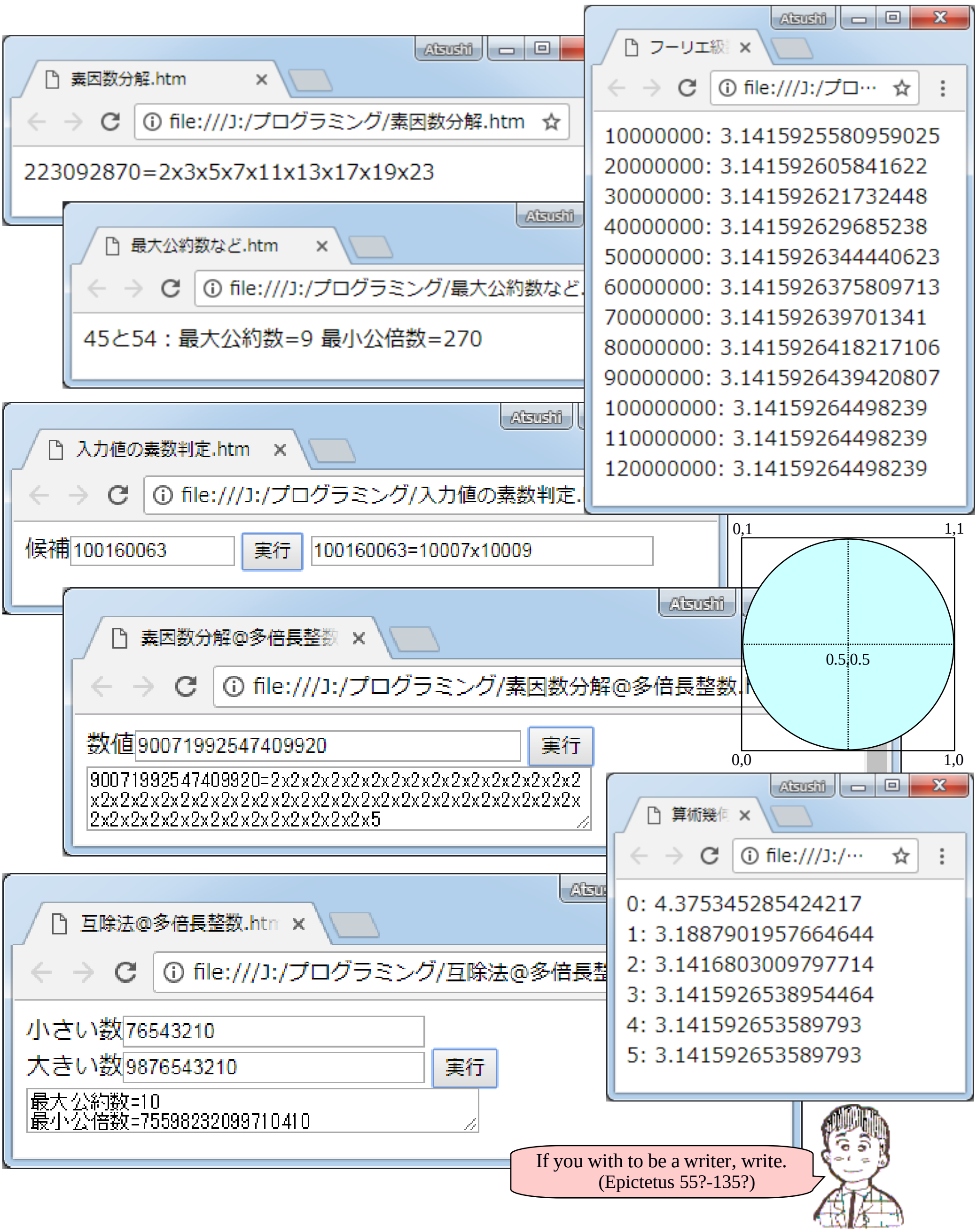
ブラウザに計算をさせてみよう！

2018 年 1 月　加藤 厚

― スクリプトで求める素因数・最大公約数と最小公倍数・円周率 ―



目次

【補足】

Ａ５判見開きでの閲読を想定して作成した本冊子の内容は、筆者が Windows 7 を OS とする PC 上で確認した結果をまとめたものです。お気づきの点などありましたら atsushi.kato.1958@gmail.com までお知らせ頂ければ嬉しく思います。

【用語集】

アルゴリズム（algorithm）：定型的な解法　例：エラトステネスの篩（ふるい）

円周率（π）：円の直径に対する周長の比　例：Yes(3), I(1) have(4) a(1) number(6).

英語での覚え方の一例

改行記号：web 頁では
（break）、script では"¥n"（next）で「改行」

級数（series）：一定の規則で項が連なる式が与える数　例：等差／等比級数

最小公倍数（L.C.M.）：複数の整数に共通する正で最小の倍数

最大公約数（G.C.D.）：複数の整数に共通する最大の約数

スクリプト（script）：処理の手順を記述した簡易プログラム　例：ECMAScript

素因数分解（resolution into prime factors）：整数を素数の積で表すこと

素数（prime number）：1 とその数以外に約数を持たない正の整数⇔合成数

Ｄ＆Ｄ(Drag & Drop)：押さえながら移動した対象を特定の場所で離すこと

ブラウザ（browser）：Web 頁を表示するための
ソフトウェア

ページが応答しません：処理に時間が掛かる時
の確認メッセージ（継続なら「待機」を選択）

モンテカルロ法（Monte Carlo method）：乱数に基づいて問題の解を得る統計実験

ライブラリ（library）：スクリプトなどから呼び出して使うプログラム集

乱数（random numbers）：無作為復元抽出を繰り返して得られる数

01　手順の指示はスクリプトで

ネット検索やブログ閲読などで日常的に使っているブラウザは「手順」さえ指示してやれば高速＆高機能な計算機としても使えます。

例えば、ある数が素数（＝１とその数以外に約数を持たない正の整数）かどうかを調べたいとします。63 なら 63=3x21 なので「素数ではない（＝合成数）」と簡単に判断できますが、100160063 だったらどうでしょう。奇数なので２では割り切れない、各桁の値の和が 17 なので３でも割り切れない、下二桁＝63 が４の倍数ではないので４でも割り切れない‥という具合で暗算では判断困難、電卓で÷６、÷７‥と試みても割り切れる値にはなかなか辿りつきません。

100160063 は 10007 で割り切れるので、素数ではなく合成数です。でも、割算を延々と繰り返すような作業は面倒なので自分ではしたくありません。そこで以下のように「手順」を書き並べ、そのファイルをブラウザに drag & drop（以下、Ｄ＆Ｄ）すると、今回の処理＝数万回程度の試し割りなら１秒以内で実行してくれます。

```
<script>
 被除数=100160063;除数=2
 while(被除数%除数!=0)除数++
 document.write(被除数+"="+除数+"x"+(被除数/除数))
</script>
```

<script>から</script>までの５行をコピーしてメモ帳などに貼り付け、スクリプトの一例.htm※というファイル名で保存したらそれをブラウザにＤ＆Ｄしてみましょう。図１の結果が表示されるハズです。

