2022/05/05

# 数学トピックス04

---

## 間接的にこのシステムを探るための私の理論について😊(2022/05/04)

2022/05/02からのこのシステムの誕生前後の私の視点をお伝えたことが功を奏し、日本の私のことを知っているみなさんはほぼこのシステムを信用しなくなりました。これにより、私の散歩中では間接的にこのシステムを探るために考えた私の理論もほとんど使わなく済むようになりました。😀 ちょうど良いころですので、その理論について簡単に説明したいと思います。😀

実は5年前にこのシステムの関係者の声に傾けている時に、この理論を説明していたのです。当時はこのシステムのことを恐れていましたから、とりあえずプレゼントしておきました。😀 ちょうど京都大学院で大変お世話になった先生が、このシステムの中に入られていたようで、先生に説明していました。今振り返りますと本当に先生自身かは曖昧な感じもしています。

この理論は京都大学院（MBA）の先生の研究室配属時に学んだマルコフ連鎖のようなものを用います。マルコフ連鎖は、簡単に説明しますと、状態の移り変わりを確率を用いて表せるものです。

たとえば、私は俳優をしていますので、ある場所では、人に言えないような非常に目立つ写真を見たりしていました。😉 その後、このシステムがその私の行動を流しているかいないかをチェックするとします。2択です。

始めの段階では、私はこのシステムの動きを全く知りません。ということですので、このシステムがその情報を流しているかいないかは両方とも50%となります。コインの裏表と同じです。

その後、私は散歩等でこのシステムを見た人の話等のデータを得るようになります。すべての人がその情報の話をするわけではありません。出来るだけその私の行動に関連する話だけを抽出していきます。その集めたデータを一つずつ、その私の行動に関連する度合いを決めていき、このシステムがその情報を流しているかいないかの確率を変化させていくというものです。

始めの一つのデータが得られ、このシステムがその情報を流している可能性が少し高めとして7:3の割合とします。つまりこの情報を用いて、上の5:5の情報を更新するのです。その更新するために行列を用います。以前も数学のトピックで行列を用いた変換を書きましたが、ここでは情報を更新するという意味で使います。

$$(Yes_2, No_2) = (Yes_1, No_1)\begin{pmatrix} Yes_1, No_1 \\ Data_1 \end{pmatrix}$$

$$= (0.5 \quad 0.5)\begin{pmatrix} 0.5 & 0.5 \\ 0.7 & 0.3 \end{pmatrix}$$

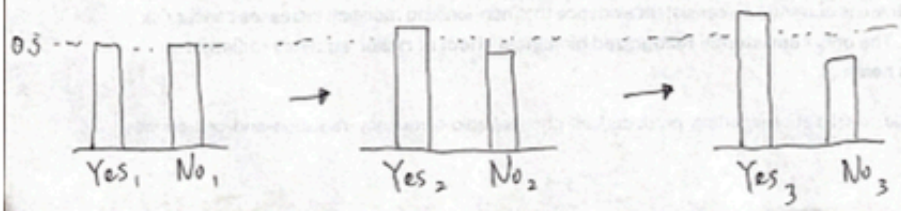
$$= (0.25 + 0.35,\ 0.25 + 0.15)$$

$$= (0.6 \quad 0.4)$$

$$(Yes_3, No_3) = (Yes_2, No_2)\begin{pmatrix} Yes_2, No_2 \\ Data_2 \end{pmatrix}$$

