KSS PC BOOK 2022

2021-05-xx 版　　KSS PC Club 発行

目次

はじめに 3

この部誌を支える技術 4

ICPCへのお誘い 9

ワンクリック詐欺サイト解剖してみた 16

ニュートン法で近似をしよう！！ 22

はじめに

このたびは本書をお手に取っていただきありがとうございます。本書は令和 4 年度けやき祭のためにパソコン部の有志によって作成された部誌です。

古河中等教育学校パソコン部 (通称：KSS PC Club) では部員それぞれが主に好きなことをし、競技プログラミングや Web 開発、ゲーム開発など様々なことに取り組んでいますが、本書では部員の興味を持っていることや語りたいことなどについて記事を自由に執筆してもらい、一冊の本に仕上げました。

新型コロナウイルスの影響により、文化祭でパソコン部が作品等を展示するのは 2019 年以来の 3 年ぶりであり、久しぶりの作品展示の機会であるのでぜひ特別棟2階奥のパソコン室にもお立ち寄りください。

部誌の制作については今年度からの初の試みであり、自分の言葉で物事を発信する場を設けるという意義があります。このような発信活動の場のひとつである本書を通して、読者のみなさまにも創作・技術に触れる楽しさ、好きなことに接する楽しさを感じてもらえると幸いです。

部長 細島涼雅

お問い合わせ先

本書に関するお問い合わせは部員まで。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任及び判断の上で行ってください。これらの情報による開発等の結果について、著者はいかなる責任も負いません。

この部誌を支える技術

Ryoga.exe

はじめに

こんにちは、5 期生の Ryoga.exe です。この部誌は今年からの初の試みということもあり折角なので (？) 今回はこの部誌を支える技術についてお話しようかと思います。この部誌については GitHub 上で管理されているため、そのリポジトリも合わせてご覧ください。[1]

Vivliostyle について

本書は HTML & CSS で組版ができる `Vivliostyle` [2] の `Create Book` [3] というものを使って書かれています。

▲図1: Vivliostyle のロゴ

Vivliostyle は CSS 組版というWeb標準技術をベースにした、自動組版システムのオープンソースプロジェクトです。

CSS組版はHTMLとCSSをベースにしているので、普段から HTML/CSS を扱っている人にとっては Vivliostyle のプロダクトが比較的手軽に感じるでしょう。

[1] https://github.com/kss-pc-club/book-2022

[2] https://vivliostyle.org/ja/

[3] https://docs.vivliostyle.org/#/ja/create-book

Vivliostyle を選択した理由

まず、複数人で書くことが前提となっていたので原稿等を共有する必要があります。また、見た目などもある程度統一させたいです。そのため、Word 等のソフトウェアは真っ先に候補から外れました。

そして、本を書く上で新たに多くの記法などを覚えるのは大変かつ、脳のリソースの無駄になります。そこで、Markdown で書けて、スタイルをコードで整えることができ、GitHub 上で管理できるといったことが必要でした。

そのため、Vivliostyle を選択しました。(日本語のドキュメントが充実しているという理由もありました。)

VFM について

Create Book がサポートする Markdown 方言は、Vivliostyle Flavored Markdown (VFM) です。詳細は公式ドキュメント [4] を参照してください。

今回はこの VFM でサポートされている記法について、ざっくりとご紹介します。

```
---
title: セロ弾きのゴーシュ
---

# セロ弾きのゴーシュ

**宮沢賢治**[^1]

ゴーシュは町の活動写真館でセロを弾く係りでした。
けれどもあんまり上手でないという評判でした。
上手でないどころではなく実は仲間の楽手のなかではいちばん下手でしたから、
いつでも楽長にいじめられるのでした。
ひるすぎみんなは楽屋に円くならんで今度の町の音楽会へ出す
第六{交響曲|こうきょうきょく}の練習をしていました。

![挿絵]('image/fig334.png'){width=300}

[^1]:
https://ja.wikipedia.org/wiki/%E5%AE%AE%E6%B2%A2%E8%B3%A2%E6%B2
```

[4] https://vivliostyle.github.io/vfm/#/ja/vfm

基本的には GitHub でサポートされている Markdown の方言である GFM (GitHub Flavored Markdown) の記法は使えます。しかし、上のサンプルにはなにか見慣れない記法がありますね。それぞれ説明します。

ルビ

`{親文字|ヨミ}` とすることでルビが振れます。便利。べんり　ミライ。~~悔やむと書いてミライと読ませることもできますね！~~

脚注 (後注)

文章内に `[^n]` を置きその文の後の箇所で `[^n]: hoge` とすると脚注をつけることができます。ちなみに、`@vivliostyle/theme-techbook` のテーマパッケージを使用している場合は `<span class="footnote">hoge</span>` とすると本文 (ページ) の下に注記をつけることができます。

余談ですが、脚注をつける記法は実は GFM で同様にサポートされていたりします。

Sass を使う

この節では Sass と一緒にビルド/プレビューする方法をご紹介します。CSS を使ってスタイルを当てるのですが CSS よりも Sass の方がいろいろと楽だったりします。

Create Book でプレビューするとき、直接 .scss ファイルを読み込めないため、スタイルを調整しながらプレビューするといったことが困難です。

これを解決するために、`npm-run-all` を使います。`run-p` というコマンドがあり、パラレル実行ができるためこれを活用します。npm や yarn でインストールして

```
$ yarn add --dev npm-run-all
```

package.json の script に以下を追加します。

```
"start": "run-p preview watch:scss",
"preview": "vivliostyle preview",
"watch:scss": "sass --watch scss:css"
```