※.txt ではなく.htm として保存（文字コードは UTF-8 がオススメです）。

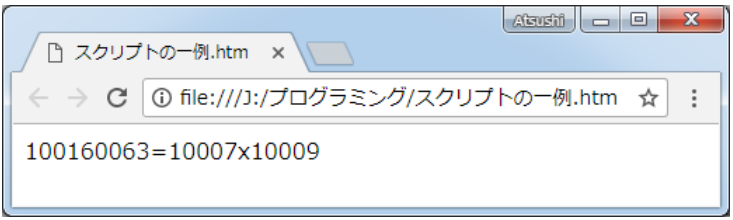


図１　素数判定用スクリプトの一例の実行結果

変数名に漢字（例：被除数、除数）が使えて用語も簡単な英語（例：script、while、document、write‥）、エディタ（メモ帳など）で書けて身近なブラウザで動く（従ってＯＳ不問）といった点で、この「手順」の書き方（JavaScript といいます）は近頃話題のプログラミングを齧（かじ）るにも最適な言語の１つと考えます※。

※ただし、小学校などでの利用が想定されている Scratch や文科省提供の「プログラミン」とは異なり、JavaScript は（見ての通りの）テキスト言語です。

それでは、これから素因数分解、最大公約数と最小公倍数、円周率といった懐かしい（？）計算を例にしながら JavaScript の書き方を順を追って紹介していきます。

補足１：I.E.におけるスクリプトの「実行制限」の回避方法

Windows 8.1 まで標準ブラウザだった Internet Explorer は、スクリプトの htm ファイルをＤ＆Ｄしても「この Web ページは‥」という実行制限メッセージを表示し、許可ボタンをクリックするまで実行しません。

毎回のクリックが面倒なら以下の１行を<script>の上に追加しましょう。

<!-- saved from url=(0008)about:internet -->

02　基本は「基準達成まで処理を反復」

01 でブラウザに与えた以下の５行の指示はどんな「手順」だったのでしょうか？

```
<script>
 被除数=100160063;除数=2
 while(被除数%除数!=0)除数++
 document.write(被除数+"="+除数+"x"+(被除数/除数))
</script>
```

最初の<script>と最後の</script>は「手順」の始まりと終わりの決り文句です（script＝台本・脚本）。

まず、被除数=100160063 と除数=2 で変数「被除数」と「除数」への値の代入を指示します（;は指示の区切り）。

次の while(被除数%除数!=0)除数++は「()内の条件が成り立つ間()に続く処理を繰り返しなさい（＝不成立なら処理はせず次の指示へ）」という指示です（while＝～の間）。条件内の%は剰余（割算の余り）、!=は≠なので、被除数%除数!=0 は「被除数÷除数の余りが０でない間」（＝割り切れるまで）、()に続く処理である除数++は「除数の値を 1 ずつ増やしなさい」という指示です。

そして document.write()は()内をブラウザの画面に表示しなさいという指示です（数値に+"="※などすると全体が文字列になります）。

※文字である等号（"="）を数値の後ろに付ける(+)。

具体値で辿っていくと、被除数に対し除数 2 では余り≠0 なので+1 して 3、3 でも余り≠0 なので 4（中略）、除数 10007 で初めて余り=0

になるので次行に進み、被除数、除数と被除数/除数の３値を文字の"="と"x"とで結合した 100160063=10007x10009 が画面表示されます。

ここで、被除数に素数を指定（例：被除数=10007）して適切なファイル名（例：被除数=10007.htm）で保存の上ブラウザにＤ＆Ｄしてみましょう。被除数が素数なら除数++を繰り返しても剰余≠0 が続き、除数＝被除数になって初めて商=1 で割り切れるため、ブラウザには 10007=10007x1 と表示されます（図２参照）。この結果は、被除数が（１とその数以外に約数を持たない）素数であることを示しています。

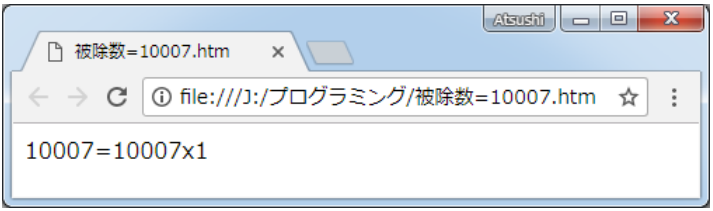


図２　被除数を素数にした場合の一例の実行結果

補足２：数値関連の指示及び本資料での script の書き方※など

①=は（「等しい」ではなく）代入です。従って、例えば除数=除数+1 という表現もＯＫです（意味は除数++と同じく「値を１増やす」）。
②「等しくない」が!=であるのに対し、「等しい」は==です。
③加減乗除（+−*/）に加えて、剰余（%）が使えます。

※本資料では、「正式」な書き方に拘らず、無くても大丈夫な表現は省略しています（例：行末の;）。また「手順」は効率よりも単純さ≒理解し易さを重視します（例：除数++を被除数の平方根で打ち切らない）。

03　素数判定を繰り返せば素因数分解

01 ・ 02 の手順で可能なのは素数か合成数かの判定でした。剰余が最初に 0 になった時点の除数は確かに「被除数の素因数」の１つです。しかし、その時点の商が合成数（素数の積）なら素因数分解としてはまだ途上、商が素数になって初めて完了です。従って、ブラウザに素因数分解をさせるには、被除数を割り切る除数を順に求めて記録しつつ、その商を次の被除数にして商が素数になるまで素数判定を繰り返す「手順」の指示が必要です。

以下のスクリプトはその手順の一例です：

```
<script>
 被除数=223092870;結果=被除数+"="
 while(被除数!=1){
  除数=2
  while(被除数%除数!=0)除数++　//一行なら{}無しでＯＫ
  結果=結果+除数+"x";被除数=被除数/除数
 }
 document.write(結果.slice(0,結果.length-1)) //補足の③参照
</script>
```

<script>から</script>までの９行をコピーしてメモ帳などに貼り付け、素因数分解.htm として保存したらそれをブラウザにＤ＆Ｄしてみましょう。図３のような結果が表示されるハズです。

今回の手順では、まず被除数と共に結果という変数を設定しています。これは素因数分解の過程を「被除数=除数 x 除数 x 除数 x‥」という文字列として記録するためのものです。

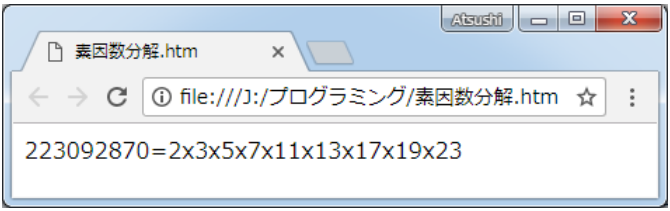


図３　素因数分解用スクリプトの一例の実行結果

次行の while の{から４行下の}までの一組の指示によって、被除数が 1 になるまで以下の手順を繰り返させています：
「剰余が 0 になる（＝割り切れる）まで除数を 2 から 1 ずつ増やす」→「割り切れた時の除数と"x"を結果に追加し、商（＝被除数/除数）を次の被除数に代入する」

商が素数になるとその次の商は 1（02 の例を参照）、従って被除数==1 となるため while の条件は満たされず、その時点の結果の文字列から末尾の一文字（＝過剰な"x"）を削除した部分がブラウザに表示されます（結果.slice(0,結果.length-1)＝結果の文字列の最初（0）～長さ-1 文字まで）※。
※「元値=被除数として除数=2;if(元値!=被除数)結果=結果+"x"」が別案。

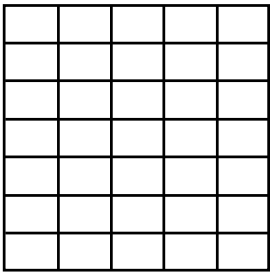
1234321、123454321、12345654321 などの適当な数を被除数にして素因数分解し、結果を電卓などで確認してみましょう。

