```
#ifdef RCSID
static const char rcsid[] =
  "$Id: roots.c,v 1.1 2005/07/04 01:47:29 cperciva Exp $";
#endif

#include <assert.h>
#include <stdint.h>

#include "roots.h"
```

1 Correctly rounded double-precision constants

A value x can be converted into a correctly rounded double precision floating-point constant via the following MAPLE code:

```
r := proc(x)
        global Digits;
        local y, sgn, pow, mant;

        if x = 0 then
                RETURN("0.0");
        fi;

        Digits := 30;
        sgn := sign(evalf(x));
        pow := floor(log[2.](x * sgn));
        y := evalf(x * sgn * 2^(52 - pow));

        if abs(y - round(y)) > 0.49999 then
                RETURN("CANNOT ROUND, NEED MORE DIGITS");
        fi;

        if sgn > 0 then
                sgn := "0x1.";
        else
                sgn := "-0x1.";
        fi;

        cat(sgn, substring(convert(round(y), hex), 2..-1), "p", pow);
end;
```

Values which are more than 0.49999ulp away from from the nearest representable double precision floating-point value will output an error message, in which case the parameters 30 and 0.49999 should be increased.

2 Values of $\exp(i\pi/2^n) - 1$

The table *omc_s* is generated by the following MAPLE code:

```
for n from 7 to 29 do
        printf("%s, %s,\n", r(cos(2 * Pi / 2^n) - 1), r(sin(2 * Pi / 2^n)));
od;
```

and, taking $w_{k+7} = \text{omc\_s}_{2k} + i\text{omc\_s}_{2k+1}$, satisfies

$$w_k \approx \exp(2i\pi/2^k) - 1$$

for $7 \le k \le 29$.

```
static double omc_s[] = {
        -0x1.3BC390D250439p-10, 0x1.91F65F10DD814p-5,
        -0x1.3BCFBD9979A27p-12, 0x1.92155F7A3667Ep-6,
        -0x1.3BD2C8DA49511p-14, 0x1.921D1FCDEC784p-7,
        -0x1.3BD38BAB6D94Cp-16, 0x1.921F0FE670071p-8,
        -0x1.3BD3BC5FC5AB4p-18, 0x1.921F8BECCA4BAp-9,
        -0x1.3BD3C88CDCA13p-20, 0x1.921FAAEE6472Ep-10,
        -0x1.3BD3CB98226DCp-22, 0x1.921FB2AECB360p-11,
        -0x1.3BD3CC5AF3E1Dp-24, 0x1.921FB49EE4EA6p-12,
        -0x1.3BD3CC8BA83EEp-26, 0x1.921FB51AEB57Cp-13,
        -0x1.3BD3CC97D5562p-28, 0x1.921FB539ECF31p-14,
        -0x1.3BD3CC9AE09BFp-30, 0x1.921FB541AD59Ep-15,
        -0x1.3BD3CC9BA36D7p-32, 0x1.921FB5439D73Ap-16,
        -0x1.3BD3CC9BD421Cp-34, 0x1.921FB544197A1p-17,
        -0x1.3BD3CC9BE04EEp-36, 0x1.921FB544387BAp-18,
        -0x1.3BD3CC9BE35A2p-38, 0x1.921FB544403C1p-19,
        -0x1.3BD3CC9BE41CFp-40, 0x1.921FB544422C2p-20,
        -0x1.3BD3CC9BE44DBp-42, 0x1.921FB54442A83p-21,
        -0x1.3BD3CC9BE459Dp-44, 0x1.921FB54442C73p-22,
        -0x1.3BD3CC9BE45CEp-46, 0x1.921FB54442CEFp-23,
        -0x1.3BD3CC9BE45DAp-48, 0x1.921FB54442D0Ep-24,
        -0x1.3BD3CC9BE45DDp-50, 0x1.921FB54442D16p-25,
        -0x1.3BD3CC9BE45DEp-52, 0x1.921FB54442D18p-26,
        -0x1.3BD3CC9BE45DEp-54, 0x1.921FB54442D18p-27,
};
```