$$= (0.6 \quad 0.4)\begin{pmatrix} 0.6 & 0.4 \\ 0.9 & 0.1 \end{pmatrix}$$

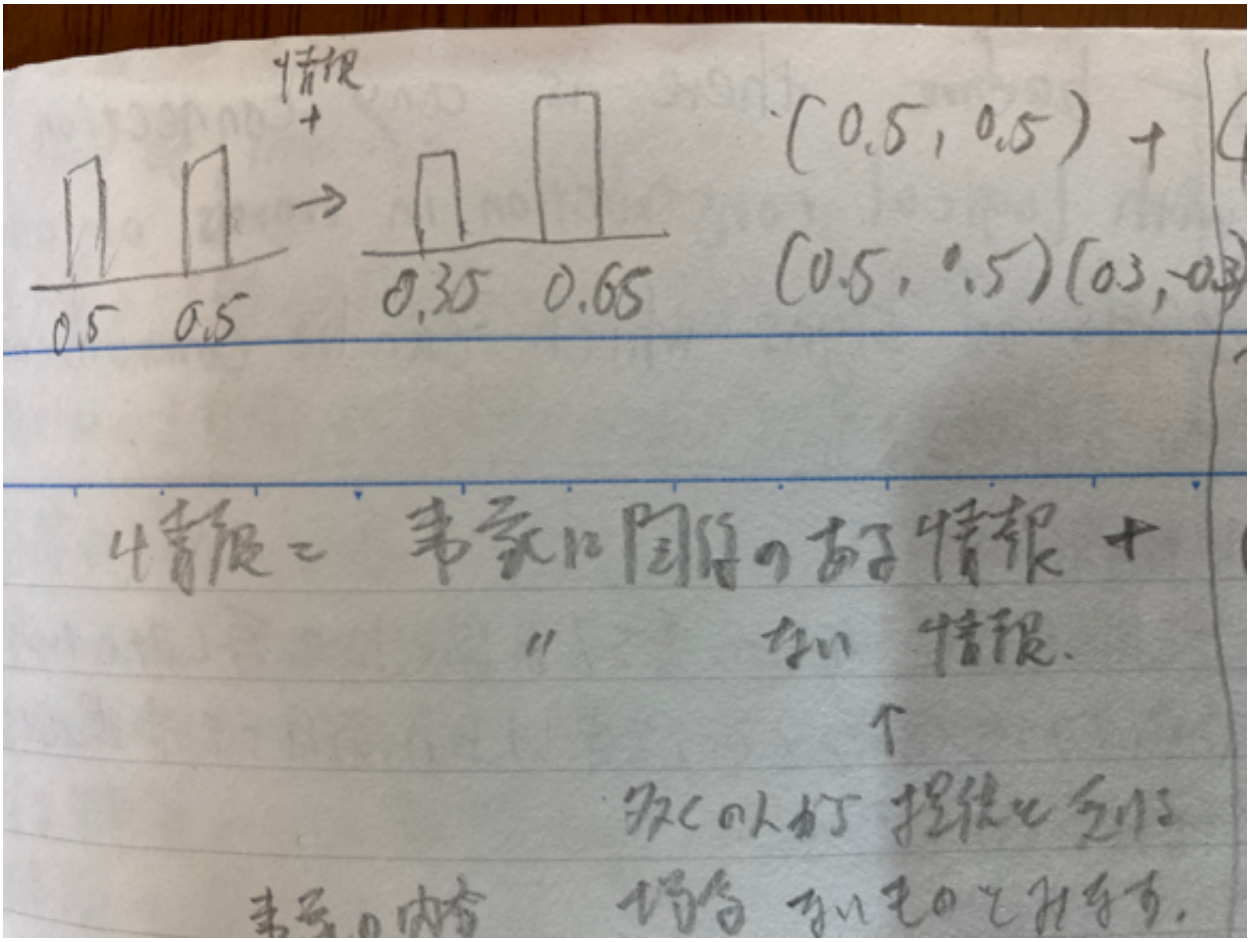
$$= (0.72 \quad 0.28)$$




(Yes1, No1)=(0.5, 0.5)から(Data1)=(0.7, 0.3)と(Data2)=(0.9, 0.1)を得て、(Yes2, No2)=(0.6, 0.4)、(Yes3, No3)=(0.72, 0.28)と続く場合を書いております。

また、マルコフ連鎖の場合は、同じ行列を掛けていくために色々便利で良いのですが、私の理論の場合は、行列がデータごとに違うため、行列の交換法則が成り立たない時もあり、データの順序を変えますと、結果が異なってしまう可能性があるという欠点があります。🤦

実は5年前にこのシステムに説明した時は、式もほとんど出していない状態でした。😉 ちなみに当時のノートに少し書いていました。




私は頭の中で、実際この理論に基づいて計算をいつも行なっているわけではないです。理論的に説明すると、そんな感じという程度です。上の欠点を補うために最終的に結果の平均を取っていくというのも良いかもしれません。😉 もしかしたら、行列の交換法則が成り立つような違う数学