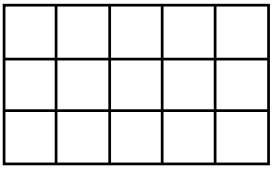
補足３：文字列関連の指示など

①引用符（""）で括った数字、文字、記号は文字列として扱われます。
②内容が数値の変数に文字列（例："="、"x"）を結合すると全体が文字列になります（123+"456"→"123456" vs. 123+456→579）。
③〇.slice(開始位置,終了位置)で〇の内容の一部を抜き出します（〇.length が〇の内容の長さなので、その値-1 で最後の 1 文字が削除されます）。

04　素朴に求める最大公約数と最小公倍数

小学校で、連除法（すだれ算、はしご算‥）などで求めたことを思い出す人も多いことでしょう。実際的な意味まで説明してくれる先生は少ないかもしれませんが、整数ａとｂの最大公約数・最小公倍数は例えば以下の現実場面に該当します：最大公約数は「ａ×ｂcmの長方形の壁に隙間無く敷き詰められる最大の正方形タイルの一辺の長さ」、最小公倍数は「ａ×ｂcmの長方形タイルを同じ向きに隙間無く敷き詰めて作れる最小の正方形の一辺の長さ」。





最大公約数は整数ａとｂの両者を割り切れる最大の整数ですから、素朴に考えれば01 ・ 02の素数判定のスクリプトの条件を「ａもｂも割り切れる値まで小さい方の値-1を繰り返す」にするだけです。また、２つの整数の最小公倍数は、試し割を繰り返さなくても「ａとｂの最小公倍数＝ａ×ｂ／最大公約数」で求められます。

以下のスクリプトはその手順の一例です：

```
<script>
 小さい数=45;大きい数=54;候補=小さい数
 document.write(小さい数+"と"+大きい数+"：")
 while(小さい数%候補!=0||大きい数%候補!=0)候補--
 document.write("最大公約数="+候補+" ")
 document.write("最小公倍数="+小さい数*大きい数/候補)
</script>
```

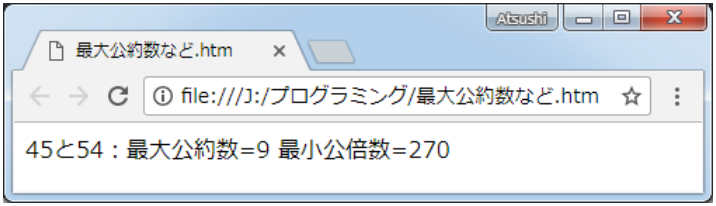


図４　最大公約数などを求めるスクリプトの一例の実行結果

コピペして最大公約数など.htm というファイル名で保存したらブラウザにＤ＆Ｄしてみましょう。図４の結果が表示されるハズです。

今回の手順では、まず「小さい数」の値を「候補」に代入し、その値が小さい数と大きい数を共に割り切れるまで候補の値-1 を繰り返しています。新しい指示の表現は１つ（||）のみで、その意味は「あるいは＝or＝論理和」つまり２つの剰余のいずれかが 0 でない間（＝２つとも 0 になるまで）」という条件指定です。

具体値で辿っていくと、まず候補=45 で小さい数(=45)は割り切れますが大きい数(=54)は割り切れないので候補--で 44、これでは両数とも割り切れないので 43、（中略）9 で初めて両数とも割り切れるので文字列"最大公約数="+9+" "と"最小公倍数="+270 を画面表示※。

※45x54/9=270(補足の②も参照：45=9x5、54=9x6。最小公倍数＝最大公約数（共通要素の 3x3）×独自要素（5x6）=9x30=270)

補足４：「あるいは」と「かつ」など

①あるいは＝or＝論理和は||、かつ＝and＝論理積は&&。

②連除法　3) 45　54
　　3) 15　18　　最大公約数＝3x3=9
　　5　6　　最小公倍数＝3x3x5x6=270

05　用紙（form）を使って値を入力

これまでのスクリプトでは、処理する値（例：被除数、小さい数）の変更には.htm ファイルの書き換えが必要でした。様々な値の処理を連続して行うにはこれでは不便、「欄に入力した値をボタンのクリックで処理」という使い勝手の方がずっと便利です。

そのために必要な入力欄と実行ボタン、ついでに出力欄を追加した素数判定の手順の一例が以下のスクリプトです：

```
<form>
 候補<input type="text" size="12" value="100160063">
 <input type="button" value="実行" onClick="判定()">
 <input type="text" size="30">
</form>
<script>
function 判定(){
 被除数=document.forms[0].elements[0].value;除数=2
 while(被除数%除数!=0)除数++
 結果=被除数+"="+除数+"x"+(被除数/除数)
 document.forms[0].elements[2].value=結果
}
</script>
```

コピペして入力値の素数判定.htm というファイル名で保存したらブラウザにＤ＆Ｄし、実行ボタンをクリックしてみましょう。図５の結果が表示されるハズです。

<form>から</form>までの５行で、ブラウザの起動画面に示される文字列、１行入力欄、ボタン、出力欄を指示しています（form＝用紙）。

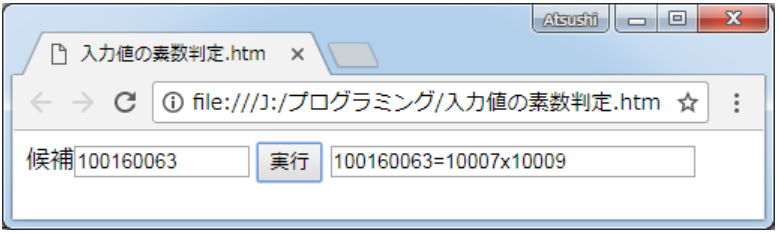


図５　入力値の素数判定を行うスクリプトの一例の実行結果

type="text"で一行欄（input とありますが入出力「両用」）を、type="button"でボタンを用紙に設定します。size は(半角の文字数で)欄の長さの指定、value は text では初期値、button ではボタンの名前の指定で、onClick でボタンがクリックされた時に何をするか（今回は function 判定()の実行）を指定します。

<script></script>では function（関数・機能）が１つ指定され、その処理の内容は（被除数の指定及び結果の出力の部分を除いて）01・02 の素数判定.htm と同一です。被除数に代入される値と結果を代入＝出力する場所で使われている document（中略）value という長々しい表現は document（＝ブラウザ画面）の forms[0]（＝最初の form）の elements[n]（＝ｎ番目の要素）の value（値）という指定で、具体的には被除数には用紙の最初（n=0）の要素である一行欄の値（初期値のままなら 100160063）が代入され、また結果は３番目（n=2）の要素である一行欄に代入＝出力されます（JavaScript では順番は 0 から数え始めます。ちなみに、２番目（n=1）の要素はボタン）。

つまり、このスクリプトでは、計算処理を１つの関数とし、用紙の入力欄に指定された値をその関数で処理して結果を用紙の出力欄に代入する、という流れで仕事が指示されています。

06　本質的な改善とあまり意味のない「改善」

05 のスクリプトの初期値 100160063 の素数判定は一瞬（１秒未満）で終わることでしょう。では、１の桁の 3 を 9（＝100160069）にして実行ボタンをクリックしてみてください。環境にもよりますが、出力欄に 100160069=100160069x1（つまり素数）と出力されるまで若干（十秒程度）の時間が掛かることでしょう。そして、素数の場合の処理時間は桁が１増すごとに 10 倍ずつ増えてゆきます（４桁増えたら約一万倍）！

