Babylonian Square Roots

Jason Nguyen

THE ALGORITHM

- *Divide-and-Average algorithm*
- *Very accurate after a few iterations*
- *Large numbers (~11 digits) can be calculated within accuracy using less than 25 iterations*
  - *The file specification supports a maximum of 10 digits and 6 decimal places, for a maximum number of 9999999999.999999. Yes, this can be calculated in < 25 iterations (on C, it takes 20)*
  - *The COBOL program has a timeout of 1000 iterations, which is somewhat extreme.*

WHAT IS IT?

- In order to take *N*'s square root, we first make an initial guess, R_0, on what it is. In the COBOL program, it takes the original number *N* and divides it by 2, which—when generalized to all numbers—is actually a pretty good guess.
- In order to cycle through a single iteration, we take the initial guess—R_0—and average it with N / R_0—which is the original number divided by this initial guess.
- $R_1 = \frac{R_0 + \frac{N}{R_0}}{2} =$
- $Guess = \frac{PreviousGuess + \frac{OriginalNumber}{PreviousGuess}}{2}$

WHAT IS IT?

- As stated earlier, this algorithm's family is *divide-and-average*—as such—it is nearly logarithmic. One could create a naïve-but-easy implementation of this algorithm (using 100 as the radicand) as follows:

```
OriginalNumber ← 100
Guess ← OriginalNumber / 2

for i ← 0 to 1000 do
    Guess ← (Guess + (OriginalNumber / Guess)) / 2
```

WHAT IS IT?

- This algorithm's naïveté comes from the knowledge that large numbers rarely need more than 25 iterations, so we can forego verifying precision by just iterating 1000 times. That being said—1000 iterations is still fast—and the TAs probably wouldn't notice.

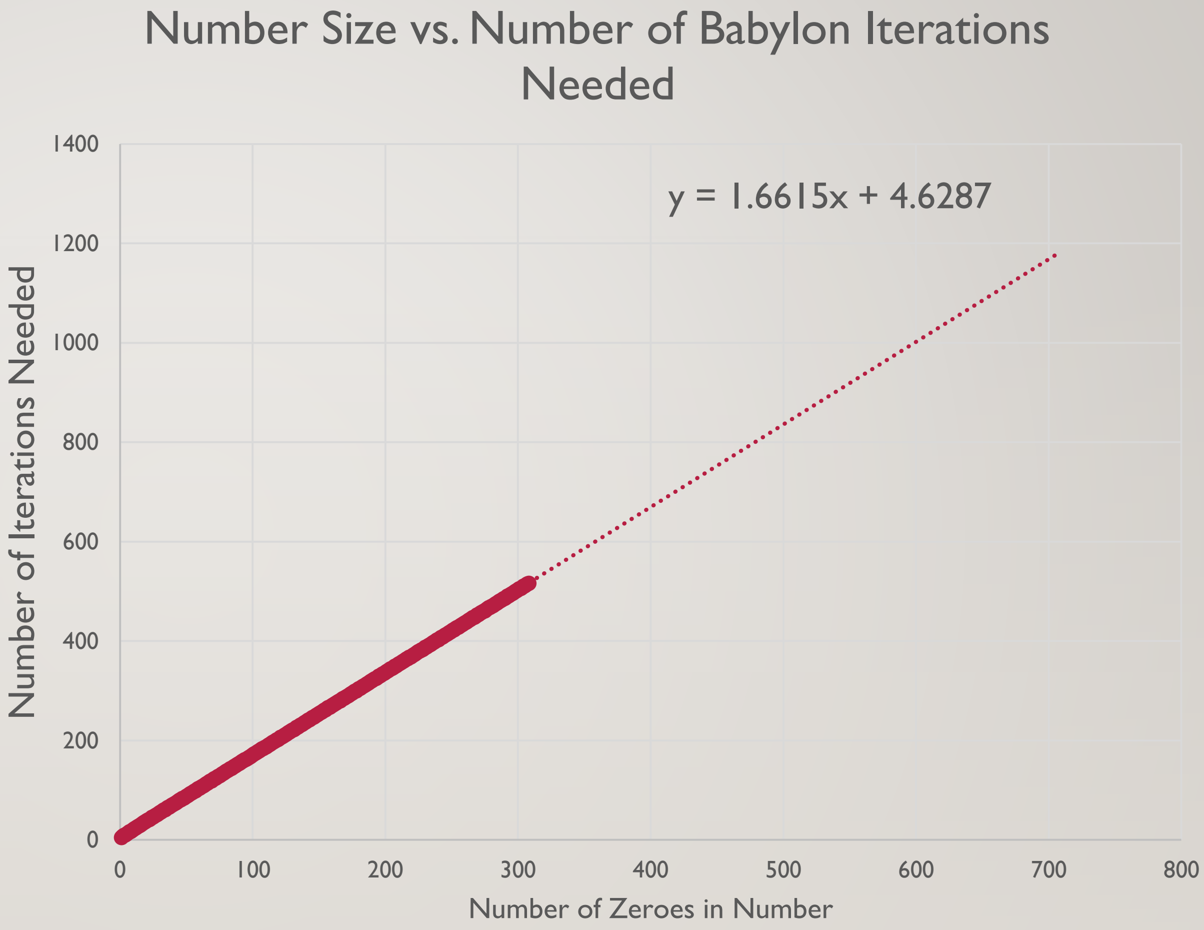
```
OriginalNumber ← 100
Guess ← OriginalNumber / 2

for i ← 0 to 1000 do
    Guess ← (Guess + (OriginalNumber / Guess)) / 2
```

