GSLetterNeo vol.144

2020 年 7 月

シミュレーションで始める
量子コンピューティング (2)

熊澤 努 kumazawa @ sra.co.jp

はじめに

Vol. 142 に続いて，シミュレータを使いながら量子アニーリング方式の量子コンピュータを体験します。プログラミングとシミュレーションには，前回と同様に PyQUBO を使います。今回は、より具体的な応用として、機械学習でよく扱われるクラスタリング（クラスター分析）を実行するプログラムを作ってみましょう。本稿では、以下の研究論文で提案されている技術を一部紹介します。詳しくは、論文を参照してください。

Kumar, V., Bass, G., Tomlin, C., & Dulny III, J. (2018). Quantum annealing for combinatorial clustering. In *Quantum Information Processing* (Vol. 17). Springer. doi:https://doi.org/10.1007/s11128-017-1809-2.

量子アニーリングでのクラスタリングの実現については、前回紹介した以下の文献でも解説されており、本稿の執筆の際に参考にさせていただきました。

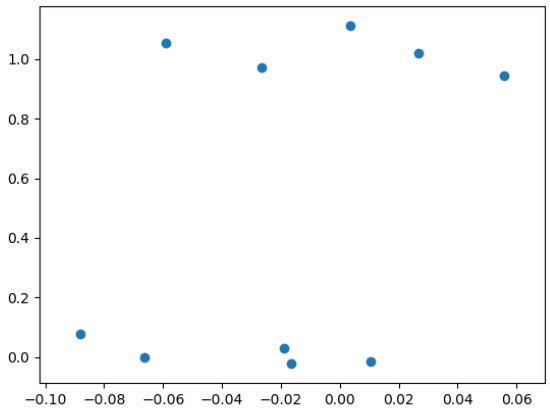
西森秀稔，大関真之：量子アニーリングの基礎，共立出版，2018.

クラスタリング

クラスタリングは、たくさんあるデータをうまくグループに分けることを目的としたデータ分析の技術です[1]。データを分けることで、全体の傾向を把握しやすくしたり、人が気づいていなかったグループを新たに発見したりするなど、データの分析手段として広く使われています。

クラスタリングの基本的なアイデアは、「似たデータ同士は同じグループに所属させ、似ていないデータは互いに違うグループに所属させる」というものです。似ているかどうかは人が予め定めておけばよく、普通はデータ同士の距離として与えておきます。しかし、問題は、どのデータを同じグループにすべきか簡単には判定できないことにあります。

下の図は、10 個の 2 次元データを xy 平面上に描いたものです。どのようにグループ分けすればよいでしょうか。



ぱっと思いつくのは、縦軸の 0.8 より上と、0.2 より下の 2 つにデータを分けることでしょうか。左下の 2 点を別グループにして、合計 3 つのグループとすることもできそうです。一方、一番左上の点と左下の 2 点とで一つのグループとし、残りの 7 点を別のグループとして、左右に分けることも考えられます。このように、グループ分けをするにしても、分け方にはいろいろあり、一つには決まらないことが多いです。しかしながら、もっと深刻な問題は、コンピュータでの実行時間にあります。コンピュータでクラスタリングをする最も簡単な方法は、とにかくすべての点を適当にグループ分けし、うまくできていることを距離を使って計算して確認する方法です。うまくいかなければ、別のグループ分けを試します。簡単のため、グループを二つに限定して話を進めると、各点がグループ 1 からグループ 2 のどち

[1] 本稿ではクラスタリングの詳細な解説は割愛します。例えば、高村大也，奥村学：言語処理のための機械学習入門，コロナ社，2010．を参照してください．

らかに分けられることになります。何通りのグループ分けを試せばよいでしょうか。上の図の 10 個の点の場合、$2^{10} = 1024$通りです。点を 20 個にすると約100万通り、100 個では約10^{31}通りとなり、点を 100 個程度に増やしただけで、計算に膨大な時間がかかるようになります。そこで量子コンピュータを使ってみようというわけです。

Python プログラムを書く

グループ数を 2 とした場合について、量子アニーラ用の Python プログラムを考えます。点同士の距離は、普通のコンピュータで先に計算されているものとします。全ての点の座標が分かっている状況を考えても構いません。Vol.142 で見たように、量子アニーラで計算をするには、変数とハミルトニアンを定義して、シミュレータに渡す必要があります。

```
def binary_clustering(distances):
    # 変数の定義
    # ハミルトニアンの定義
    # シミュレータで、ハミルトニアンが最小になるような変数の値を求める
    return 結果
```