これで `yarn start` とすると SCSS のビルドとプレビューが同時にできます。便利。

GitHub Actions でビルドする

この部誌は GitHub 上で管理されており、各部員がブランチを切り、PR を出す…といった感じで書かれています。

しかし、現状どのような見た目になっているかを確認できない部員もいたりします。そのため、GitHub Actions を活用してビルドし、自動で `publish` ブランチに push されるような仕組みにすることにしました。以下のワークフローを作成するとできます。

.github/workflows/build.yml

```yaml
name: Build
on:
  workflow_dispatch:

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    env:
      TZ: Asia/Tokyo
      GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
    steps:
      - name: Checkout
        uses: actions/checkout@v2.3.4
        with:
          ref: main
      - name: Setup Node
        uses: actions/setup-node@v2.1.5
        with:
          node-version: '14.16.0'
      - name: Get yarn cache directory path
        id: yarn-cache-dir-path
        run: echo "::set-output name=dir::$(yarn cache dir)"
      - name: Cache deps
        uses: actions/cache@v2.1.6
        with:
          path: ${{ steps.yarn-cache-dir-path.outputs.dir }}
          key: ${{ runner.os }}-yarn-${{
hashFiles('yarn.lock') }}
          restore-keys: |
            ${{ runner.os }}-yarn-
      - name: Install deps
        run: yarn install --frozen-lockfile
      - name: Install ghostscript
        run: |
          sudo apt-get -yqq install libgbm1 ghostscript
```

```
          sudo apt install poppler-utils poppler-data
      - name: Build
        run: yarn build
      - name: Build Press-Ready
        run: yarn press-ready
      - name: Deploy
        uses: s0/git-publish-subdir-action@develop
        env:
          REPO: self
          BRANCH: publish
          FOLDER: public
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

`workflow_dispatch` ではなく main ブランチ等への push をトリガーにしたほうが楽かもしれないですね。

この部誌をビルドする

前述したとおりこの部誌は GitHub のリポジトリで管理されているのでそれを clone してきましょう。

```
$ git clone git@github.com:kss-pc-club/book-2022.git
```

依存関係をインストールし、`yarn build` とするとビルドができます。

```
$ yarn install
$ yarn build
```

`public/book.pdf` が作られます。

入稿用のデータを `yarn press-ready` というコマンドで作ることができますが、ghostscript と poppler-utils のインストールが必要なので注意してください。

おわりに

ある程度しっかりとした本を既に知っている技術で作れました。

CSS 組版で本を作りたい！Markdown で本を書きたい！...そんな方はぜひ Vivliostyle で始めてみませんか？

ICPCへのお誘い

Asa

はじめに

こんにちは、第3期生の土屋です。ネットでは「Asa (@a01sa01to)」として活動しています [1] 。2021年3月に古河中等教育学校を卒業し、現在は埼玉大学工学部情報工学科に所属しています。この度、細島部長にお声がけいただき、このような形で部誌の編集に携わることとなりました。

技術系（特にプログラミング関連）の記事がほとんどである、この部誌を読んでくださっている方の中には、「競技プログラミングについて聞いたことがある / すでに参加している」という方がいると思います。そこで、国際大学対抗プログラミングコンテスト（ICPC）に関する記事を書きたいと思います。

ICPCってなに？

国際大学対抗プログラミングコンテスト（International Collegiate Programming Contest・通称 ICPC） [2] は、同じ大学の3人で1チームを作り、プログラミングの問題を解く大会です。中学生・高校生の皆さんは、「国際情報オリンピックのチーム参加版」と捉えていただけるとわかりやすいかもしれません。

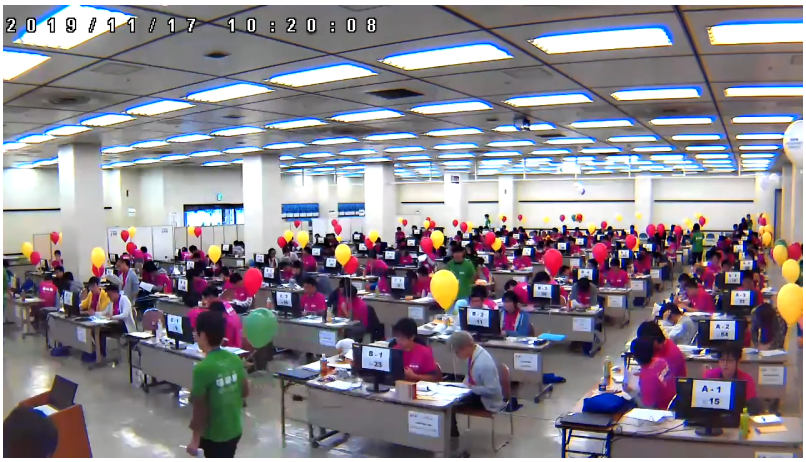
▲図1: ICPCのロゴ（http://icpc.foundation/ より引用）

[1] ~~ここまでの情報で私のフルネームと誕生日がわかるらしいですよ~~

[2] https://icpc.global/（日本語版: https://icpc.iisf.or.jp/ ）

この大会には世界各国から毎年3万人以上が参加しており、日本からも様々な大学から参加しています。2021年度、私の所属する埼玉大学からも2チーム出場し、私もそのうちの1チームに参加していました [3] 。

大会の流れとしては、「国内予選→アジア地区大会→世界大会」といった流れです。情オリに似ていますね。国内予選では3時間で6-7問、アジア地区大会では5時間で10問程度の問題を解きます。情オリとは違い、得点ではなく、解いた問題数で競います。また、アジア地区大会で利用できるコンピュータもチームで1台と制限があります [4] 。そのため、1人が解いている間にほかの人がアイデアを出すなどといったチームワークが重要になります。さらに知りたい方は、ICPC日本公式団体に掲載されている「3分でわかるICPC」（https://icpc.iisf.or.jp/acm-icpc/3min/ ）も併せてご覧ください。



▲ 図2: 2019年度アジア地区大会の様子（https://youtu.be/MwD254vH3Pw）

どんな問題を解くの？

「○○くらいの難易度です！」と言っても個人差があると思うので、実際に出された問題を見てみましょう。

[3] ちなみに私のチームの結果は、予選43位、アジア地区大会11位でした。もう1チームは予選104位で、アジア地区大会に進出できませんでした。

[4] 2020年度・2021年度はオンラインだったため、1人1台でした。なお国内予選では、コードではなく、自分のコンピュータで出た答えを提出します。~~時間制限は実質無限です。~~

2021年度 国内予選 A問題

https://icpc.iisf.or.jp/past-icpc/domestic2021/contest/all_ja.html#section_A より、問題文の概要を書いてみます。

4つのお椀とその中に入ったビー玉がある。1つもビー玉が入っていないお椀があることがあるが、少なくともどれか1つのお椀にはビー玉が入っていることが保証されている。
4つのお椀に対して、空でないお椀が1つだけになるまで、以下の操作を繰り返す。

1. 空でないお椀のうち、ビー玉が少ないものを選ぶ。2つ以上あるときは、一番左のお椀を選ぶ。
2. 選んだお椀以外の、空でないすべてのお椀から、選んだお椀と同じ数のビー玉を取り除く。ただし、選んだお椀のビー玉はそのままにしておく。

さて、最終的にお椀に残ったビー玉の個数はいくつ？

制約

- それぞれのビー玉の個数は、0個以上100個以下。
- 少なくとも1つは0ではない。
- データセットは100個以内。
- `0 0 0 0` は終了の合図。

サンプルの入出力を見てみましょう。

