General Test Params:

n=1000000000 Tested 3 Separate times in the following order – Warmup – With Own Clock – With Cristal Clock

The test inputs stayed the same and the output array is the same for all tests. So the outputs get overwritten each time.

Checking for the precision

The search for the largest element is really expensive. Maybe a quicksort algorithm would be kind of usefull but then the results must be remapped what could also take really long.

One Execution

TIME OC: 867.28815699999996

TIME CC: 856.35544410073157

```
for (size_t i = 1; i < n; ++i) { if (input[i] > max_element) max_element = input[i];
res[0] = (double)max_element;
```

Two Executions

TIME OC: 1429.213712

TIME CC: 1455.0524009631931

```
for (size_t i = 1; i < n; ++i) { if (input[i] > max_element) max_element = input[i];
res[0] = (double)max_element;
for (size_t i = 1; i < n; ++i) { if (input[i] > max_element) max_element = input[i-1
res[1] = (double)max_element;
```

Plane Loop

Function call (had been changed)

TIME OC: 1336.8112080000001

TIME CC: 1323.1885089965103

Function call without loop (had been changed)

TIME OC: 0.000137

TIME CC: 1.2864116937051667e−05

Range Reduction

One execution

TIME OC: 799.05626700000005

TIME CC: 761.56983049265386

```
SDOUBLE x   = LOAD_DOUBLE_VEC(&input[i]);

const SDOUBLE ranges_away = MUL_DOUBLE_S(x, one_over_2_pi);
const SDOUBLE num_ranges_away = FLOOR_DOUBLE_S(ranges_away);
const SDOUBLE range_multiple = MUL_DOUBLE_S(num_ranges_away, two_pi);
const SDOUBLE in_outer_range = SUB_DOUBLE_S(x, range_multiple);

SIMD_TO_DOUBLE_VEC(&res[i], in_outer_range);
```

Two executions

TIME OC: 794.39666099999999

TIME CC: 788.55947432130688

```
SDOUBLE x    = LOAD_DOUBLE_VEC(&input[i]);

const SDOUBLE ranges_away = MUL_DOUBLE_S(x, one_over_2_pi);
const SDOUBLE num_ranges_away = FLOOR_DOUBLE_S(ranges_away);
const SDOUBLE range_multiple = MUL_DOUBLE_S(num_ranges_away, two_pi);
const SDOUBLE in_outer_range = SUB_DOUBLE_S(x, range_multiple);

const SDOUBLE ranges_away1 = MUL_DOUBLE_S(in_outer_range, one_over_2_pi);
const SDOUBLE num_ranges_away1 = FLOOR_DOUBLE_S(ranges_away1);
const SDOUBLE range_multiple1 = MUL_DOUBLE_S(num_ranges_away1, two_pi);
const SDOUBLE in_outer_range1 = SUB_DOUBLE_S(in_outer_range, range_multiple1);

SIMD_TO_DOUBLE_VEC(&res[i], in_outer_range);
```

Taylor Polynom Plot

The results are for different degrees shown in the plot at ./plots/taylor_degree_test.png.

As a little remark, i do not know why the *two executions are faster than the one execution*.

Graph

One Execution

```
SDOUBLE x    = LOAD_DOUBLE_VEC(&input[i]);
const SDOUBLE centered_values = SUB_DOUBLE_S(x, center_point);
SDOUBLE result = LOAD_DOUBLE(TAYLOR_COEFF_SIN[taylor_last_coeff]);

for (int j = taylor_loop_iteration; j >= 0; --j) {
    SDOUBLE coeff = LOAD_DOUBLE(TAYLOR_COEFF_SIN[j]);
    result = MUL_DOUBLE_S(result, centered_values);
    result = ADD_DOUBLE_S(result, coeff);
}

SIMD_TO_DOUBLE_VEC(&res[i], result);
```

Two Executions

```
SDOUBLE x    = LOAD_DOUBLE_VEC(&input[i]);
const SDOUBLE centered_values = SUB_DOUBLE_S(x, center_point);
SDOUBLE result = LOAD_DOUBLE(TAYLOR_COEFF_SIN[taylor_last_coeff]);

for (int j = taylor_loop_iteration; j >= 0; --j) {
    SDOUBLE coeff = LOAD_DOUBLE(TAYLOR_COEFF_SIN[j]);
    result = MUL_DOUBLE_S(result, centered_values);
    result = ADD_DOUBLE_S(result, coeff);
}

const SDOUBLE centered_values1 = SUB_DOUBLE_S(x, center_point);
SDOUBLE result1 = LOAD_DOUBLE(TAYLOR_COEFF_SIN[taylor_last_coeff]);

for (int j = taylor_loop_iteration; j >= 0; --j) {
    SDOUBLE coeff = LOAD_DOUBLE(TAYLOR_COEFF_SIN[j]);
    result1 = MUL_DOUBLE_S(result1, centered_values1);
    result = ADD_DOUBLE_S(result1, coeff);
}

SIMD_TO_DOUBLE_VEC(&res[i], result);
```

Quadrant Evaluation setup

One Execution

- TIME OC: 818.80264299999999
- TIME CC: 795.44322714426596

```
SDOUBLE x   = LOAD_DOUBLE_VEC(&input[i]);

const SDOUBLE multiplied_quadrants = MUL_DOUBLE_S(x, quadrant_multiplier);
const SDOUBLE quadrant_evaluation = ADD_DOUBLE_S(multiplied_quadrants, addition_ve
const SDOUBLE quadrant_evaluated_result = MUL_DOUBLE_S(x, quadrant_evaluation);

SIMD_TO_DOUBLE_VEC(&res[i], quadrant_evaluated_result);
```

Two Executions

- TIME OC: 770.94519400000001
- TIME CC: 775.73264921769305

```
SDOUBLE x   = LOAD_DOUBLE_VEC(&input[i]);

const SDOUBLE multiplied_quadrants = MUL_DOUBLE_S(x, quadrant_multiplier);
const SDOUBLE quadrant_evaluation = ADD_DOUBLE_S(multiplied_quadrants, addition_ve
const SDOUBLE quadrant_evaluated_result = MUL_DOUBLE_S(x, quadrant_evaluation);

const SDOUBLE multiplied_quadrants1 = MUL_DOUBLE_S(x, quadrant_evaluated_result);
const SDOUBLE quadrant_evaluation1 = ADD_DOUBLE_S(multiplied_quadrants1, addition_
const SDOUBLE quadrant_evaluated_result1 = MUL_DOUBLE_S(quadrant_evaluated_result,

SIMD_TO_DOUBLE_VEC(&res[i], quadrant_evaluated_result1);
```