これまでのスクリプトの「割り切れないなら除数＝被除数になるまで除数+1」という素朴な手順は（実は）大きな無駄を含んでいます。100 を例にするとその約数は順に 2,4,5,10,20,25,50、そして各約数は 2x50,4x25,5x20 のように 10＝√100 の上下で対を成します。この性質を踏まえれば、ｎの約数の検討は√ｎまでで十分、そこまでに約数がなければその先にも約数はありえずｎは素数なのです。

そこで、05 のスクリプトの **while(被除数%除数!=0)除数++**の１行を以下の３行に書き換えます（{から}までが一組の指示）。

```
while(被除数%除数!=0){
  除数++;if(Math.sqrt(被除数)<除数)除数=被除数
}
```

つまり、除数の値が被除数の平方根を超えたら（それ以上の値は無視して）被除数を除数に代入、被除数%除数==0 なので while 終了です。

書き換えた内容を素数判定改.htm というファイル名で保存したらブラウザにＤ＆Ｄし、初期値を 100160069 に変えて実行ボタンをクリックしてみましょう。100160069=100160069x1 という結果が、今度は一瞬（１秒未満）で表示されるハズです。

また、JavaScript には標準機能で処理できる値は 9007199254740992（2 の 53 乗）までという制約があります。そこで不適切な入力値に対しその旨表示する機能を追加したのが以下のスクリプトです：

```
<form>
 候補<input type="text" size="16" value="9007199254740995">
 <input type="button" value="実行" onClick="判定()">
 <input type="text" size="36">
</form>
<script>
function 判定(){
 被除数=document.forms[0].elements[0].value;除数=2
 if(被除数<2||9007199254740992<被除数)alert("値が不適切！")
 while(被除数%除数!=0){
  除数++;if(Math.sqrt(被除数)<除数)除数=被除数
 }
 結果=被除数+"="+除数+"x"+(被除数/除数)
 document.forms[0].elements[2].value=結果
}
</script>
```

2 未満（素数は 2 から）や上限を超える値には注意を表示した上で処理しますが、結果は誤りで「有意味な改善」とは言えません。

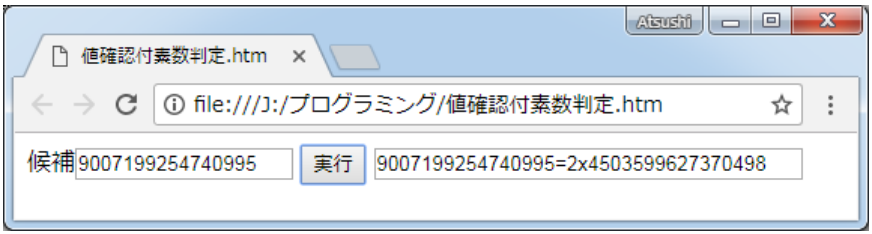


図６　候補に不適切な値を入力した場合の実行結果（誤り）

07　多倍長整数ライブラリで桁数制限無しの素数判定

2 の 53 乗≒九千兆は日常感覚では「無制限」に近いのですが、暗号関連の目的などには不足なため、より大きい数を扱える様々な拡張機能（＝多倍長演算ライブラリ）が作成・公開されています※。

※Google などで JavaScript 用多倍長演算ライブラリを検索してみましょう。

ここでは、上限値の制約の本質的改善を目的として、多倍長整数ライブラリである bigint.js を利用するスクリプトを作成します※。

https://github.com/dankogai/js-math-bigint

※今回のスクリプトの作動には↑上の URL の web 頁からダウンロードした圧縮ファイル（js-math-bigint-master.zip 約６KB）から展開したライブラリ bigint.js（Dan Kogai, 2014）が同一フォルダ内に必要です（図７参照）。

```
<form>候補
 <input type="text" size="32" value=100160063>
 <input type="button" value="実行" onClick="判定()"><br>
 <input type="text" size="44">
</form>
<script src="bigint.js"></script>
<script> //<form>中の<br>=break は改行記号@web 頁
function 判定(){
 被除数=bigint(document.forms[0].elements[0].value)
 除数=bigint(2) //上行の幅不足による改行
 while(被除数.mod(除数)!=0){
  除数++;if(Math.sqrt(被除数)<除数)除数=被除数
 }
 結果=被除数+"="+除数+"x"+(被除数.div(除数))
 document.forms[0].elements[2].value=結果
}
</script>
```

前回のスクリプト（値確認付素数判定.htm）との機能上の相違は①ライブラリ bigint.js の組み込み（</form>の次の行）、②被除数と除数の bigint(値)形式での代入、③剰余の被除数.mod(除数)形式・除算の被除数.div(除数)形式での指定※の３点です（不要な if()alert()行は削除。なお、除数=bigint(2)の改行は行幅不足＠本稿が理由）。
※bigint 用には値を x=bigint()、y=bigint()の形式で代入の上、+は x.add(y)、-は x.sub(y)、*は x.mul(y)、/は x.div(y)、%(剰余)は x.mod(y)と指定。

コピペして素数判定＠多倍長整数.htm として保存したらブラウザにＤ＆Ｄし、候補に 9007199254740995 を入力の上実行ボタンをクリックしてみましょう。今度は（図６とは異なり）図８に示した正しい結果「5 で割り切れる＝素数ではない」が表示されるハズです（１の桁が 5 の整数は当然 5 で割り切れます）。

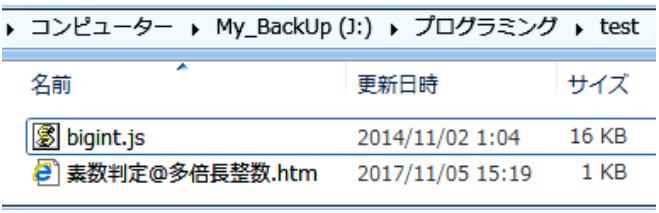


図７　スクリプトと同一フォルダに保存したライブラリ bigint.js

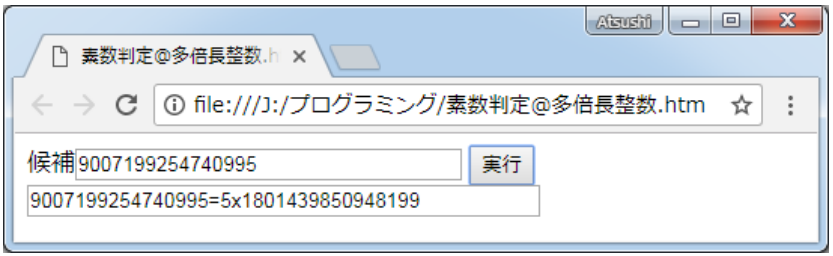


図８　2 の 53 乗より大きい値を正しく判定した結果の一例

08　桁数制限無しの素因数分解@多倍長整数

今度は、03 のスクリプトを桁数制限無しの素因数分解スクリプトに改訂してみます。

```
<form>
 数値<input type="text" size="32" value=9007199254740992>
 <input type="button" value="実行" onClick="分解()"><br>
 <textarea rows="3" cols="45"></textarea>
</form>
<script src="bigint.js"></script>
<script>
function 分解(){
 被除数=bigint(document.forms[0].elements[0].value)
 結果=被除数+"="
 while(被除数!=1){
  除数=bigint(2)
  while(被除数.mod(除数)!=0){
   除数++;if(Math.sqrt(被除数)<除数)除数=被除数
  }
 結果=結果+除数+"x";被除数=被除数.div(除数)
 }
結果=結果.slice(0,結果.length-1) //幅不足のため改行
document.forms[0].elements[2].value=結果
}
</script>
```