```
10 8 4 6
0 21 7 35
5 45 13 3
52 13 91 78
0 0 0 0 (←入力終了の合図)
```

```
2
7
1
13
```

さて、この問題をあなたならどう解きますか？

まず思いつくのは、単純にシミュレーションしてみることです。

具体的には、ビー玉の残ったお椀が1以下ではない限り、次の手順を繰り返します。

1. お椀を、ビー玉の個数でソートする
2. 一番少ないお椀を選び、そのビー玉の個数を他から減らす
3. ビー玉が0以下になったお椀を削除する

このような方法でも、計算量は高々 $O(T \sum a_i)$ (10^4 くらい) なので、高速です [5] 。なお、データセット数を T としています。

ans.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  while (true) {
    vector<int> a(4);
    for (int i = 0; i < 4; ++i) cin >> a[i];

    if (a[0] + a[1] + a[2] + a[3] == 0) break;

    while (a.size() > 1) {
      sort(a.begin(), a.end());

      for (int i = 1; i < a.size(); ++i) a[i] -= a[0];

      auto itr = remove_if(a.begin(), a.end(), [](int x) {
        return x <= 0;
      });
      a.erase(itr, a.end());
    }
    cout << a[0] << endl;
  }
  return 0;
}
```

ICPCでは、このような単純な繰り返し処理や条件分岐などができれば、チームに貢献できます。実際、当時 AtCoder茶色 の私でも、解くことができました。

参考までに、さらに高速な方法をご紹介します。それは、最大公約数を用いる方法です。

---

[5] 各手順で少なくとも1つは取り除かれるので、最大でビー玉の個数分の手順しか行われません。

実は、それぞれのデータセットの答えは、$a_1, \ldots, a_4$ の最大公約数になることが証明できます [6] 。

これを用いると、$O(T \log(\min a_i))$ で計算できます。

ans.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  while (true) {
    int a, b, c, d;
    cin >> a >> b >> c >> d;

    if (a + b + c + d == 0) break;

    int g = a;
    g = gcd(g, b); g = gcd(g, c); g = gcd(g, d);

    cout << g << endl;
  }
  return 0;
}
```

2020年度 アジア地区大会 A問題

本当は2021年度の問題を載せようと思ったのですが、まだ公開されていませんでした...。（何かの体積を計算させる問題だったと記憶しています）

そこで、2020年度の問題を載せます！ [7]

> $N \times N \times N$ の立方体に収まるある立体を、$x-y, y-z, z-x$ 平面それぞれに射影した図が与えられます。与えられた図に適する図形が存在するか判定してください。
> 制約: $1 \leq N \leq 100$

[6] https://icpc.iisf.or.jp/past-icpc/domestic2021/commentaries.html#A

[7] https://icpc.iisf.or.jp/past-icpc/regional2020/problems-2020.pdf すべて英語です。本番では、機械翻訳は許されません（辞書はOK）。

解法は、$N \times N \times N$ すべてが埋まった立方体から、「削って」いき、最終的に出来上がった立体が、条件が満たしているかを確認する方法です。

アジア地区大会では、自分のコンピュータではなく、ジャッジ用コンピュータで正誤判定されるため、時間制限が定められています。この問題では2秒ですが、以上の解法は $O(N^3)$ なので、ACになります。

実装は大変ですが、これも問題なく解けると思います。

ans.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define rep(i, n) for (int i = 0; i < (n); ++i)

int main() {
  int n;
  cin >> n;
  vector yz(n, vector<bool>(n)), zx(n, vector<bool>(n)),
xy(n, vector<bool>(n));
  rep(i, n) {
    string s; cin >> s;
    rep(j, n) yz[j][n - i - 1] = (s[j] == '1');
  }
  rep(i, n) {
    string s; cin >> s;
    rep(j, n) zx[j][n - i - 1] = (s[j] == '1');
  }
  rep(i, n) {
    string s; cin >> s;
    rep(j, n) xy[j][n - i - 1] = (s[j] == '1');
  }

  vector ans(n, vector(n, vector<bool>(n, true)));
  rep(i, n) rep(j, n) if (!yz[i][j]) rep(k, n) ans[k][i][j] =
false;
  rep(i, n) rep(j, n) if (!zx[i][j]) rep(k, n) ans[j][k][i] =
false;
  rep(i, n) rep(j, n) if (!xy[i][j]) rep(k, n) ans[i][j][k] =
false;

  rep(i, n) rep(j, n) {
    if (yz[i][j]) {
      bool chk = false;
      rep(k, n) if (ans[k][i][j]) chk = true;
      if (!chk) { puts("No"); return 0; }
    }
  }
  rep(i, n) rep(j, n) {
```

```
    if (zx[i][j]) {
      bool chk = false;
      rep(k, n) if (ans[j][k][i]) chk = true;
      if (!chk) { puts("No"); return 0; }
    }
  }
  rep(i, n) rep(j, n) {
    if (xy[i][j]) {
      bool chk = false;
      rep(k, n) if (ans[i][j][k]) chk = true;
      if (!chk) { puts("No"); return 0; }
    }
  }
  puts("Yes");
  return 0;
}
```