3 Values of $\exp(2ki\pi/2^6)$

The table *c_s* was constructed by the following MAPLE code:

```
for k from 0 to 8 do
        printf("%s, %s,\n", r(cos(2 * Pi * k / 2^6)),
            r(sin(2 * Pi * k / 2^6)));
od;
```

and, taking $w_k = \mathrm{c\_s}_{2k} + i\mathrm{c\_s}_{2k+1}$, satisfies

$$w_k \approx \exp(2ki\pi/2^6)$$

for $0 \le k \le 8$.

```
static double c_s[] = {
        0x1.0000000000000p0, 0.0,
        0x1.FD88DA3D12526p-1, 0x1.917A6BC29B42Cp-4,
        0x1.F6297CFF75CB0p-1, 0x1.8F8B83C69A60Bp-3,
        0x1.E9F4156C62DDAp-1, 0x1.294062ED59F06p-2,
        0x1.D906BCF328D46p-1, 0x1.87DE2A6AEA963p-2,
        0x1.C38B2F180BDB1p-1, 0x1.E2B5D3806F63Bp-2,
        0x1.A9B66290EA1A3p-1, 0x1.1C73B39AE68C8p-1,
        0x1.8BC806B151741p-1, 0x1.44CF325091DD6p-1,
        0x1.6A09E667F3BCDp-1, 0x1.6A09E667F3BCDp-1,
};
```

4 Computation of $\exp(2ki\pi/2^{29})$

We make use of the following three recurrences:

1. $\exp(i(x + \pi/4)) = i \cdot \text{conjugate}(\exp(i(\pi/4 - x)))$.
2. For k not a multiple of 2^{n-6}, $0 \le k < 2^{n-3}$, $0 \le q < 2^3$, $0 < r < 2^{n-6}$, $k = q2^{n-6} + r$,
$$\exp(2\pi ik/2^n) = \exp(2\pi iq/2^6) + \exp(2\pi iq/2^6) \cdot (\exp(2\pi ir/2^n) - 1)$$
3. For $2^{n-m} < k < 2^{n-m+1} \le 2^{n-6}$, $k = 2^{n-m} + r$,
$$\exp(2\pi ik/2^n) - 1 = x_0 + x_1 + x_0x_1$$
where $x_0 = \exp(2\pi i/2^m) - 1$, $x_1 = \exp(2\pi ir/2^n) - 1$.

and note that, together with tables of $\exp(2\pi ik/2^6)$ for $0 \le k \le 2^3$ and $\exp(2\pi i/2^m) - 1$ for $7 \le m \le 29$, these permit the computation of $\exp(2\pi ik/2^{29})$ for $0 \le k < 2^{27}$.

The function *expm1_tbl*(LUT, n, m) computes the complex values $\exp(2\pi ik/2^{m+7}) - 1$ for $0 \le k < 2^n$ and writes them into LUT. The inputs must satisfy $0 \le m \le 22$, $0 \le n \le m+1$, and LUT must have space to store 2^{n+1} doubles (i.e., 2^{n+4} bytes).

```
static void expm1_tbl(double * LUT, int n, int m)
{
        double x0r, x0i, x1r, x1i;
        uint32_t i, N;

        if (n == 0) {
                LUT[0] = 0.0;
                LUT[1] = 0.0;
        } else {
                expm1_tbl(LUT, n - 1, m);

                x0r = omc_s[2 * (m + 1 - n)];
                x0i = omc_s[2 * (m + 1 - n) + 1];

                N = 1 << (n - 1);
                for (i = 0; i < N; i++) {
                        x1r = LUT[2 * i];
                        x1i = LUT[2 * i + 1];

                        LUT[2 * (i + N)] = x0r + (x1r +
                            (x0r * x1r - x0i * x1i));
                        LUT[2 * (i + N) + 1] = x0i + (x1i +
                            (x0r * x1i + x0i * x1r));
                }
        }
}
```

Lemma 1. *Given LUT as produced by* expm1_tbl*(LUT, n, m) and making the assignment* $w_k = LUT_{2k} + iLUT_{2k+1}$*,*

$$\left|w_k + 1 - \exp(2\pi ik/2^{m+7})\right| < 12.75\epsilon \cdot 2^n/2^{m+7} + \sum_{j=m+1-n}^{m} \left|W_j - \widehat{W}_j\right|$$

for all $0 \le k < 2^n$*, where* $W_j = \exp(2\pi i/2^{j+7}) - 1$ *and* $\widehat{W}_j$ *is the value of* W_j *as stored in* omc_s*.*

Further, if $n = m + 1$*, then*

$$\left|w_k + 1 - \exp(2\pi ik/2^{m+7})\right| < 14\epsilon/64$$

for all $0 \le k < 2^n$*.*

Proof. We prove the first part by induction. If $n = 0$, then there is clearly no error, and the result holds. Now assume that the result holds for $n = n_0 \le m$, and consider the behaviour of *expm1_tbl*$(LUT, n_0 + 1, m)$.