<form>の部分では（素数判定ではないので）「候補」を「数値」とし、（出力が長くなる場合を想定して）第３要素を１行の text から複数行の textarea に変更します（rows="3" cols="45"は３行 45 文字）。

ライブラリの組み込みは 07 と同様です。ボタンのクリックで実行される function 分解の部分では、代入を bigint()として行い、剰余を.mod()・除算を.div()の形式で記述し、出力を form の３つ目の要素の elements[2]の value として行うように書き換えました（07 と同様、一行幅の不足による「改行」あり）。

コピペして素因数分解@多倍長整数.htm として多倍長整数ライブラリ bigint.js のあるフォルダに保存したらブラウザにＤ＆Ｄし、数値は初期値の 9007199254740992 のまま[実行]をクリックしてみましょう。9007199254740992=2x2x（中略）x2 と最初の 2 に続いて 52 回の x2 が出力されます（JavaScript の標準機能の上限）。続いて、候補の末尾に 0 を追加して[実行]をクリックしてみましょう。図９の結果が示されるハズです。初期値を 10＝2x5 倍した結果が 2 の 54 乗 x5 と示され、標準機能ではできない処理が多倍長整数ライブラリの組み込みによって正しく行われています。

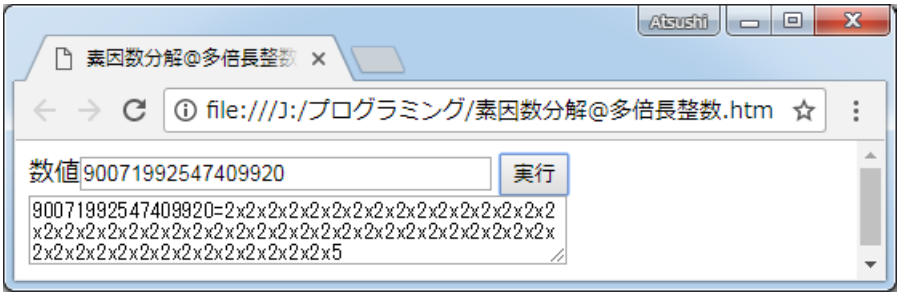


図９　上限数値 x10 を正しく素因数分解した結果の一例

09　桁数制限無しの最大公約数と最小公倍数@多倍長整数

04 の素朴な考え方のスクリプトを、最大公約数と最小公倍数を桁数制限無しに求めるそれに改訂してみます。

```
<form>
 小さい数<input type="text" size="24" value=76543210><br>
 大きい数<input type="text" size="24" value=9876543210>
 <input type="button" value="実行" onClick="計算()"><br>
 <textarea rows="2" cols="40"></textarea>
</form>
<script src="bigint.js"></script>
<script>
function 計算(){
 小さい数=bigint(document.forms[0].elements[0].value)
 大きい数=bigint(document.forms[0].elements[1].value)
 候補=小さい数
 while(小さい数.mod(候補)!=0||大きい数.mod(候補)!=0)候補--
 結="最大公約数="+候補+"¥n" //¥n は改行記号@script
 果="最小公倍数="+小さい数.mul(大きい数).div(候補)
 document.forms[0].elements[3].value=(結+果)
}
</script>
```

<form>の部分では、順に２つのテキスト欄が第１・第２、実行ボタンが第３、そして 40 文字×２行のテキストエリアが第４要素になります。ライブラリを組み込んだ後、bigint()として代入を行い、剰余を.mod()・積算を.mul()・除算を.div()の形式で指定し、改行＝"¥n"を含む文字列の結と果を結合して第４要素のテキストエリアに代入＝出力させます。

コピペして最小公倍数など@多倍長整数.htm として保存したらブラウザにＤ＆Ｄし､数値は初期値の 76543210 と 9876543210 のまま[実行]をクリックしてみましょう。計算終了まで時間がやや掛かります※が、やがて図 10 の結果が示されるハズです。

※「ページが応答しません」などのメッセージに対しては[待機]などで続行。

小さい数・大きい数とも 1 の桁が 0 なので 10 で割り切れることは自明ですが、最大公約数も 10 しかないようです。また、最小公倍数の 75598232099710410※は 17 桁（7 京 5598 兆 2310 億‥）で、多倍長整数ライブラリの組み込みによって標準機能ではできない処理が正しく行われていることが示されています。

※75598232099710410=7654321×987654321×10

ただし、「素朴な考え方」に基づくこのスクリプトでは、計算終了までにかなりの時間がかかります（環境にもよりますが 1 分程度？）。もっと効率のよい「求め方」は無いのでしょうか？

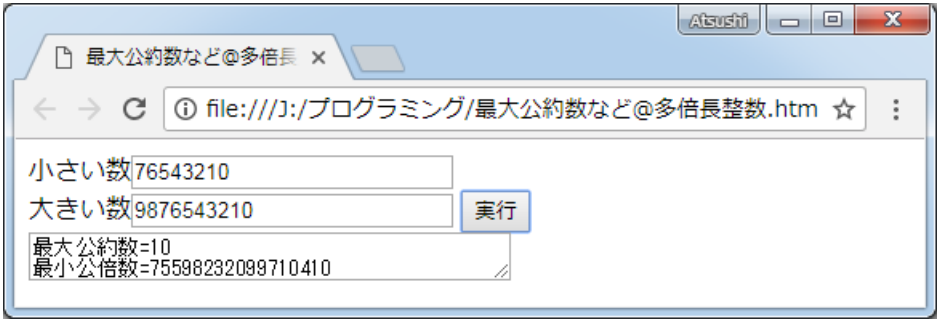


図 10　大き目の数の最大公約数と最小公倍数を求めた結果の一例

10　賢い「求め方」で効率を本質的に改善

最大公約数については、紀元前から知られている「ユークリッドの互除法」という有名な「求め方」（アルゴリズム）があります※。

※広辞苑（第六版）にも以下のように載っています：

> 自然数（または整式）の最大公約数を求める方法の一つ。ａをｂで割った余りをｒとすると、ａとｂの最大公約数は、ｂとｒの最大公約数に等しい。このことを利用して、次にｂをｒで割る、…と次々とこの手続きを行い、余りが０となった時の除数が最大公約数である。

09 のスクリプトを互除法に基づくそれに改訂してみます。

```
<form>
 小さい数<input type="text" size="24" value=76543210><br>
 大きい数<input type="text" size="24" value=9876543210>
 <input type="button" value="実行" onClick="計算()"><br>
 <textarea rows="2" cols="40"></textarea>
</form>
<script src="bigint.js"></script>
<script>
function 計算(){
 小=bigint(document.forms[0].elements[0].value)
 大=bigint(document.forms[0].elements[1].value)
 積=小.mul(大)
 while(大.mod(小)!=0){余=大.mod(小);大=小;小=余}
 結="最大公約数="+小+"¥n" //"¥n"は改行記号
 果="最小公倍数="+積.div(小)
 document.forms[0].elements[3].value=(結+果)
}
</script>
```

変更点は、入力値を変数：小と変数：大に代入する、両値の積を変数：積に残す（→最小公倍数の計算に使用）、その上で（小、大、余の値が入れ替わる）互除法を行う、の３点です。

コピペして互除法@多倍長整数.htm として保存したらブラウザにD&Dし、初期値の 76543210 と 9876543210 のまま[実行]をクリックしてみましょう。今度は、一瞬で図 11 の結果が示されることでしょう。