最後に

ICPCの問題を実際に見てみましたが、「こんなの解けるわけがない！」というものではなかったと思います。むしろ、少し考えたうえで、プログラミングの初歩である「繰り返し処理」「条件分岐」を使いこなすだけで、最初の問題は解けます。

競技プログラミングをするうえで、ICPCは一生に5回程度しか参加できないため、貴重な体験になることは間違いないです。そしてICPCは、プログラミング力ではなく、チームワークが鍵です。チーム戦で生まれる連帯感というのは、なかなか体験できないと思います。

この記事を通じて、少しでも興味を持っていただけたのであれば幸いです。大学に入学した際には、ぜひともICPCに参加してみてください！！！！ [8]

最後までお読みいただき、ありがとうございました！

[8] 参加資格のない方は...順位表実況や過去問などでも楽しめます！

ワンクリック詐欺サイト解剖してみた

張替健太/hnm876_md

はじめに

こんにちは。古河中等教育学校６期生の張替健太です。僕の記事の前に書かれた小難しそうな記事を見て、「プログラミング難しそうだな」とか「変態の領域なのかな」感じている人がいるかもしれません。実際そうですが、Webプログラミングは意外と簡単です。どれほど簡単なのかWebプログラミングの基本が詰まった教科書「ワンクリック詐欺サイト」を 見ながら体感していただければ幸いです。

ワンクリック詐欺サイトってなあに

その名の通り、一見ありふれたボタンをクリックした途端に「会員登録完了」や「〇〇日までに料金を払ってね」などの脅しがたくさん出力されるWebサイトのことです。2018年時点では存在していましたが、今でもあるんでしょうか。もし引っかかっても、僕のように電話はかけずに無視してブラウザバックしましょう。

ワンクリック詐欺サイト解剖

支払い期限タイマーの表示

▲図1: ハッタリタイマー

　こんな感じでクリックした人を焦らせて個人情報入力させようという魂胆です。冷静に見てみるとおもちゃのような仕組みです。

index.html

```
<div>
    <p><span id="hour"></span>hours</p>
    <p><span id="min"></span>minutes</p>
    <p><span id="sec"></span>seconds</p>
</div>
```

timer.js

```
const hour = document.getElementById("hour");
const min = document.getElementById("min");
const sec = document.getElementById("sec");

function countdown(){
  const now = new Date(); //現在時刻を取得
  const tomorrow = new
Date(now.getFullYear(),now.getMonth(),now.getDate()+1); //明日
の0:00を取得
```

```
  const diff = tomorrow.getTime() - now.getTime(); //時間の差を
取得（ミリ秒）

  //ミリ秒から単位を修正
  const calcHour = Math.floor(diff / 1000 / 60 / 60);
  const calcMin = Math.floor(diff / 1000 / 60) % 60;
  const calcSec = Math.floor(diff / 1000) % 60;

  //取得した時間を表示（2桁表示）
  hour.innerHTML = calcHour < 10 ? '0' + calcHour : calcHour;
  min.innerHTML = calcMin < 10 ? '0' + calcMin : calcMin;
  sec.innerHTML = calcSec < 10 ? '0' + calcSec : calcSec;
}
countdown();
setInterval(countdown,1000);
```

1. HTMLで書かれたファイルのhour,min,secをJavaScriptという言語で取得する
2. 現在の時刻と明日の0時を取得して差分を出して取得する
3. modの考え方を利用して時間、分、秒を算出する
4. 見やすく二桁表示にして、カウントダウンする

大体こんな手順です。あんまり怖くないですね。

機密(?)情報の表示

情報が既に抜き取られていて逃げられない…なんてことはありません。端末情報やブラウザのバージョンなんかは開示されているので、誰でも簡単に抜き取ることができます。

kowakunaiyo~.js

```
<script src="platform.js"></script>
    <script>
      document.write(platform.name); //Firefox
      document.write(platform.version); //69.0
      document.write(platform.os.toString()); //OS X 10.14
      document.write(platform.layout); //Gecko
    </script>
```