上のプログラムの引数 `distances` は、点同士の距離を格納する 2 次元のリストとします。`distances[i][j]`が点 i と点 j の距離です。以降、Kumar 等のアプローチを実装します。
変数については、各グループへの所属を 1 と-1 で表すような pyqubo.Spin 変数を、点の数だけ用意すればよいでしょう。`pyqubo.Array.create` の引数 `vartype` に`'SPIN'`を指定すると、pyqubo.Spin 変数の配列を生成します。

```
def binary_clustering(distances):
    # 変数の定義
    npts = len(distances)
    qbits = pyqubo.Array.create('s', shape=npts, vartype='SPIN')
    # ハミルトニアンの定義
    # シミュレータで、ハミルトニアンが最小になるような変数の値を求める
    return 結果
```

次はハミルトニアンです。クラスタリングのアイデアである「近い者同士を一つのグループにする」となるようにハミルトニアンを定義しなくてはいけません。以下に Python での定義例を示します（折り返しにより 2 行になっています）。

```
def binary_clustering(distances):
    # 変数の定義
    npts = len(distances)
    qbits = pyqubo.Array.create('s', shape=npts, vartype='SPIN')
    # ハミルトニアンの定義
    h = sum(sum(distances[i][j]*qbits[i]*qbits[j] for j in range(npts) if i<j)
     for i in range(npts))
    # シミュレータで、ハミルトニアンが最小になるような変数の値を求める
    return 結果
```

「全ての点のペアについて、近い 2 点が同じグループになり、遠い 2 点が違うグループになるとハミルトニアンが低い値をとる」ようにハミルトニアンを定義しているのですが、プログラムが少し複雑です。詳しく見ていきましょう。まず、和の計算が 2 重になっていますが、それぞれ 2 つの点 i と j について足し算をするだけです。大事なのは計算式 distances[i][j]*qbits[i]*qbits[j]です。distances[i][j]は点 i と j の距離、qbits[i]と qbits[j]はそれぞれ点 i と j が所属するグループですが、それらの乗算は何を意味しているのでしょうか。次の 2 つのケースを考えると、その理由が見えてきます。

- distances[i][j]が大きな値、つまり、点 i と j が遠い位置にある場合は、両者は違うグループに所属することが望ましいはずです。つまり、qbits[i]と qbits[j]が異なる値になり（一方が 1 ならばもう一方は-1 です）、両者の積 qbits[i]*qbits[j]の値は-1 です。距離は常に 0 以上ですので[2]、distances[i][j]*qbits[i]*qbits[j] の符号はマイナスになります。もし同じグループに所属すると、qbits[i]と qbits[j]は両方とも 1、両方とも-1 のどちらかなので、qbits[i]*qbits[j]の値は 1 になります。すると、distances[i][j]*qbits[i]*qbits[j] の符号がプラスになり、違うグループに所属する場合よりハミルトニアンの値が大きくなります。
- distances[i][j]が小さい値、つまり、点 i と j が近い位置にある場合は、qbits[i]と qbits[j]が同じ値なのが望ましいですが、上と同じように考えると、違う値になるように思えます。しかし、distances[i][j]が相対的に小さい値のため、違うグループにしてもハミルトニアンの低減にはさほど貢献しません。他の大きい距離で効果が打ち消されてしまい、結果的に同じグループに所属した方がハミルトニアンの値が低くなります。

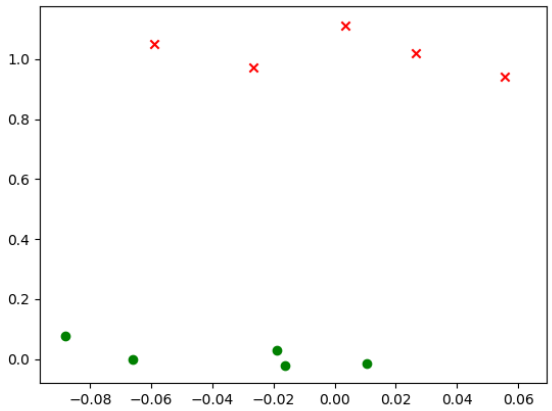
[2] マイナスの距離は存在しません。東京駅 - 上野駅間の距離を -3.6km とは言わないですよね。