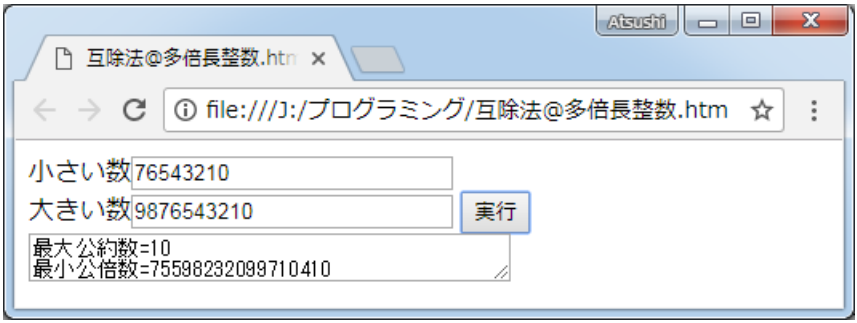
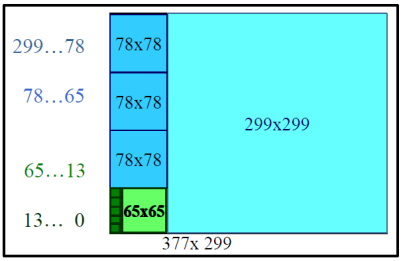
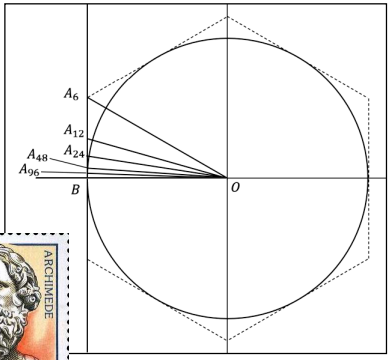


図 11　大き目の数の最大公約数と最小公倍数を互除法で求めた結果の一例



大きい数を小さい数で割り、その余りで今度は小さい数を割り‥という手続きで最大公約数が得られる理由を、377 と 299 を例にした右図で考えてみます。この場合、入れ替え３回目の小さい数 65 は余りの 13 で割り切れます（65/13=5）。65+13 の 78 も 13 で、78x3+65 の 299 も 13 で、そして 299+65+13 の 377 も 13 で割り切れます。つまり互除法とは、大きい数×小さい数の長方形に隙間なく敷き詰められる最大の正方形を「隅への追い込み」で求める手続きなのです。

11　円周率に挑戦！：その１　フーリエ級数@標準機能



　素因数分解、最大公約数といった小・中学校段階の算数・数学の概念を取り上げてスクリプトを紹介してきた本資料の締め括りとして、円周率に挑戦します。

　紀元前３世紀に円周率に数学的に取り組んだアルキメデスは、正六角形を用いた $3<\pi<2\sqrt{3}$ から始めて順に正 12 ・24 ・48 ・96 角形の場合を求め、3 と 10/71<π<3 と 1/7（3.1408‥<π<3.1428‥）という値を示しています。

http://tsujimotter.hatenablog.com/entry/archimedes-circle

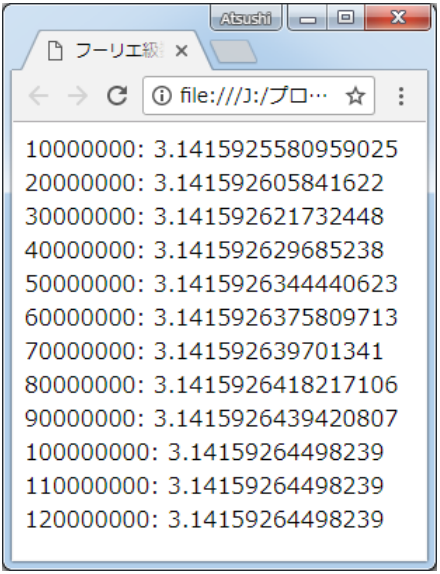
　分数を駆使したこの考え方はスクリプトによる再現向きではないので別の方法を探すと、フーリエ級数※というのがありました。

※$1/n^2$ を n＝1～∞まで足し上げた値（$1/1^2+1/2^2+1/3^2+\cdots$）は円周率$^2/6$

　単純な処理の繰り返しこそコンピュータの得意技、そこでこの級数を指定項数まで途中経過付で求めるスクリプトを書いてみます：

```
<script>
項数=120000000;間隔=10000000;項=0;和=0
while(項<項数){
  項++;和=和+1/項/項
  if(項%間隔==0)document.write(項+": "+Math.sqrt(和*6)+"<br>")
}
</script>
```

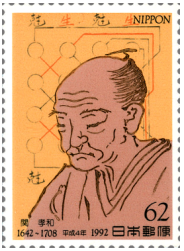
図 12　JavaScript の標準機能でフーリエ級数に基づいて求めた円周率



初期値 0 の変数：和に 1/項^2 を項=1 から指定項数まで足し上げ、項の値が変数 :間隔の倍数になる毎に項とその時の円周率の近似値を画面表示して改行する、という内容です。コピペしてフーリエ級数＠標準機能.htm として保存し、ブラウザにD＆Dすると数秒で図 12 の結果が示されることでしょう。

円周率の値は 3.14159265358979‥ですから、計算結果は 9 千万項までは真の値に近づいています。しかし、1 億項を超えてからは同じ値が示され、06 で言及した「標準機能で処理できる値」の制約により加算ができなくなっているようです。

円周率の値は、日常的にはアルキメデスが 2 千年以上昔に確定した 3.14 で十分でしょうが、東アジアでは 5 世紀に南朝の天文学者祖沖之（そ ちゅうし）が 355/113（＝3.14159292‥）という近似値を示し、日本でも 1681 年に関孝和が 3.14159265359 微弱という値を得ています。21 世紀の我々としては、ブラウザにもう一頑張りさせたいもの、そしてそれに必要なのは小数も扱える多倍長演算ライブラリです。

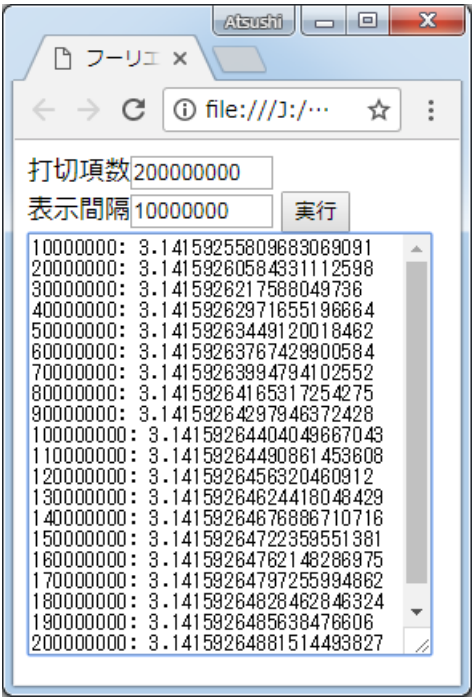


12　その２　フーリエ級数@多倍長小数

任意精度演算ライブラリの big.js※を利用すると共に打切項数と表示間隔の入力を可能にしたスクリプトの一例です：
https://github.com/MikeMcl/big.js
※今回のスクリプトの作動には↑上の URL の web 頁からダウンロードした圧縮ファイル（big.js-master.zip 約 1.2MB）から展開したライブラリ big.js（MikeMcl, 2017 22KB）が同一フォルダ内に必要です（15 の図 17 参照）。

```
<form>
 打切項数<input type="text" size="8" value=20000000><br>
 表示間隔<input type="text" size="8" value=1000000>
 <input type="button" value="実行" onClick="計算()"><br>
 <textarea rows="20" cols="33"></textarea>
</form>
<script src="big.js"></script>
<script>
function 計算(){
 項数=document.forms[0].elements[0].value
 間隔=document.forms[0].elements[1].value
 項=0;和=Big(0);結果="" //big は× Big が○
 while(項<項数){
  項++;和=和.plus(1/項/項)
  if(項%間隔==0){
   結果=結果+項+": "+(和.times(6)).sqrt()+"¥n"
   document.forms[0].elements[3].value=結果
  }
 }
}
</script>
```