上部のように書くと、こんな感じに抜き取ることができます。

▲図2: 抜き取り使用例

```
// on IE10 x86 platform preview running in IE7 compatibility
mode on Windows 7 64 bit edition
platform.name; // 'IE'
platform.version; // '10.0'
platform.layout; // 'Trident'
platform.os; // 'Windows Server 2008 R2 / 7 x64'
platform.description; // 'IE 10.0 x86 (platform preview;
running in IE 7 mode) on Windows Server 2008 R2 / 7 x64'

// or on an iPad
platform.name; // 'Safari'
platform.version; // '5.1'
platform.product; // 'iPad'
platform.manufacturer; // 'Apple'
platform.layout; // 'WebKit'
platform.os; // 'iOS 5.0'
platform.description; // 'Safari 5.1 on Apple iPad (iOS 5.0)'

// or parsing a given UA string
var info = platform.parse('Mozilla/5.0 (Macintosh; Intel Mac
OS X 10.7.2; en; rv:2.0) Gecko/20100101 Firefox/4.0 Opera
11.52');
info.name; // 'Opera'
info.version; // '11.52'
info.layout; // 'Presto'
info.os; // 'Mac OS X 10.7.2'
info.description; // 'Opera 11.52 (identifying as Firefox
4.0) on Mac OS X 10.7.2'
```

上部は抜き取れる情報と抜き取り方みたいなことが書かれています。重要なのはこの内容ではなく、いろんな情報が誰でも抜き取れるんだなあということです。このようなライブラリ（https://github.com/bestiejs/platform.js）という誰でも使えるものがあるのでハッキングとかではないんだなと思っておくと気が楽です。

写真を撮ったかのような演出

　謎のシャッター音と点滅で「もしかして写真を撮られたのかも」と 思うかもしれませんがそんなことありません。演出です。

▲図3: パシャ

　今ではWebポリシー改定によってシャッター音を急に鳴らすことはかなり難しくなっています。できないことはありませんが、実装するためのSPA(シングルページアプリケーション)開発者・経験者が少ないらしいので、こちらの解剖はせず、点滅の方の解剖をしていこうと思います。

camera.html

```
<div class="camera">
    <img
src="https://media.discordapp.net/attachments/757627473427955862

</div>
<style>
.camera{
  animation: flash 0.3s linear 1s;
  width:256px;
  height:auto;
  display: block;
  margin-left: auto;
```

```
  margin-right: auto;
}

@keyframes flash {
  0%,100% {
    opacity: 1;
  }

  50% {
    opacity: 0;
  }
}

  </style>
```

カメラのイラストが点滅して少し怖くなると言ったハッタリですね。これはcssアニメーションというものを使っています。カメラの画像の透過率を100%→0%→100%と変化させて、flashというアニメーションを作っています。後は上手に点滅しているように見せるために 1.0秒後に0.3秒間等しい速度でflashを使えとカメラのイラストに指定すれば完成です。

おわりに

いかがでしたか、よければ今後の糧にでもしていただけると幸いです。近年、情報分野の発達は目覚ましいので、こんなものより より高度な技術でみなさんを騙す人が出てくることは必至でしょう。そのためにもぜひ自ら情報の力をつけてみてはいかがでしょうか。

参考・参照

- https://www.webdlab.com/jquery/jquery-new-date/
- https://tcd-theme.com/2021/08/javascript-countdowntimer.html
- https://qiita.com/TD12734/items/671064e8fce75faea98d
- https://teratail.com/questions/265276
- http://kentaro-shimizu.com/lecture/fraud/pay.html
- https://coco-factory.jp/ugokuweb/
- https://deadlinetimer.com/

ニュートン法で近似をしよう!!

Anthony

はじめに

世の中には、特別な記号を使わずして表すことのできない、多くの数が存在しています。具体的に言うと、$\sqrt{2}$、$\log 2$、π(円周率)、e(ネイピア数)などの無理数と呼ばれる数がそれにあたります。

そして、我々人間はその値がどの程度であるか確かめるために様々な方法でその値を近似してきました。具体的には、円周率を正多角形を用いて求めてみたり、不等式で値を概算してみたり、級数を用いたりなどです。

今回はそんな人間が生み出してきた近似方法の一つである、ニュートン法について焦点を当てて考えていこうと思います。

ニュートン法とは

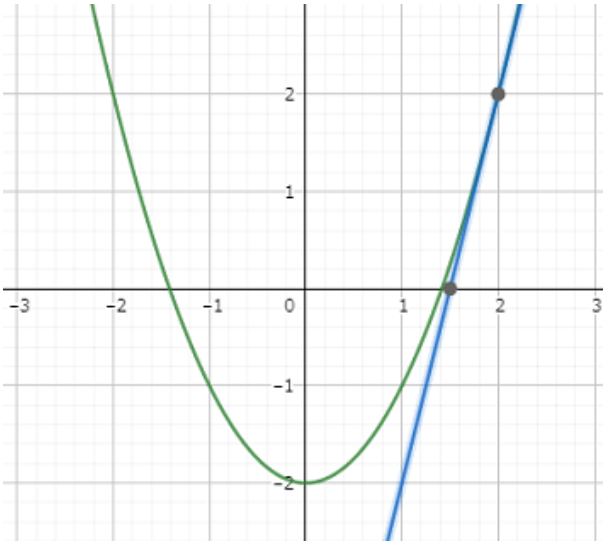
まず、ニュートン法の説明をします。ニュートン法とは、簡単に言うと接線を引きその交点を求めることによって、求めたい値を近似していく方法です。この時、接線の傾きを求める関係上、$f(x)$ は微分可能な関数でなくてはいけません。

厳密に言うと $f(x) = 0$ の解 (求めたい数) を接線を用いることによって近似します。ここで、$f(x)$ の接線を求めるために、求めたい数より大きな x 座標全てで微分可能であることを前提とします。最初の x 座標を x_1 とし、最初の y 座標を $f(x_1)$ とします。