条件 `if i<j` には 2 つの役割があります。一つは、点 i と j が同じ位置の場合を除くこと(距離は 0 なので計算しなくてよい)、もう一つは、点 i と j を入れ替えた場合を除くこと(入れ替えても距離は同じなので冗長)です。最後にすべてを足せば終わりです。
長くなりましたが、もう少しでプログラムは完成です。シミュレータで計算する部分は Vol.142 と同じ流れです。違う点は、使う変数 pyqubo.Spin に応じたシミュレーションを実行するようするところです。QUBO ではなく pyqubo.Spin 変数を使ったイシングモデルの最小化を実行するために、`model.to_ising` でイシングモデルを生成した後、シミュレータ `pyqubo.solve_ising` で最小化します。

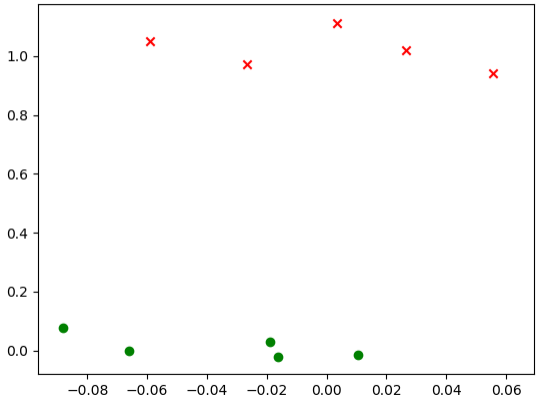
```
def binary_clustering(distances):
    # 変数の定義
    npts = len(distances)
    qbits = pyqubo.Array.create('s', shape=npts, vartype='SPIN')
    # ハミルトニアンの定義
    h = sum(sum(distances[i][j]*qbits[i]*qbits[j] for j in range(npts) if i<j)
     for i in range(npts))
    # シミュレータで、ハミルトニアンが最小になるような変数の値を求める
    model = h.compile()
    linear, quad, _ = model.to_ising()
    sample = pyqubo.solve_ising(linear, quad)
    solution, _, objective = model.decode_solution(sample, vartype='SPIN')
    return list(solution['s'].values()), objective
```

シミュレーションを実行する

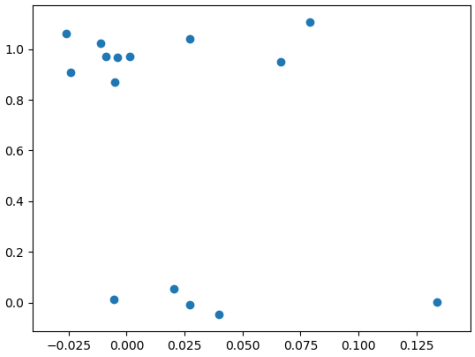
シミュレータを使い、実際に「クラスタリング」のセクションで例示したデータをクラスタリングしてみた結果を下に示します。各グループに所属する点を、それぞれ赤のバツと緑の丸で描画しました。上下のグループに綺麗に分けられていることが分かります。

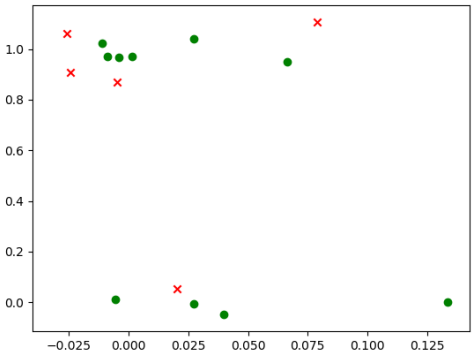


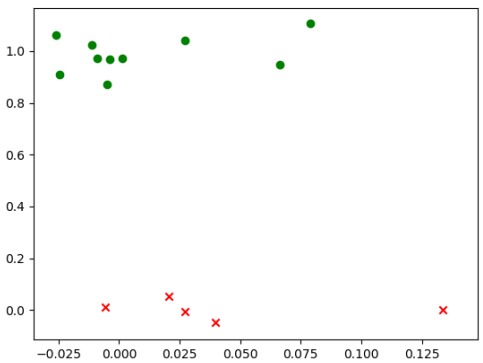
確認のため、グループ分けの全組合せを計算してクラスタリングを実行した結果を下に載せます。結果は全く同じです。



