進数が 2 の場合の変換でしたが、より一般的に、ユーザーが指定した値を 2 進数表現にするプログラムを次に示します。

```
import math
import pyqubo

def q_digit_to_binary(n):
    # (1) Variabels
    shape = math.floor(math.log(n, 2)) + 1 if 0 < n else 1
    qbits = pyqubo.Array.create('x', shape=shape, vartype='BINARY')
    # (2) Hamiltonian
    hamiltonian = (sum(qb*(2**d) for d, qb in enumerate(reversed(qbits)))-
 n)**2
    # (3) Find the best sample with Simulated Annealing
    model = hamiltonian.compile()
    qubo, _ = model.to_qubo()
    sample = pyqubo.solve_qubo(qubo)
    solution, _, objective = model.decode_solution(sample, vartype="BINARY")
    return list(solution['x'].values()), objective
```

プログラムの詳細な解説は省きますが、考え方は 2 を変換したプログラムと同じです。まず、(1)下位から上位の各ビットに対応する pyqubo.Binary 変数の配列を用意します。次に、(2)ハミルトニアンとして、各ビットから元の数を復元し、その差の 2 乗を計算する式を準備します。最後に、(3)ハミルトニアンを QUBO に変換してシミュレータで最小化する、という流れです。
試しに実行してみましょう。

```
q_digit_to_binary(10) # 1010
>> ([1, 0, 1, 0], 0.0)

q_digit_to_binary(100) # 1100100
>> ([1, 1, 0, 0, 0, 1, 1], 1.0) # 99!
```

10 進数 10 は正しく 2 進数表現に変換できましたが、100 はできていませんね（答えとして得られた 2 進数 1100011 は、10 進数の 99 です）。シミュレータは、ハミルトニアンの最小化を必ず達成することは保証していません（達成できない、という意味ではありません）。「できるだけ」最小化するという方が近いでしょう。うまくいくことも、いかないこともあるということです。ここにも、従来のコンピュータでの計算との違いが見られます。

おわりに

本稿では、シミュレータを使って量子コンピューティングを体験しました。量子アニーリング方式を使用する際には、ハミルトニアンという関数のような数式をユーザーが用意します。それをシミュレータで最小化することで、計算を行います。

応用に使うには、ハミルトニアンをどうやって用意するかが重要になります。計算したい問題に応じて、ユーザーが適切に設定する必要があります。例でみたように、10 進数を 2 進数に変換するという簡単な計算でも、なかなか大変な作業です。このように、現時点では、量子コンピュータの普通のコンピュータに対する優位性については、はっきりしない点もあり、専門家の間で議論が続いています。また、量子コンピュータに関しては、現状大規模な計算の実現が難しいという課題があります。適切な規模の問題を適切に定式化する、ということが大変重要です。

量子コンピュータが実用化に至るまでは、まだまだ多くの課題があります。その一方で、これまでのコンピュータと違った挙動をして、使ってみると面白い結果が得られます。まずはシミュレータで気軽に楽しんでみてはいかがでしょうか。

GSLetterNeo Vol.142
2020 年 5 月 20 日発行
発行者　株式会社 SRA 先端技術研究所

編集者　土屋正人
バックナンバー http://www.sra.co.jp/gsletter
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8