Since *expm1_tbl*$(LUT, n_0 + 1, m)$ calls *expm1_tbl*(LUT, n_0, m) and does not subsequently modify the first 2^{n_0} complex values stored in LUT, the required result holds for $0 \le k < 2^{n_0}$; consequently, we need only prove that the result holds for $2^{n_0} \le k < 2^{n_0+1}$.

For such a k, w_k is computed as

$$w_k := x_0 + (w_{k-2^{n_0}} + x_0 w_{k-2^{n_0}})$$

and where x_0 is the value of $\exp(2i\pi 2^{n_0}/2^{m+7}) - 1$ obtained from *omc_s*. The error in w_k is therefore the sum of the following:

1. The error introduced in computing the complex product $x_0 w_{k-2^{n_0}}$. The relative error in computing a complex product less than $\epsilon\sqrt{5}$ [1], so this error is less than $\epsilon\sqrt{5}\,|x_0|\,|w_{k-2^{n_0}}|$. Since $|\exp(ix) - 1| < x$ for $x > 0$, we note that $|w_{k-2^{n_0}}| \le |x_0| \le 2\pi 2^{n_0}/2^{m+7} \le \pi/64$, and consequently the error introduced in computing the complex product is less than $\epsilon\sqrt{5}(\pi/64)^2 2^{n_0}/2^m$.

2. The error introduced in the addition of $w_{k-2^{n_0}}$ and $x_0 w_{k-2^{n_0}}$. The real and imaginary parts of this error are bounded by half of the ulp of the real and imaginary parts of the result respectively. Since the result of this addition is approximately equal to $\exp(2\pi ik/2^{m+7}) - \exp(2\pi i 2^{n_0}/2^{m+7})$, the real error is bounded by

$$\begin{aligned}\frac{1}{2}\mathrm{ulp}(\cos(2\pi 2^{n_0+1}/2^{m+7}) - \cos(2\pi 2^{n_0}/2^{m+7})) &= \frac{1}{2}\mathrm{ulp}\left(\frac{3(2\pi 2^{n_0-m-7})^2}{2}\right)\\ &= \frac{1}{2}\mathrm{ulp}(2^{2n_0-2m-9})\\ &= \epsilon 2^{2n_0-2m-9}\end{aligned}$$

   noting that the value within ulp() can be approximated without changing the result.

   Similarly, the real error is bounded by

$$\begin{aligned}\frac{1}{2}\mathrm{ulp}(\sin(2\pi 2^{n_0+1}/2^{m+7}) - \sin(2\pi 2^{n_0}/2^{m+7})) &= \frac{1}{2}\mathrm{ulp}(2\pi 2^{n_0}/2^{m+7})\\ &= \epsilon 2^{n_0-m-5}\end{aligned}$$

   and thus the absolute complex error is bounded by

$$\begin{aligned}\epsilon\sqrt{(2^{2n_0-2m-9})^2 + (2^{n_0-m-5})^2} &= \epsilon 2^{n_0-m-5}\sqrt{1 + 2^{2n_0-2m-8}}\\ &\le \epsilon 2^{n_0-m-5}\sqrt{\frac{257}{256}} \le 1.002\epsilon \cdot 2^{n_0-m-5}\end{aligned}$$

3. The error introduced in computing the sum of x_0 and $w_{k-2^{n_0}} + x_0 w_{k-2^{n_0}}$. Following the same argument as above, the real error is bounded by

$$\begin{aligned}\frac{1}{2}\mathrm{ulp}(\cos(2\pi 2^{n_0+1}/2^{m+7}) - 1)) &= \frac{1}{2}\mathrm{ulp}(\frac{(2\pi 2^{n_0+1}/2^{m+7})^2}{2})\\ &= \epsilon 2^{2n_0-2m-8}\end{aligned}$$

   and the imaginary error is bounded by

$$\begin{aligned}\frac{1}{2}\mathrm{ulp}(\sin(2\pi 2^{n_0+1}/2^{m+7})) &= \frac{1}{2}\mathrm{ulp}(2\pi 2^{n_0+1}/2^{m+7})\\ &= \epsilon 2^{n_0-m-4}\end{aligned}$$

and consequently the absolute complex error is bounded by

$$\epsilon\sqrt{\left(2^{2n_0-2m-8}\right)^2+\left(2^{n_0-m-4}\right)^2}=\epsilon 2^{n_0-m-4}\sqrt{1+2^{2n_0-2m-8}}$$
$$\leq \epsilon 2^{n_0-m-4}\sqrt{\frac{257}{256}}\leq 1.002\epsilon\cdot 2^{n_0-m-4}$$

4. The error introduced from inaccuracy in the input x_0. This is $\left|W_{m-n_0}-\widehat{W}_{m-n_0}\right|$ for W as defined earlier.