今度は点の数を増やしてみましょう。以下に 15 点にした場合のデータ、シミュレータの実行結果、全組合せを計算した結果を示します。シミュレータを使った場合、グループ分けがすっきりとはできていないことが分かります。ハミルトニアンが最小でないところでシミュレータの計算が終了したためです。Kumar 等の論文では、データ数が 40、1000、2000 の場合について、D-Wave Systems 社の実機を使った実験結果が論じられています。





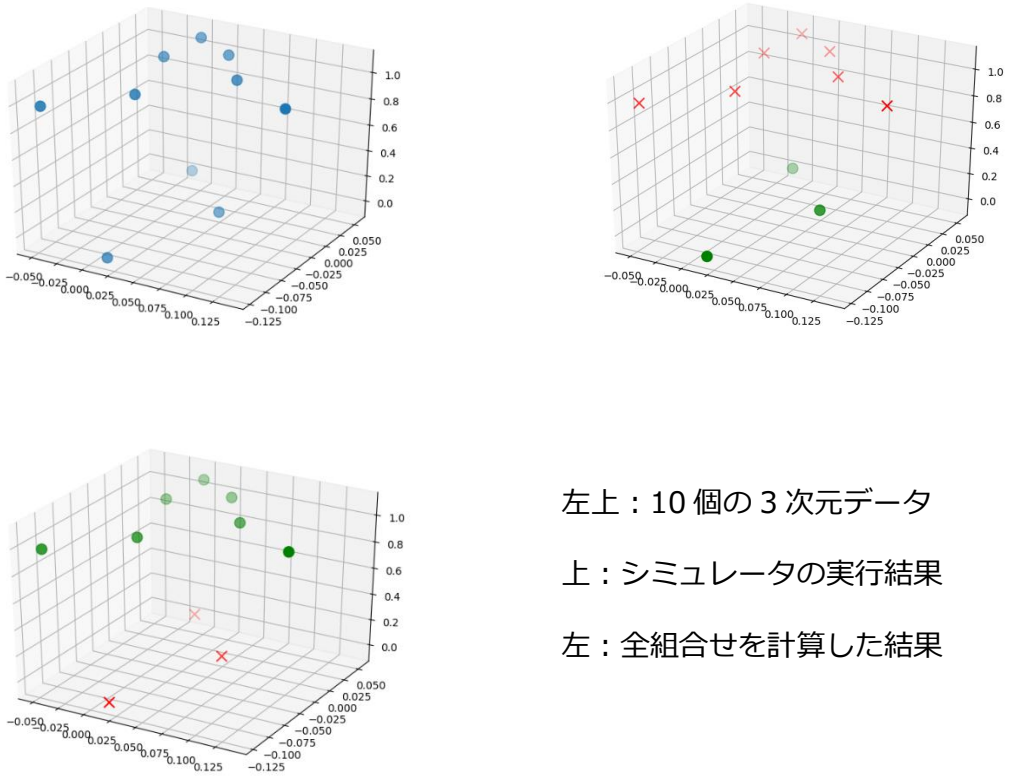


左上：15 個の 2 次元データ

上：シミュレータの実行結果

左：全組合せを計算した結果

最後に 3 次元データ 10 個の場合の結果を示します。シミュレータと全組合せとでマーカーが逆転しているのは、たまたまグループへの分割の仕方が逆になったためです。



左上：10 個の 3 次元データ

上：シミュレータの実行結果

左：全組合せを計算した結果

おわりに

本稿では、Kumar 等の論文で提案されている、クラスター数 2 のクラスタリングを紹介しました。加えて、論文には、クラスターが 2 より多い場合についての分析する技術の詳しい議論があります。

データの数を増やして量子アニーリング方式の量子コンピュータ動かそうとすると、ハードウェアの制約のため、うまくいかないことがあります。本稿ではそこまで説明することはできませんが、問題を小さい規模に分割したり、データに前処理を施すなどの技術が追加で必要になります。本稿のような小規模な問題であれば、そのような処理を行わなくともシミュレータや実機で解きやすいので、体験をするのに相応しいと言えるでしょう。

量子アニーリングはハミルトニアンを最小化する計算方式です。一般にこのような技術は最適化といわれています。最適化技術は現在も活発に技術開発が進められており、多くの分野で応用されています、今回使用したシミュレータも、焼きなまし法（シミュレーレッド・ア

ニーリング法）という最適化技術を実装したものです。従来の最適化技法と量子アニーリングよる最適化とを比較してみるのも面白いかもしれません。

GSLetterNeo Vol.144
2020 年 7 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

夢を。Yawaraka Innovation
やわらかいのべーしょん