〈form〉の第１要素の値を項数に、第２要素の値を間隔に代入し、多倍長小数が必要な和に big.js の形式で初期値 0 を代入し、和に関連する処理を big.js の形式（加算なら和=和.plus(1/項/項)、積算と平方根なら(和.times(6)).sqrt()）で指定します。また項が間隔の倍数になる毎に項とその時の円周率の近似値及び改行記号（"¥n"）を出力用の結果に追記していきます。



コピペしてフーリエ級数@多倍長小数.htm として保存し、ブラウザにＤ＆Ｄしたら、まずは項数の値を 2000000、表示間隔の値を 100000（それぞれの初期値の 1/10）にして実行してみましょう。数秒程度で図 13 のような結果が示されます。ただし、図 13 は項数と間隔を初期値の 10 倍にした結果で、その完了には 1/10 にした場合の 100 倍の時間（10 分程度）が掛かっています。

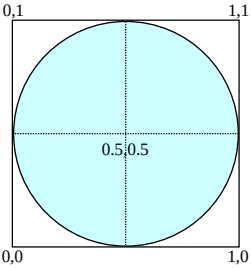
図 13　項数２億、間隔１千万（それぞれ初期値の 10 倍）で実行した結果

多倍長小数演算によって１億項以降も漸近は続き、１億２千万項以降は四捨五入で 3.14159265 になる値が得られています。他方、この方法では処理に時間が掛かるのに加え「そもそもフーリエ級数でなぜ π が求まるのかが不明」という本質的な未達成感が残ります。

13　その３　中学生にも納得できる「面積比と乱数」

フーリエ級数のような「証明には積分が必要で、知らない／忘れた人にはブラックボックス」的な方法ではなく、平均的な中学生にも理解と納得が可能な円周率の求め方に「面積比を乱数で求める方法」※があります。

※「モンテカルロ法」と呼ばれる数値実験（シミュレーション）の一種



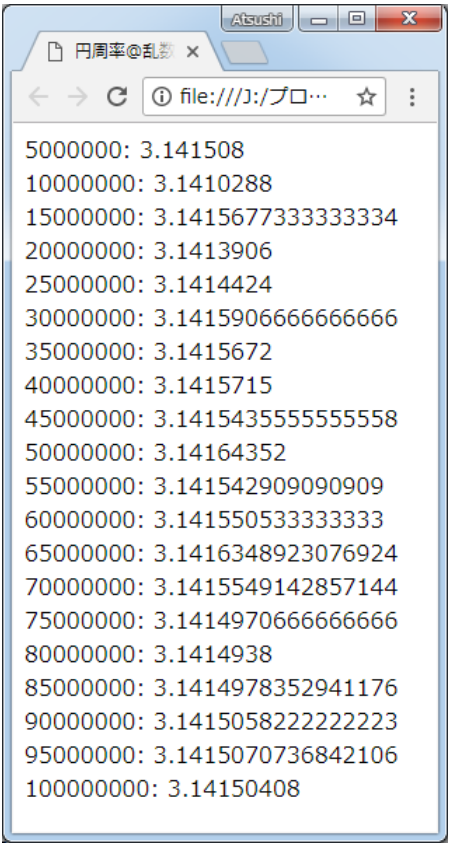
一辺が１の正方形とそれに内接する円を考えます。正方形の面積は当然１、円の面積（πr^2）は$\pi \times 1/2 \times 1/2 = \pi/4$。従って、この円（水色）の面積を何らかの方法で求めれば、面積×４＝π。そして、面積は乱数による多数の点で近似が可能です！

０以上１未満の乱数を発生させる Math.random()で正方形中の無作為の位置に一億個の点を打ち、円の中心から点までの距離※が 0.5 未満なら円内として数え上げ、途中経過を「試行数：その時点での円周率の近似値」の書式で 20 回表示するスクリプトの一例は以下の通りです：

※ピタゴラスの定理：直角三角形の斜辺の長さ＝√(他の辺１^2＋他の辺２^2)

```
<script>
回数=100000000;試行=0;円内=0
while(試行<回数){試行++
 x=Math.random()-0.5;y=Math.random()-0.5
 if(Math.sqrt((x*x)+(y*y))<0.5)円内++
 if(試行%(回数/20)==0){
  document.write(試行+": "+4*円内/試行+"<BR>")
 }
}
</script>
```

図 14　JavaScript の標準機能で乱数から求めた円周率と途中経過



コピペして円周率@乱数.htm として保存し、ブラウザにD＆Dすると数秒で図14のような結果※が示されることでしょう。

※乱数に基づく結果は毎回異なります。

今回の場合、最初の５百万回までの近似値が 3.141508、一億回＝最終値も 3.14150408 で、この程度の試行回数では、点の個数＝試行回数が増えてもπの近似値は収束しません。また、途中経過を辿ると、3.141 までは一貫して得られているものの、それ以降の桁の値は真の値の 3.14159‥を挟んで変動しています。

今回のスクリプトは、中学生にも理解と納得が可能な手順で「円周率の値は 2×2 の正方形に内接する円の面積に等しく、その値は 3.141～2 程度」という内容を示せるものです。しかし、3.141～2 という値の精度は、アレクサンドリアのプトレマイオスが２世紀前半に示した 377/120（≒3.1417）の水準に留まります。無闇に試行回数を増やすことなく、より正確な値を得る方法・工夫は無いものでしょうか？

14　その４　算術幾何平均法@標準機能

円周率を求める方法の１つに算術幾何平均（Arithmetic Geometric Mean）によるアルゴリズムがあります。ガウス(Johann Gauss 1777-1855)の算術幾何平均による完全楕円積分のアルゴリズムとルジャンドル(Adrien-Marie Legendre 1752-1833)の楕円積分に関する関係式に基づくアルゴリズムだそうですが、ボクには何のことやらわかりません。しかし、アルゴリズム（算法＝演算手続き）である以上、たとえ理解できなくても使用はできます。しかも「スーパーコンピュータの（略）πの超高精度計算（略）のアルゴリズムがガウス・ルジャンドル法であることが多い」（寒川 2017）とのことで、期待が持てそうです。
http://www.oishi.info.waseda.ac.jp/~samukawa/GaussLeg2.pdf

以下は、図 15 に示した十進 BASIC のプログラム（寒川 2017 p.3）に基づくスクリプトの一例です：

```
<script>
回数=7;試行=1;a=1;b=1/Math.sqrt(2);s=b*b;t=1
while(試行<回数){
 x=(a+b)/2;y=Math.sqrt(a*b);c=(a-b)/2;t=t*2;s=s+t*c*c
 document.write(試行+": "+((2*a*a)/(1-s))+"<BR>")
 a=x;b=y;試行++
 }
</script>
```

反復計算の回数を 6 とし、試行の値が回数=7 より小さい間（つまり 6 になるまで）の各時点での近似値を試行回数と共に画面表示させます。このスクリプトをコピペして算術幾何平均法@標準機能.htm として保存し、ブラウザにＤ＆Ｄすると一瞬で図 16 の結果が示されることでしょう。