の理論があるかもしれません。🧐

Keywords: マルコフ連鎖, 確率, 行列, 線形代数, 交換法則

## 自己参照は不思議な力がある！？🧐 (2022/03/25)

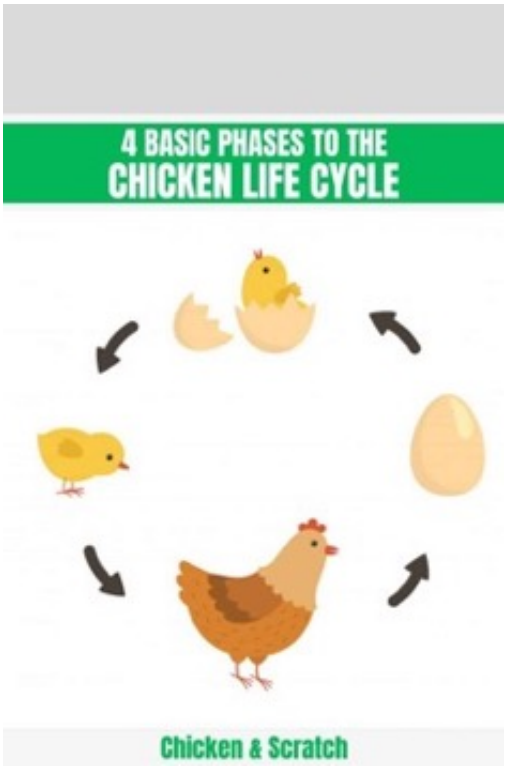
ポッドキャスト「Breaking Math」の最新エピソードを聴いていました。「自己参照」についての面白い話が沢山ありました。ありがとうございます。😊以下はこのエピソードの内容の一部を変更しています：

> 円グラフには、円グラフのどれだけの部分が赤くなっているかが書かれています。つまり、この円グラフは自分自身のグラフの説明をしている円グラフということになります。🙃

またプログラミング言語でBootstrappingのことも学べました。😀また、プログラミング言語で良く出てくる、ファンクションの中にそのファンクションを呼ぶという再帰ファンクションというのがあります。😀

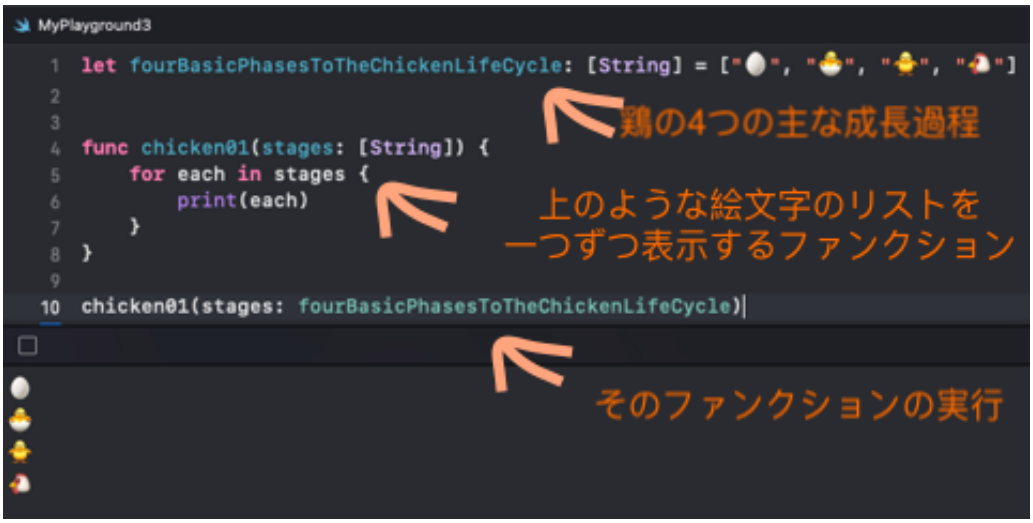
私は生物の基本的な仕組みは再帰ファンクションのように単純なのではないかと思うようになりました。「4 Basic Phases to the Chicken Life Cycle」という記事を参考に Swift Playgroundとアップルスクリプトで簡単に説明したいと思