一回目の操作の時、接線の傾きは $f'(x_1)$ で、これは点 $(x_1, f(x_1))$ を通るから接線の方程式は

$$y = f'(x_1)(x - x_1) + f(x_1)$$

となる。



▲図1: 接線を引いた様子

これを繰り返すことにより、n 回目の接線の傾きは $f'(x_n)$、接点は点 $(x_n, f(x_n))$ であるから n 回目の接線の方程式は

$$y = f'(x_n)(x - x_n) + f(x_n)$$

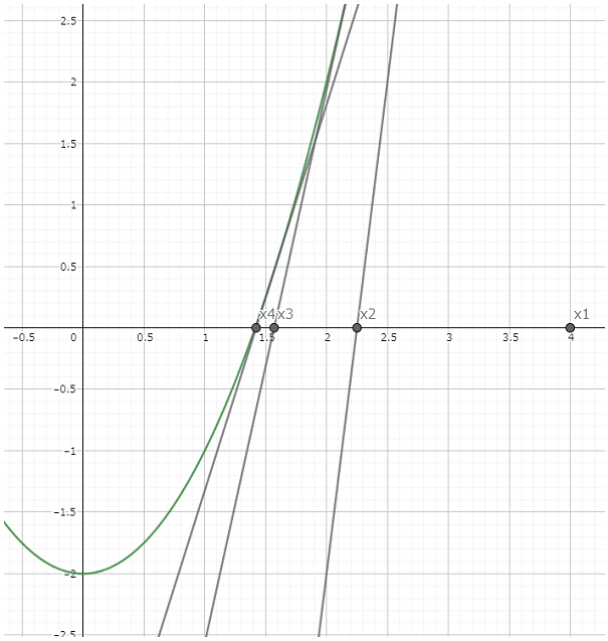
となる。

この時、$y = 0$ の時の x 座標が x_{n+1} となるので、$y = 0, x = x_{n+1}$ として移項すると

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

という式が得られる。

この式の n を限りなく大きくすることによって得られる x が答えとなります。



▲ 図2: $x_1 = 4$としたときの x_4 まで求めた様子

図からなんとなく収束していきそうな感じがしますが、証明によりきちんと値が収束することを示します。

証明を以下の3ステップで行います。

1. $\sqrt{a} < x_{n+1} < x_n$ を示す

$x_1 > \sqrt{a} > 0$（前提より）
数学的帰納法により $x_n > \sqrt{a}$を示す
(1) $n = 1$のとき $x_1 > \sqrt{a}$
(2) $n = k$のとき成り立つとすると $x_k > \sqrt{a}$
$n = k + 1$ のとき

$$x_{k+1} = \frac{x_k^2 + a}{2x_k} > \frac{(a + a)}{2\sqrt{a}} = \sqrt{a} > 0$$

よって $n = k$ が成り立つとき、$n = k + 1$ も成り立つ
(1)(2) より全ての自然数kにおいて $x_n > \sqrt{a}$ が成り立つ

ここで

$$x_n - x_{n+1} = x_n - (\frac{x_n}{2} + \frac{a}{2x_n}) = \frac{x_n^2 - a}{2x_n}$$

$x_n > \sqrt{a}$ より、$x_n^2 - a > 0$
故に

$$\frac{x_n^2 - a}{2x_n} > 0$$

したがって $x_{n+1} < x_n$
よって

$$\sqrt{a} < x_{n+1} < x_n$$

2. $x_{n+1} - \sqrt{a} < \frac{1}{2}(x_n - \sqrt{a})$ を示す

$$\begin{aligned}\frac{1}{2}(x_n - \sqrt{a}) - x_{n+1} + \sqrt{a} &= \frac{\sqrt{a}}{2} - \frac{a}{2x_n} \\ &= \frac{\sqrt{a}}{2x_n}(x_n - \sqrt{a})\end{aligned}$$

$x_n > \sqrt{a}$ より $\frac{\sqrt{a}}{2x_n}(x_n - \sqrt{a}) > 0$
したがって

$$x_{n+1} - \sqrt{a} < \frac{1}{2}(x_n - a)$$

3. 極限値が $\sqrt{a}$ に収束することを示す

2.で求めた不等式を繰り返すことにより

$$\begin{aligned} x_n - \sqrt{a} &< \frac{1}{2}(x_{n-1} - \sqrt{a}) \\ &< (\frac{1}{2})^2(x_{n-2} - \sqrt{a}) \\ &< \cdots < (\frac{1}{2})^{n-1}(x_1 - \sqrt{a}) \end{aligned}$$

$x_1 > \sqrt{a}$ より

$$0 < x_n - \sqrt{a} < (\frac{1}{2})^{n-1}(x_1 - \sqrt{a})$$

ここで

$$\lim_{n\to\infty}(\frac{1}{2})^{n-1}(x_1 - \sqrt{a}) = 0$$

はさみうちの原理により

$$\lim_{n\to\infty}(x_n - \sqrt{a}) = 0$$

したがって

$$\lim_{n\to\infty} x_n = \sqrt{a} \quad \blacksquare$$

プログラミングしてみよう!!