図 15
算術幾何平均によるアルゴリズム（右）
及びそれに基づくプログラム（下）

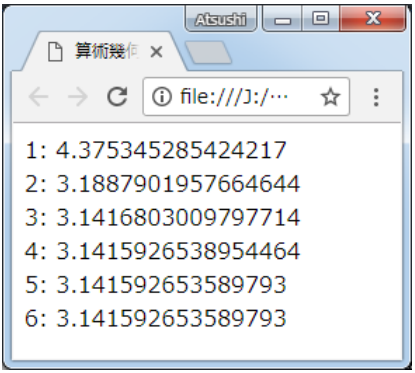
$$a_0 = 1,\ b_0 = c_0 = \frac{1}{\sqrt{2}},\ a_{n+1} = \frac{a_n + b_n}{2},$$

$$b_{n+1} = \sqrt{a_n b_n},\ c_{n+1} = \frac{a_n - b_n}{2}$$

$$n = 0, 1, 2, \ldots \text{ とすれば, } \quad \pi = \frac{\mathrm{M}\left(1, \frac{1}{\sqrt{2}}\right)^2}{1 - \sum_{n=0}^{\infty} 2^n c_n^2}$$

```
LET a=1
LET b=1/SQR(2)
LET s=b^2
LET t=1
FOR i=1 to 5
   LET x=(a+b)/2          ! 算術平均  ( a_i + b_i ) / 2
   LET y=SQR(a*b)         ! 幾何平均  sqrt{ a_i * b_i }
   LET c=(a-b)/2          ! c_i = ( a_i - b_i ) / 2
   LET t=t*2              ! 2^i
   LET s=s+t*c^2          ! s_i = \sum 2^i * c_i^2
   LET a=x
   LET b=y
   LET p=(2*a^2)/(1-s)    ! 算術幾何平均法による \pi の計算値
   PRINT I;"p=";P;
   PRINT p-PI             ! 組込み定数 PI と計算値との差
NEXT i
END
```

図 16　算術幾何平均法で求めた円周率（標準機能）



近似値は第 4 試行（4:）で既に 3.14159265‥になり、フーリエ級数で 1 億項以降まで求めた場合に匹敵する精度が得られています。第 5 試行で小数第 15 位まで正しい値が得られる一方、第 6 試行でも同じ値が示され、11 などでも言及した「標準機能で処理できる値」の限界に到達してしまっているようです。ということは、再び任意精度演算ライブラリの出番です。

補足 5 ：スクリプトを自分で書く時には

用語を別色で、対応 () を強調して表示するエディタ（例：Mery）が便利。

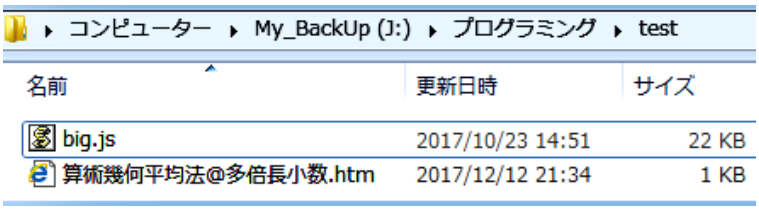
```
<script>↲
›  項数=120000000;間隔=10000000;項=0;和=0↲
›  while(項<項数){↲
›  ›  項++;和=和+1/項/項↲
›  ›  if(項%間隔==0)document.write(項+": "+Math.sqrt(和*6)+"<br>")↲
›  }↲
</script>↲
[EOF]
```

15　その５　算術幾何平均法@多倍長小数

前回のスクリプトを任意精度演算ライブラリ big.js※用に書き直すと以下のようになります：

※12 と同様、このスクリプトの作動には big.js（MikeMcl, 2017 22KB）が同一フォルダ内に必要です。

図 17 スクリプトと同一フォルダに保存したライブラリ big.js




```
<script src="big.js"></script>
<script>
回数=7;試行=0;Big.DP=50
a=Big(1);b=Big(2).sqrt().pow(-1);s=b.pow(2);t=Big(1)
while(試行<回数){
 x=(a.plus(b)).div(2);y=(a.times(b)).sqrt()
 c=(a.minus(b)).div(2);t=t.times(2);s=s.plus(t.times(c.pow(2)))
 π=a.pow(2).times(2).div(Big(1).minus(s))
 document.write(試行+": "+π+"<BR>")
 a=x;b=y;試行++
}
</script>
```

Big.DP=50 で計算する小数位（Decimal Places）を 50 までに指定し※全ての初期値を定数なら Big(n)、平方根なら.sqrt()、べき乗なら.pow(n)で代入します。加算は x.plus(y)、減算は x.minus(y)、乗算は x.times(y)、除算は x.div(y)です。

※指定がない場合は Big.DP=20‥小数第 20 位まで計算します。

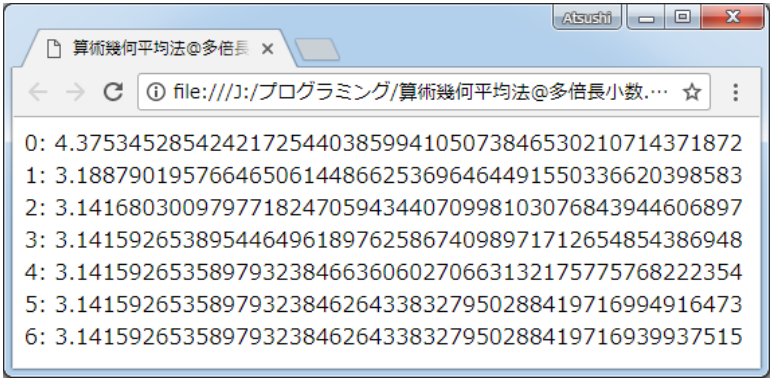


図 18　ライブラリ big.js を用いて小数第 50 位まで求めた円周率

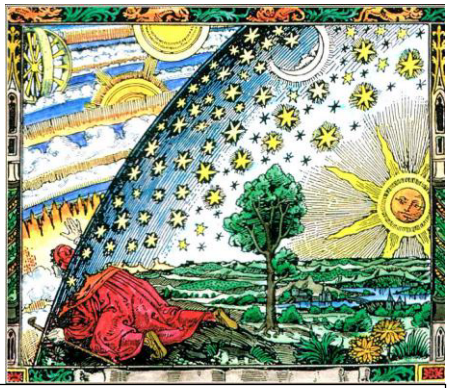
このスクリプトをコピペして算術幾何平均法@多倍長小数.htm として保存し、ブラウザにD＆Dすると図 18 の結果が示されることでしょう。

第３～５試行で小数第 9～42 位まで、そして第６試行では小数第 49 位まで正しい値が得られています（48 位以降の真値は 5105820‥）※。語呂合わせの「才子異国に婿さ、子は苦無く身ふさわし」の２倍以上の桁数に達しており、ライブラリの効果が示されています。

※末尾（今回は小数第 50 位）の値は全計算過程の「丸めの誤差」を含みます。

さて、今回の紹介はここまで。ブラウザ、スクリプト、アルゴリズム、ライブラリといった素敵な道具を創ってくれた先人たちの肩に乗る※ことで、ボクもかなり遠くまで見晴らすことができました。Alan Key（1940-）が Alto を作った 1973 年からまもなく半世紀、Steve Jobs（1955-2011）が Lisa を作った 1983 年からでも 35 年が過ぎました。彼らが目指した「個人の能力を拡張することでその活動を支援する道具」としてのコンピュータとネットの利用をこれからも追求していきたいと思います。

※If I have seen further it is by standing on the shoulders of Giants.
(Isaac Newton 1642-1727)



Peering through the Cosmic Sphere (Nicolas Camille Flammarion, 1888)