います。😀




画像：上の記事から

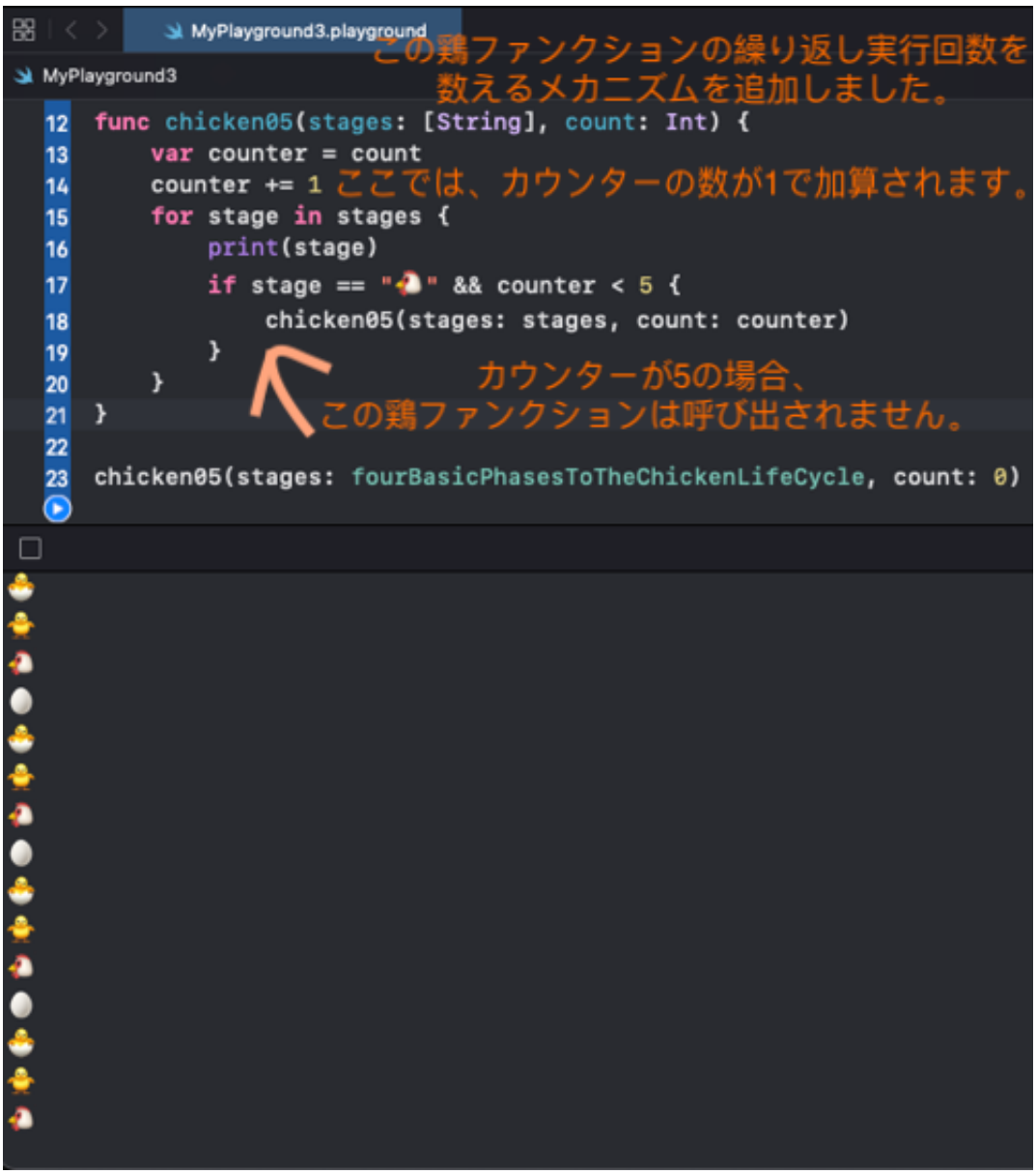
例えば、鶏の一生を4つの絵文字（🥚、🐣、🐥、🐓）で表すとしますと、それらを初めから一つずつ表示する鶏ファンクションを考えます。鶏が卵の状態からひよこになり、最後に鶏になります。これは、下の画像のような結果になります。アップルスクリプト版では絵文字を1秒ずつ表示する動画を作りました。😀