OWO *NOTICES ITERATION COUNT*

You would need a 600-digit number in order to require 1000 iterations (assuming a 0.000001 epsilon value on absolute error ($|R_0 - R_1|$)).

Data based on C program



LOOKING BACK

- Notice in our pseudocode, we don't use `PreviousGuess` as shown in the equation:

$$Guess = \frac{PreviousGuess + \frac{OriginalNumber}{PreviousGuess}}{2}$$

- We don't actually need `PreviousNumber` for the core of the algorithm to work—it *is* useful for if you want to look back a guess though. Up next is the implementation in C, but a non-naïve approach that does use the two variables. You'll see why it's needed.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

This is our precision constant. Every time a new guess is made, it is subtracted from the old guess and compared with the epsilon to check for "doneness". If it exceeds this, we do it again. This continues until the difference between guesses is smaller than it.

That was absolute precision. An alternative—relative precision—is where you check for the same constant, but with the *ratio* of the numbers—not the difference.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

This is our Babylon function.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

It is called here.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

Let us dance…?

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

Like earlier, we have two variables.

- `guess` is our latest guess. As of this point in the code, it is simply half the original number.
- `previous_guess` is the previous guess, but it doesn't exist yet, so it's uninitialized.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

I was going to make a **while** loop, but the comparison check between `guess` and `previous_guess` is only possible after the latter is set.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

First, we set the previous_guess to be the current guess. It isn't logical I know, but give me a sec!

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

Then we do the calculation. Now, our naïve example where we used *only* `guess` still holds here—`previous_guess` is always going to be equal to `guess` in this step. This is actually a disguised `guess = (guess + (N / guess)) / 2.0`. It's just that after this step, we now have a record of our new guess, while having a record of our old guess. After `guess` is set, `previous_guess` still holds the before value.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

We do this to compare the guesses. Now instead of doing it naively 1000 times, we are now checking the absolute error (difference) to see if we need to re-enter the loop or not. The `math.h` library is used for absolute value, and we are comparing it with precision from earlier.

```c
#include <stdio.h>
#include <math.h>

#define precision 0.000001

double babylon(double N)
{
    double guess = N / 2.0;
    double previous_guess;

    do {
        previous_guess = guess;
        guess = (previous_guess + (N / previous_guess))/ 2.0;
    } while (fabs(guess - previous_guess) > precision);

    return guess;
}

int main(void)
{
    printf("%f\n", babylon(9458151235.0));
    return 0;
}
```

Because guess is the most recent iteration's value, we return it.

Stay tuned for part 2