5. The error introduced from inaccuracy in $w_{k-2^{n_0}}$. By assumption, this is bounded by

$$12.75\epsilon\cdot 2^{n_0}/2^{m+7}+\sum_{j=m+1-n_0}^{m}\left|W_j-\widehat{W}_j\right|$$

Adding these together, we obtain

$$\left|w_k+1-\exp(2\pi ik/2^{m+7})\right|<\epsilon\left(12\cdot 1.002+\frac{\pi^2\sqrt{5}}{32}\right)2^{n_0}/2^{m+7}+\left|W_{m-n_0}-\widehat{W}_{m-n_0}\right|$$
$$+12.75\epsilon\cdot 2^{n_0}/2^{m+7}+\sum_{j=m+1-n_0}^{m}\left|W_j-\widehat{W}_j\right|$$
$$<12.75\epsilon\cdot 2^{n_0+1}/2^{m+7}+\sum_{j=m-n_0}^{m}\left|W_j-\widehat{W}_j\right|$$

for all $2^{n_0}\leq k<2^{n_0+1}$ and the result follows.

To establish the second part of the proof, we compute the sum using the following MAPLE code

```
Digits := 50;

ERR_e := proc(x)
        local y, pow;

        if x = 0 then RETURN(0); fi;
        y := abs(x);
        pow := floor(log[2.](y));
        evalf(y - round(y * 2^(52 - pow)) * 2^(pow - 52)) * 2^53;
end;

ERR_c := z -> sqrt(ERR_e(Re(z))^2 + ERR_e(Im(z))^2);

sum( ERR_c(exp(2 * Pi * I / 2^n) - 1), n = 7..29);
```

and obtain $0.01689\ldots\epsilon<1.25\epsilon/64$; adding this to the other $12.75\epsilon/64$ provides the stated bound. □

The function *tricl_roots_makelut*(LUT, n) computes the complex values $\exp(2\pi ik/2^n)$ for $0 \le k < 2^{n-2}$. The value n must satisfy $2 \le n \le 29$, and LUT must have space to store 2^{n-1} doubles (i.e., 2^{n+2} bytes).

```
void tricl_roots_makelut(double * LUT, int n)
{
        double x0r, x0i, x1r, x1i;
        int32_t i, j, N;

        assert(2 <= n && n <= 29);

        if (n == 2) {
                /* We only want k = 0 */
                LUT[0] = 1.0;
                LUT[1] = 0.0;
        } else if (n <= 6) {
                /* Copy values of exp(i x) for 0 <= x < pi / 4 from c_s[] */
                N = 1 << (6 - n);
                for (i = 0; i < (1 << (n - 3)); i++) {
                        LUT[2 * i] = c_s[2 * i * N];
                        LUT[2 * i + 1] = c_s[2 * i * N + 1];
                }

                /* Copy value of exp(i pi / 4) */
                LUT[2 * (1 << (n - 3))] = c_s[2 * 8];
                LUT[2 * (1 << (n - 3)) + 1] = c_s[2 * 8 + 1];

                /*-
                 * Continue by noting that
                 * exp(i (x + pi / 4)) = i conj(exp(i (pi / 4 - x)))
                 */
                for (i = (1 << (n - 3)) + 1; i < (1 << (n - 2)); i++) {
                        LUT[2 * i + 1] = LUT[2 * ((1 << (n - 2)) - i)];
                        LUT[2 * i] = LUT[2 * ((1 << (n - 2)) - i) + 1];
                }
        } else {
                /*
                 * Compute exp(2 pi i k / 2^n) - 1 for 0 <= k < 2^(n - 6)
                 */
                expm1_tbl(LUT, n - 6, n - 7);

                /*
                 * Combine with appropriate exp(2 pi i k / 2^6) to
                 * obtain exp(2 pi i k / 2^n) for 0 <= k < 2^(n - 3)
                 */
```

```
                N = 1 << (n - 6);
                for (i = 7; i >= 0; i--) {
                        for (j = 0; j < N; j++) {
                                x0r = c_s[2 * i];
                                x0i = c_s[2 * i + 1];
                                x1r = LUT[2 * j];
                                x1i = LUT[2 * j + 1];

                                LUT[2 * (i * N + j)] = x0r
                                    + (x0r * x1r - x0i * x1i);
                                LUT[2 * (i * N + j) + 1] = x0i
                                    + (x0r * x1i + x0i * x1r);
                        }
                }

                /* Copy value of exp(i pi / 4) */
                LUT[2 * (1 << (n - 3))] = c_s[2 * 8];
                LUT[2 * (1 << (n - 3)) + 1] = c_s[2 * 8 + 1];

                /*-
                 * Continue by noting that
                 * exp(i (x + pi / 4)) = i conj(exp(i (pi / 4 - x)))
                 */
                for (i = (1 << (n - 3)) + 1; i < (1 << (n - 2)); i++) {
                        LUT[2 * i + 1] = LUT[2 * ((1 << (n - 2)) - i)];
                        LUT[2 * i] = LUT[2 * ((1 << (n - 2)) - i) + 1];
                }
        }

        return;
}
```

