On Error Free Transformations and Applications to Polynomial Equations

Aidan C O'Neill

MathFest

August 3, 2018

IEEE 754

- Storage

Sign	Exponent	Mantissa

- Exceptions
  - 0
  - Overflow
  - Underflow
  - Not a Number
- Rounding
  - To the Nearest Even

Example of Floating Point Conversion

1 10000011 01101100000000000000000

Example of Floating Point Conversion

1 10000011 01101100000000000000000

$-1.011011 * 2^{131-127}$

Example of Floating Point Conversion

1 10000011 01101100000000000000000

$-1.011011 * 2^4$

$-(2^4 + 2^2 + 2^1 + 2^{-1} + 2^{-2})$

$-(16 + 4 + 2 + \frac{1}{2} + \frac{1}{4})$

$-(22\frac{3}{4})$

Naïve Operations

$$[10^{16}, 1.0, 1.0, -1 * 10^{16}]$$

Naïve Operations

$$[10^{16}, 1.0, 1.0, -1 * 10^{16}]$$

$$10000000000000001$$

Naïve Operations

$$[10^{16}, 1.0, 1.0, -1 * 10^{16}]$$

$$[10^{16}, -1 * 10^{16}, 1.0, 1.0]$$

Error Free Transformations

Knuth's Algorithm [4]:

```
twoSum(double a, double b){
        double first = a + b
        double temp = ans.a - a
        double second = (a - (first - temp)) + (b - temp)
        return (first, second)
}
```

Error Free Transformations

Veltkamp-Dekker Algorithm:
Note:

$$k = \lceil \frac{p}{2} \rceil$$

$$m = 2^k + 1$$

```
split(double a){
        double temp = m * a;
        double x = temp - (temp - a);
        double y = a - x;
        return (x, y);
}
```

Error Free Transformations

Veltkamp-Dekker Algorithm [1]:

```
twoProduct(double a, double b){
        double x = a * b;
        doubles (c, d) = split(a);
        doubles (e, f) = split(b);
        double y  = d*f - (((x - c*e) - d*e) - c*f);
        return (x, y);
}
```

Horner's Method

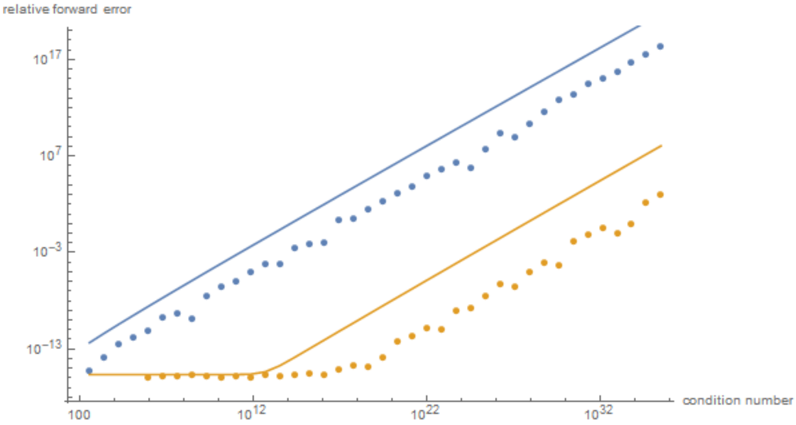
```
double standardHorner(double* poly, int deg, double x){
        double res =  poly[deg];
        for(int i = deg-1; i>=0; i--){
                res = res*x + poly[i];
        }
        return res;
}
```

Compensated Horner's Method

```
double compensatedHorner(double* poly, int deg, double x){
        double res =  poly[deg];
        double c = 0;
        two_double pres, sres;
        for(int i = deg-1; i>=0; i--){
                pres = twoProduct(res, x)
                sres = twoSum(pres.a, poly[i];
                res = sres.a;
                c = x*c + (pres.b + sres.b);
        }
        return res+c;
}
```

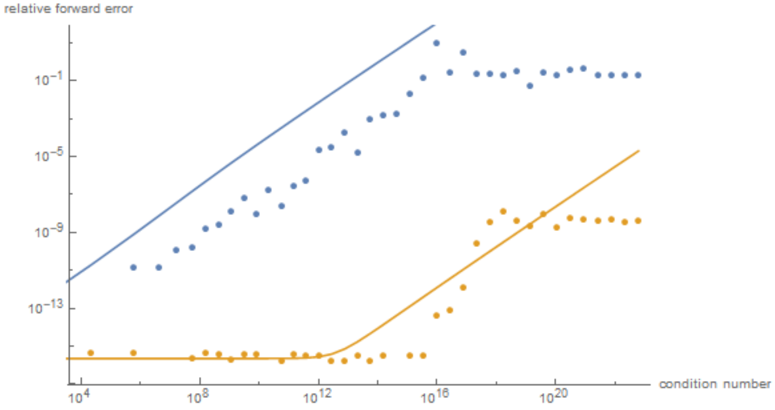
Results of Compensated Horner

[3]