　では、先ほどこの方法が成り立つことが確認できたのでプログラムによって再現していきましょう。今回は $\sqrt{2}$ の場合について作っていくことにします。

　以下のプログラムは、すべて `C++` で記述されています。

　さっそくですが、下のようなコードになります。

```cpp
// コード(cpp)
#include <bits/stdc++.h> // おまじない
using namespace std;

// プログラムのはじまり
```

```
int main() {
  double ans = 20;

  // 十分な回数(10000回)繰り返す
  for (int i = 0;i < 10000; i++) {
    // ニュートン法の式の通りに値を更新
    ans = ans -((ans*ans)-2)/(2*ans);

  }
  // 小数第15位までansの数字を表示する
  printf("%.15f\n",ans);

} // プログラムの終わり
```

これだけを見てもよく分からないと思うので補足をしてきます。

①プログラムの基本構造(C++)

プログラミング言語には日本語や英語と同じように文法が存在しており、一定の規則の上で書かなくてはいけません。下の部分はプログラムの開始、終了をコンピューターに伝える部分となっています。

```
#include <bits/stdc++.h>
using namespace std;

int main() {

}
```

②変数の宣言

今回扱う数字は実数なので、実数の変数を定義し、初期値(= x1)としておきます。

```
double ans = 20;
```

これは $x_1 = 20$ としています。また、この初期値は求めたい数より大きければどの数でもかまいません。（自然数に限らず、実数でもOKです。）

③繰り返し

`{}`で囲まれている部分を10000回繰り返します。

```
for (int i = 0;i < 10000; i++) {

}
```

ちなみに下の場合は10回の繰り返しとなります。

```
for (int i = 0;i < 10; i++) {

}
```

④ニュートン法の計算部分

初めに、ほとんどのプログラミング言語における「=」は等式を意味しているものではなく、代入を意味します。例えば `a = 20` を例にとって考えてみます。数学では `a` と `20` は同じ値であるということを意味していますが、プログラミング言語では「変数 `a` に `20` を**代入する**」ということを意味します。また、掛け算の記号は `×` でなく `*` を、割り算の記号は `÷` でなく `/` を用います。

このプログラムでは、 `ans` の値を更新していき、値を求めます。

⑤表示

10000回計算が終わった後、ansの値を小数第15位まで表示して改行します。

```
printf("%.15f\n",ans);
```

結果

前節で書いたプログラムの初期値を変えることによって値がきちんと求めることができるかと、収束のスピードを見ていきたいと思います。

また、収束したとは正確な値との差が非常に小さくなった時とします。しかし、その差の基準はいったいどうすれば良いでしょうか？

まず、x_k は常に求めたい数より大きくなる、また n が増加するにつれて x_k は小さくなって求めたい数に近づく、という二つの事象から次の x_{k+1} との差が非常に小さいとき、ほとんど近似できた、すなわち、$(x_k - x_{k+1}) < \epsilon$ (k:自然数、ϵ:計算精度を決める非常に小さな定数）を満たしたら近似できたということにしましょう。

ここで、1回の計算にかかる時間はほぼ同じと言えるので、上記の条件式を満たすようになるまでに計算した回数が小さいほど収束が早いといえます。
近似できたのが何回目の操作であるかも表示するプログラムの一例を以下に示します。

```cpp
// コード(cpp)
#include <bits/stdc++.h> // おまじない
using namespace std;

// プログラムのはじまり
int main() {

  double ans = 20; // x1の値
  int i;

  // 十分な回数(10000回)繰り返す
  for(i = 0; i < 10000; i++){
    double tmp = ans;

    // ニュートン法の式の通りに値を更新
    ans = ans -((ans*ans) - 2)/(2*ans);

    // 差が0.0000001になったら計算終了
    if(tmp - ans <= 0.0000001) break;
  }

  // ansの値と誤差が小さくなった時のiを表示
  printf("%.15f\n%d\n",ans,i);

} // プログラムの終わり
```

このプログラムを用いることで、計算にかかった回数 = 速度が分かります。

x1の値	計算した回数
2	4
20	7
200	11
2000	14
20000	17
200000	21

初期値が求める値に近いほど収束するスピードが早いことが分かります。まあ、当たり前と言えばあたりまえなのですが。

例えば、ある初期値 x_1 によって得られる x_2 が別の初期値 x_1' と同じになった時、単純に計算にかかる時間は1回分短くなります。

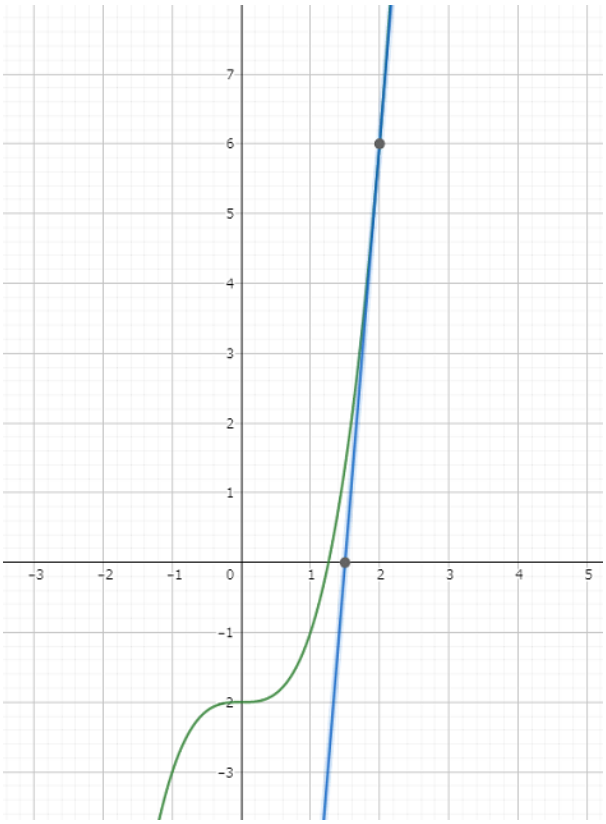
応用方法

実は、このニュートン法は $\sqrt{2}$ の時だけでなく、$\sqrt{n}$ (n は任意の正の実数)のときにも成り立ちます。

さらにいうと、これは2乗根の時だけではなく1.5乗根、3乗根、4乗根、...m 乗根(m は任意の正の実数)の時にもすべて成り立ちます。

これは、最初に求めた式 $x_{n+1} = x_n - f(x_n)/f'(x_{n+1})$ をもとに、2次式が成り立つことが分かります。

この式の $f(x)$ の次数を変えても、(例えば $f(x) = x^3 - 2$ のときでも)成り立ちます。



▲ 図3: 3次関数の場合の様子

したがって、$\sqrt[k]{m}$ (k, m はともに任意の正の実数)の値も同様の方法で近似することができます。ニュートン法の式 $x_{n+1} = x_n - f(x_n)/f'(x_n)$ を用いることは変わらないので $\sqrt[k]{c}$ を求める場合、$f(x) = x^k - c$ とおいて、少し変えると作ることが出来ます。

