documentation

June 6, 2019

1 Documentation

- `exp.c` is an unoptimized Taylor approximation of e^x without calculation of the optimal number of terms
- `exp_opt.c` is an optimized Taylor approximation of e^x with calculation of the optimal number of terms (see below)
- `ln.c` is a domain-extended Taylor approximation of $ln(x)$
- `neqdst.c` approximates e^x and $ln(x)$ for n equidistant terms in between x_{min} and x_{max} making use of the implementations in `exp_opt` and `ln`

The e^x implementatins are not domain-extended. The MIPS implementations can be found in the according `*.s` files

2 Taylor Approximation I – e^x

2.1 Optimal number of terms for e^x

```
In [2]: import matplotlib.pyplot as plt
        import math
        import numpy as np
```

2.1.1 Upper bounds of n

The nominator and denominator are stored as single-precision floating point units. That means they can reach the maximal value

```
In [3]: single_float_limit = 3.4 * 10 ** 38
```

The denonimator enforces an upper bound of 34 for n.

```
In [4]: math.factorial(34) < single_float_limit

Out[4]: True

In [5]: math.factorial(35) < single_float_limit

Out[5]: False
```

```
In [7]: def max_n(x):
            n = 34
            while(x ** n > single_float_limit):
                n = n - 1
            return n
```

```
In [8]: max_n(0), max_n(13), max_n(14)
```

```
Out[8]: (34, 34, 33)
```

The enumerator bounds n only on $x > 13$. Determine for each x the maximal number of n such that $x^n < 3.4 * 10^{38}$

2.1.2 Upper bounds of x

```
In [9]: math.exp(100)
```

```
Out[9]: 2.6881171418161356e+43
```

```
In [10]: x = 100
         while math.exp(x) > single_float_limit:
             x = x - 1

         x , math.exp(x)
```

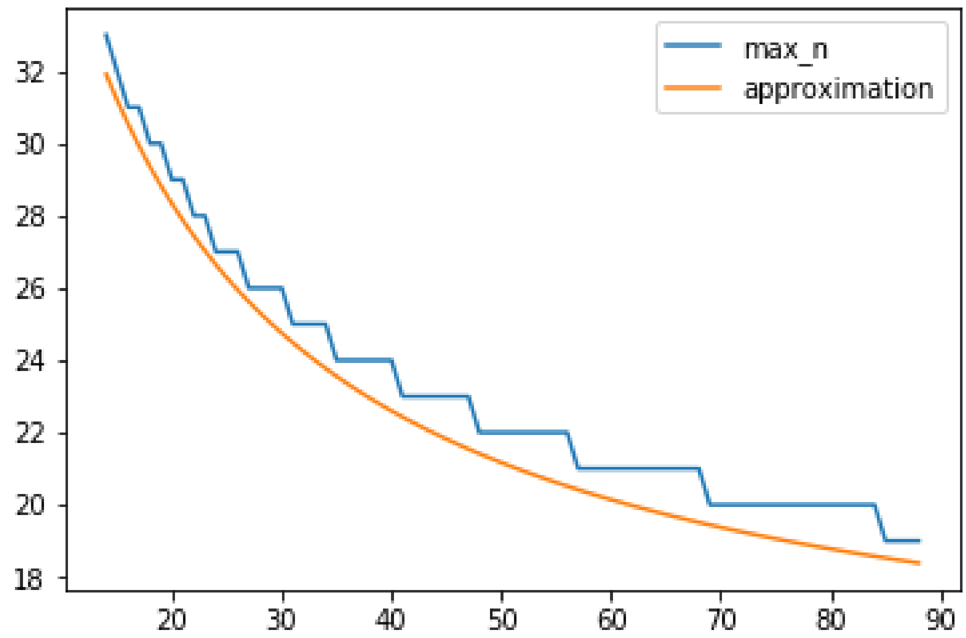
```
Out[10]: (88, 1.6516362549940018e+38)
```

2.1.3 Approximation

```
In [20]: def approx(x):
             return 430/(x + 10) + 14

         t = range(14, 88 + 1)
         plt.plot(t, [max_n(x) for x in t], label='max_n')
         plt.plot(t, [approx(x) for x in t], label='approximation')
         plt.legend()
```

```
Out[20]: <matplotlib.legend.Legend at 0x1108f3518>
```



Test: the approximation should always be lower equal the actual value

```
In [22]: correct = True
         for x in range(14, 88 + 1):
             if approx(x) > max_n(x):
                 correct = False
         correct
```

```
Out[22]: True
```

Therefore the optimal number of terms can be approximated by the following: $

$$|x| < 14 \rightarrow n = 34|x| >= 14 \rightarrow n = \frac{430}{x+10} + 14 \tag{1}$$

$

```
In [29]: def e_terms(x):
             if abs(x) < 14:
                 return 34
             return math.floor( 400 / (x + 10) + 14 )
```

```
In [32]: e_terms(1), e_terms(13), e_terms(14), e_terms(45), e_terms(88)
```

```
Out[32]: (34, 34, 30, 21, 18)
```

2.2 Calculating the factorial with floats instead of integers

As 32-bit integers have a smaller maximal value than single-precision floating point. This puts a lower boundary on n.

```
In [214]: max_int = 2147483647
```

```
In [216]: n = 1
          while(math.factorial(n) < max_int):
              n = n + 1
          n
```

```
Out[216]: 13
```

Maximum of n: 12

That's why calculating the factorial in a floating point register leads to a much better accuracy.

2.3 Worst Case Error

With above determined approximation the approximation still isn't perfect. Without the transformation of $e^x = e^{x-k \cdot ln(2)}$ the maximum error should appear on the highest x within the valid range of $x \in [-88; 88]$

```
In [59]: x = 88
```

```
In [60]: result = 1
         nominator = 1
         denominator = 1

         for i in range(1, e_terms(x) + 1):
             nominator = nominator * x
             denominator = denominator * i
             result += nominator / denominator

         abs(math.exp(x) - result)
```

```
Out[60]: 1.6516362549940018e+38
```

3 Taylor Approximation II – $ln(x)$

3.1 Domain of $ln(x)$

With the transformation of $ln(x) = ln(a) + bln(2)$ where $a \in [0, 2]$ and $b = \log_2 \frac{x}{a}$ the valid the domain of x is extended to all values, for which b fits in a floating point register. Because b is a log function this practically never happens there exists no real upper bound for x.

$\rightarrow x \in (0, \infty[$

The optimal number of terms for the ln is therefore also the highest possible number to be stored. But for a good approximation 1000 terms are definitely sufficient.