Results of Compensated Newton's Method

[2]



Our Results

Poly No.	Description	Deg.	Roots
1	Wilkinson polynomial	10	$1, \ldots, 10$
2	Wilkinson polynomial	15	$1, \ldots, 15$
3	Wilkinson polynomial	20	$1, \ldots, 20$
4	scale and shifted Wilkinson polynomial	20	$-2.1, -1.9, \ldots, 1.7$
5	reverse Wilkinson polynomial	10	$1, 1/2, \ldots, 1/10$
6	reverse Wilkinson polynomial	15	$1, 1/2, \ldots, 1/15$
7	reverse Wilkinson polynomial	20	$1, 1/2, \ldots, 1/20$
8	prescribed roots of varying scale	20	$2^{-10}, 2^{-9}, \ldots, 2^9$
9	prescribed roots of varying scale -3	20	$2^{-10} - 3, 2^{-9} - 3, \ldots, 2^9 - 3$
10	Chebyshev polynomial	20	$\cos(\frac{2j-1}{40}\pi)$
11	$z^{20} + z^{19} + \cdots + z + 1$	20	$e^{i\frac{2j}{21}\pi}$
12	C. Traverso	24	known
13	Mandelbrot	31	known
14	Mandelbrot	63	known

Our Results

Poly No.	FPML	Polzeros	AMVW
1	$3.82 \cdot 10^{-11}$	$3.84 \cdot 10^{-10}$	$1.04 \cdot 10^{-10}$
2	$4.14 \cdot 10^{-6}$	$3.63 \cdot 10^{-6}$	$1.75 \cdot 10^{-6}$
3	$3.69 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	1.05
4	$8.23 \cdot 10^{-13}$	$1.89 \cdot 10^{-13}$	$5.95 \cdot 10^{-13}$
5	$4.66 \cdot 10^{-10}$	$1.07 \cdot 10^{-10}$	$6.61 \cdot 10^{-7}$
6	$1.51 \cdot 10^{-6}$	$2.85 \cdot 10^{-7}$	0.25
7	$7.87 \cdot 10^{-2}$	$1.53 \cdot 10^{-2}$	1.28
8	$8.12 \cdot 10^{-15}$	$2.21 \cdot 10^{-15}$	$5.95 \cdot 10^{-2}$
9	0.76	$2.2 \cdot 10^{-2}$	$3.72 \cdot 10^{-2}$
10	$1.36 \cdot 10^{-10}$	$3.72 \cdot 10^{-11}$	$9.8 \cdot 10^{-11}$
11	$1.01 \cdot 10^{-15}$	$2.65 \cdot 10^{-16}$	$6.32 \cdot 10^{-16}$
12	$3.93 \cdot 10^{-8}$	$3.83 \cdot 10^{-8}$	9.89
13	$5.35 \cdot 10^{-8}$	$3.94 \cdot 10^{-7}$	$4.38 \cdot 10^{-8}$
14	0.14	0.18	0.15

Poly No.	FPML Comp
1	$6.78 \cdot 10^{-17}$
2	$4.89 \cdot 10^{-17}$
3	$1.2 \cdot 10^{-5}$
4	$9.64 \cdot 10^{-14}$
5	$3.28 \cdot 10^{-11}$
6	$1.54 \cdot 10^{-7}$
7	$3.09 \cdot 10^{-5}$
8	$1.3 \cdot 10^{-15}$
9	$9.38 \cdot 10^{-3}$
10	$6.82 \cdot 10^{-15}$
11	$2.23 \cdot 10^{-16}$
12	$7.86 \cdot 10^{-9}$
13	$1.04 \cdot 10^{-8}$
14	$1.88 \cdot 10^{-2}$

References I

- T. J. Dekker, *A floating-point technique for extending the available precision*, Numer. Math., 18 (1971), pp. 224–242.
- S. Graillat, *Accurate simple zeros of polynomials in floating point arithmetic*, Comput. Math. Appl., 56 (2008), pp. 1114–1120.
- S. Graillat, N. Louvet, and P. Langlois, *Compensated horner scheme*, tech. rep., Université de Perpignan Via Domitia, 2005.
- D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2, Addison-Wesley, Reading, Massachusetts, 1969.