Theorem 1. *Given LUT as produced by* tricl_roots_makelut*(LUT, n) with* $2 \leq n \leq 29$*, and making the assignment* $w_k = LUT_{2k} + iLUT_{2k+1}$*,*

$$|w_k - \exp(2\pi ik/2^n)| < \frac{3}{2}\epsilon$$

for all $0 \leq k < 2^{n-2}$*.*

Proof. For $n = 2$, there is only one value produced, and it is given exactly. For $2 < n \leq 6$, the values are copied from the table *c_s*, and the maximum error is trivially $\sqrt{1/2}\epsilon$, well below the bound to be proven. We therefore need only consider the third case in the code, where $n > 6$.

In this case, the value w_k is computed as $x_0 + x_0x_1$, where x_0 is the 2^6th root of unity $\exp(2\pi ik'/2^6)$ for $k' = \lfloor k/2^{n-6} \rfloor$, as stored in the table *c_s*, and x_1 is the value $\exp(2\pi i(k - k'2^{n-6})/2^n) - 1$ as generated by *expm1_tbl*$(LUT, n-6, n-7)$. The error in the values computed is consequently the sum of the following:

1. The error introduced from inaccuracy in x_0. Since $|1 + x_1| = 1$, this has the same norm as the error in the stored value of x_0, i.e., one of 8 possible values depending upon k'.

2. The error introduced from inaccuracy in x_1. Since $|x_0| = 1$, this has the same norm as the error in the computed value of x_1, which is bounded from above by $14\epsilon/64$ as shown in Lemma 1.

3. The error introduced by rounding errors in computing $x_0 + x_0x_1$. The real and imaginary parts of this each consist of four elements: Two errors introduced by multiplications in computing the complex product x_0x_1; the error introduced by the addition used in computing the complex product x_0x_1; and the error introduced by the addition of x_0 and x_0x_1.

   Considering the largest possible values at each point in the computation, we note that the real part of this introduced error is bounded by

$$\frac{1}{2}\mathrm{ulp}(\cos(2\pi k'/2^n)) + \frac{1}{2}\mathrm{ulp}(\cos(2\pi k'/2^n) - \cos(2\pi(k'+1)/2^n)) + \frac{1}{2}\mathrm{ulp}(\cos(2\pi k'/2^n)(1-\cos(2\pi/2^n))) + \frac{1}{2}\mathrm{ulp}(\sin(2\pi k'/2^n)\sin(2\pi/2^n)))$$

   and the imaginary part is bounded by

$$\frac{1}{2}\mathrm{ulp}(\sin(2\pi(k'+1)/2^n)) + \frac{1}{2}\mathrm{ulp}(\sin(2\pi(k'+1)/2^n) - \sin(2\pi k'/2^n)) + \frac{1}{2}\mathrm{ulp}(\sin(2\pi k'/2^n)(1-\cos(2\pi/2^n))) + \frac{1}{2}\mathrm{ulp}(\cos(2\pi k'/2^n)\sin(2\pi/2^n)))$$

   For each of the 8 possible values of k', these two bounds can be computed, and thereby a bound on the largest absolute error introduced by new rounding errors.

By computation, the sum of the bounds given above is maximized for $k' = 6$, where we obtain the maximum error bound of $1.488\ldots\epsilon$, below the required bound of 1.5ϵ. □

Theorem 2. *The 2^nth roots of unity can be computed in double precision using $2n+6$ precomputed values and $\frac{37}{32}2^n$ double precision floating-point operations, with a maximum absolute error of less than 1.5ϵ.*

Proof. The function *tricl_roots_makelut* behaves as required. □

For arbitrary precisions, the precise error bound will vary (since the precise errors in the precomputed constants will be different), but will always be less than 2ϵ for the relevant ϵ.

References

[1] R.P. Brent, C. Percival, P. Zimmermann, *Error Bounds on Complex Floating-Point Multiplication*, manuscript.

A Copyright