また、前節では調べる数字を $\sqrt{2}$ のみに絞って考えているため変数が少なかったのですが、ここで「何乗根であるか？」と「根の中の数字」を表す変数を追加し、新たにコードを書き直します。

コードは以下の通りになります。

```cpp
// コード(cpp)
#include <bits/stdc++.h> // おまじない
using namespace std;

// プログラムのはじまり
int main() {

  double ans = 20; // 初期値
  double  k = 3;   // 何乗根か？
  double  c = 2;   // 根の中の数字は何か？

  // 十分な回数(10000回)繰り返す
  for (int i = 0;i < 10000; i++){

    // ニュートン法の式の通りに値を更新
    ans = ans -(pow(ans,k)-c)/(k*pow(ans,k-1));

  }

  // 小数第15位までansの数字を表示する
  printf("%.15f\n",ans);

} // プログラムの終わり
```

この時、$f'(x) = kx^{k-1}$ であるから、累乗をプログラムで表現する必要が出てきます。そこで、a^b を `pow(a,b)` という標準の関数で計算します。

次節では収束条件とその証明を書きます。

収束条件とその証明

収束条件： $f(x_0) > 0$、$f(a) < 0$、$f'(x) > 0$、$f''(x) > 0$ の全てを満たす。

すなわち、x についての閉区間 $[a, x_0]$ において $f(x)$ が下に凸な単調増加なグラフであり、$f(x) = 0$ を満たす解が一つある状態です。

証明
x が区間 $[a, x_0]$ にある時、$f''(x) > 0$ より

$$\int_x^{x_0} f''(x)dx > 0$$

が成り立つ。
計算し、式を変形すると　$f'(x) < f'(x_0)$ したがって

$$\int_x^{x_0} f'(x)dx < \int_x^{x_0} f'(x_0)dx$$

が成り立つ。
計算すると

$$f(x_0) - f(x) < f'(x_0)(x_0 - x)$$

ここで $f(a) < 0 < f(x_0)$ が成り立つから中間値の定理より $f(\alpha) = 0$ を満たす α が区間 (a, x_0) に存在する。
また、関数 $f(x)$ は単調増加関数であるから、逆関数が存在する。（区間 $[a$、 $x_0]$ において $f(k) = f(l)$ でかつ $k \neq l$ を満たす数が存在しないものを逆関数という。）
よって　$f^{-1}(x)$ を $f(x)$ の逆関数とすると　$\alpha = f^{-1}(0)$ と表せる。
ここで $x = \alpha$ とおくと

$$f(x_0) - f(\alpha) < f'(x_0)(x_0 - \alpha)$$

ここで $f(\alpha) = 0, f'(x) > 0, f(x_0) > 0$より

$$\alpha < x_0 - \frac{f(x_0)}{f'(x_0)} < x_0$$

また、$x_0 - f(x_0)/f'(x_0) = x_1$ より

$$\alpha < x_1 < x_0$$

これを繰り返すことによって $\alpha < \cdots < x3 < x2 < x1$
また、x は単調減少であり下限があるため数列 $\{x_n\}$ は収束する。ここで数列 $\{x_n\}$ が β に収束するとする。
$x_{n+1} = x_n - f(x_n)/f'(x_n)$ より

$$\lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty}(x_n - \frac{f(x_n)}{f'(x_n)})$$

ここで $\lim_{n\to\infty} x_n = \beta, \lim n \to \infty x_{n+1} = \beta$ であるから

$$\beta = \beta - \frac{f(\beta)}{f'(\beta)}$$

これより $f(\beta) = 0$
関数 $f(x)$ は逆関数を持つため $\beta = f^{-1}(0)$ よって

$$\beta = \alpha \quad (\because \alpha = f^{-1}(0))$$

したがって、数列 x_n は α に収束し、$f(\alpha) = 0$ となるので
ニュートン法の数列 x_n は「$f(x_0) > 0, f(a) < 0, f'(x) > 0, f''(x) > 0$ の全てを満たす」とき $f(x) = 0$ の解 $\alpha \quad (a < \alpha < x_0)$ に収束する。 ■

おわりに

プログラミングと聞くと、難しくて初心者にはできないものだと考えてしまうかもしれませんが、この記事のように簡単な計算や、文字を表示させるなど、単純な動作だけなら今すぐにでも実装・実行できます。

それに、今の時代コンパイラはオンライン上のものを使うこともでき、インターネットで検索すれば実装方法も簡単に見つけることができますし、自分でアレンジすることができます。

ぜひ、皆さんも簡単なプログラムから初めて、自分の作りたいものを作りましょう！

参考文献等

1. ニュートン法 wikipedia
   https://ja.wikipedia.org/wiki/ニュートン法
2. ニュートン法とは？ ～定義と性質～ - 理数アラカルト
   https://risalc.info/src/newtons-method-properties.html
3. グラフ作成 Geogebra
   https://www.geogebra.org/calculator
4. 今回使用したオンラインコンパイラ : paiza.io
   https://paiza.io/ja

KSS PC BOOK 2022

2022 年 5 月 x 日　初版発行
著　者　KSS PC Club
印刷所　xxxxx