画像：鶏ファンクションとその結果

これに、この鶏ファンクションの中にこの鶏ファンクション自身を入れてみます。簡単説明しますと、一回鶏ファンクションを実行しますと、実行中に鶏ファンクションが実行するように呼ばれます。ここで注意が必要ですが、普通にこの鶏ファンクションを入れただけですと、鶏ファンクションの止まる仕組みがないため、動き続けます。私はなんとなく生物がこのような感じだと思っております。

では、この鶏再帰ファンクションを5回繰り返したら止まるようにします。結果が下の画像のようになります。ただし結果が全部表示できていません。😅またアップルスクリプト版の動画も作りました。😃




画像：鶏再帰ファンクションとその結果の一部

おまけ：

アップルスクリプトでは文字を読んでくれるので、絵文字も読んでくれると思って試してみました。👍アップルスクリプトでは、log -> sayとすれば、絵文字も読んでくれます。😊アクセシビリティのようなことも簡単に出来ます。💪しかし、絵文字は正確に鶏の一生を表していません。😅

また、04/20/2022にRon Garciaのツイートをライク経由で見つけまして、自己参照の私の考えの話と合っていると思いました。ありがとうございます。😊

彼のツイートは「Effective C」という本の一部の紹介なのですが、C言語のfor　loopの注意する点を事例を示してわかりやすかったです。😃以下は彼のツイートの画像の文章を引用です：

> ラップアラウンドのため、符号なし整数式は0未満と評価することはできません。これを見失って、常に真または常に虚偽の比較を実装するのは簡単です。**たとえば、次のforループのiは決して負の値を取ることができないため、このループは決して終了しません。**

> for (unsigned int i = n; i ›= 0; --1)

> この実行の行動は、いくつかの注目すべき現実世界のバグを引き起こしました。たとえば、ボーイング787の6つの発電システムはすべて、対応する発電機制御ユニットによって管理されています。
> **連邦航空局によると、ボーイングの実験室試験では、248日間連続して稼働した後、発電機制御ユニットの内部ソフトウェアカウンターがラップアラウンドされることが判明した。この欠陥により、エンジン搭載発電機の6つの発電機制御ユニットはすべて同時にフェイルセーフモードに入ります。**
> 計画外の行動(飛行機が空から落ちるなど)を避けるために、‹limits.h›の制限を使用してラップアラウンドを確認することが重要です。これらのチェックを実行するときは、間違いを犯しやすいので注意が必要です。

ラップアラウンドについて少し調べてみました。😀「Integer Overflow Attack and Prevention」の記事がわかりやすかったです。ありがとうございます。😊以下はこの記事の引用です：

> **整数オーバーフロー(ラップアラウンドとも呼ばれる)は、算術演算が割り当てられたメモリスペースの外側にある数値を出力するか、整数の指定された値の範囲をオーバーフローするときに発生します。主にすべてのプログラミング言語で、整数値には限られたストレージビットが割り当てられています。**

> たとえば、0から65535の範囲の符号なし整数、または-32768から32767の範囲の符号付き整数を格納できる16ビット整数値があります。したがって、算術演算中に、結果が割り当てられたスペース(65535+1など)以上を必要とする場合、コンパイラは次のことができます。

> *引き起こされたエラーを完全に無視するか、または
> *プログラムを中止する。

> ほとんどのコンパイラはオーバーフローを無視し、予期しない出力またはエラーを保存します。これにより、バッファオーバーフローなど、最も一般的な攻撃が発生し、悪意のあるプログラムや特権エスカレーションの実行につながります。

とても参考になりました。ありがとうございます。😊そういえばプログラミング言語で整数の指定された値の範囲を最近気にしてませんでした。🤦

Keywords: 自己参照, 円グラフ, Bootstrapping, 再帰ファンクション, 生物, ラップアラウンド, C言語, for　loop, ボーイング787, オーバーフロー