Numerical Analysis Takehome Exam 2

Magnus Erik Hvass Pedersen (971055)
January 2005, Daimi, University of Aarhus

1 Introduction

The purpose of this document is to verify attendance of the author to the *Numerical Analasis* course part II, at Department of Computer Science, University of Aarhus.

A handful of functions are to be numerically integrated using socalled *Romberg* integration. As it turns out, the process given in [1] and [4] is not fully automatable, although it may appear so at first glance: For these functions, the process must often be guided by a human operator. To do this, it is necessary to have some understanding of the underlying method of extrapolation, which is used to decrease the errors in the approximations.

First off, all but one of the functions are analytically integrated, where the remaining function will serve as a pure test function, to see how good a result we can obtain, without knowing the true value of the integral in advance. Naturally, the confidence in such a result, must be based solely on knowledge obtained during the numerical process.

Then two basic methods of approximating an integral are given, along with considerations of their individual pro's and con's. These approximations may be improved without further sampling of the function in question, by using a process called *Richardson* extrapolation, which is therefore described next. The exposition of this process that is given here, is perhaps more abstract than those found in the references, in that the socalled *Taylor* expansion is not involved in determining the error-terms. This corresponds closely to the implementation that is given later, in which the human operator is required to *guess* (sensibly of course), the corrective factors.

The document is believed to solve to the fullest degree, the questions posed in the exam. Furthermore, variants of the common Richardson formulae are also given, and so is an implementation for binary summation. Considerations on whether to multiply floating point numbers by 0.5 or divide them by 2, are included also. The paper ends with a discussion of Romberg integration and Richardson extrapolation in general. The reader is assumed to be familiar with the course litterature and the problem description for this exam.

2 Integrals

The exam poses five integrals which are reprinted here for easy reference:

$$\int_0^1 \sin x \; dx \tag{1}$$

$$\int_0^1 \sqrt[3]{x} \; dx \tag{2}$$

$$\int_0^1 \sqrt{x} \ln x \; dx \tag{3}$$

$$\int_0^5 e^{-x} \cos^2(x^2) \; dx \tag{4}$$

$$\int_0^1 \frac{1}{\sqrt{1-x^2}} \; dx \tag{5}$$

Clearly the integrals in Eqs.(3) and (5) are improper, because the functions to be integrated, are undefined in one of their respective endpoints, for Eq.(3) the expression $\ln x$ is undefined in $x = 0$, and for Eq.(5) the expression $1/(\sqrt{1-x^2})$ is undefined for $x = \pm 1$.

3 Analytic Integration

Analytic integration is done by *anti-differentiation*, so the integral:

$$\int f(x) \; dx$$

equals a function $F(x)$ satisfying:

$$\frac{d}{dx} F(x) = f(x)$$

meaning that the definite integral over $[a, b]$, is found as follows:

$$\int_a^b f(x) \; dx = F(x)|_a^b = F(b) - F(a)$$

Such F's are known for the basic mathematical functions f, and there exist various rules for finding the integrals of their combinations (products, sums, functions of functions, etc.), some of which are demonstrated below.

Integrals 1 & 2

The first two integrals in Eqs.(1) and (2) are easily found:

$$\int_0^1 \sin x \; dx = -\cos x|_0^1 = 1 - \cos 1$$

and

$$\int_0^1 \sqrt[3]{x}\ dx = \int_0^1 x^{1/3}\ dx = \frac{3}{4}x^{4/3}\Big|_0^1 = \frac{3}{4}$$

Integral 3

To integrate the product of two functions in Eq.(3) however, first we need to define two auxiliary functions U and V as follows:

$$U = \ln x$$

and

$$\frac{dV}{dx} = \sqrt{x}$$

Then we differentiate U to obtain:

$$\frac{dU}{dx} = \frac{1}{x},\ x > 0$$

and integrate dV/dx:

$$V = \frac{2}{3}x\sqrt{x}$$

which are then applied in partial integration as follows:

$$\int_0^1 \sqrt{x}\ln x\ dx = U \cdot V|_0^1 - \int_0^1 V \cdot \frac{dU}{dx}\ dx = \ln x \cdot \frac{2}{3}x\sqrt{x}\Big|_0^1 - \frac{2}{3}\int_0^1 \sqrt{x}\ dx \quad (6)$$

Now the first part:

$$\ln x \cdot \frac{2}{3}x\sqrt{x}\Big|_0^1$$

is improper due to $\ln x$ being undefined for $x = 0$, as described above. Therefore, we must compute the limit instead:

$$\lim_{c\to 0^+} \ln x \cdot \frac{2}{3}x\sqrt{x}\Big|_c^1 = \ln 1 \cdot \frac{2}{3} - \lim_{c\to 0^+} \ln c \cdot \frac{2}{3}c\sqrt{c} = -\lim_{c\to 0^+} \ln c \cdot \frac{2}{3}c\sqrt{c}$$

Re-arranging this and using *L'Hôpital*'s rule of limits, this equals:

$$-\lim_{c\to 0^+} \frac{\ln c}{1/c} \cdot \frac{2}{3}\sqrt{c} = \lim_{c\to 0^+} \frac{1/c}{1/c^2} \cdot \frac{2}{3}\sqrt{c} = \lim_{c\to 0^+} \frac{2}{3}c\sqrt{c} = 0$$

which means the integral in Eq.(6) equals:

$$-\frac{2}{3}\int_0^1 \sqrt{x}\ dx = \left(\frac{2}{3}\right)^2 x\sqrt{x}|_0^1 = -\frac{4}{9}$$

Integral 4

To analytically integrate Eq.(4) is a bit tedious and is therefore omitted. Thus it serves as an example of a function, of which we do not know the analytic result of integration, and the validity of the numerically obtained results, will therefore have to be argued without a referential value.

Integral 5

Now turning to the last of the five integrals. As with Eq.(3) we are dealing with an improper integral, because a division-by-zero is caused by letting $x = 1$. As before, we proceed by computing the limit:

$$\int_0^1 \frac{1}{\sqrt{1-x^2}}\,dx = \lim_{c\to 1^-}\int_0^c \frac{1}{\sqrt{1-x^2}}\,dx = \lim_{c\to 1^-} \sin^{-1} x\Big|_0^c = \lim_{c\to 1^-} \sin^{-1} c = \frac{\pi}{2}$$

4 Numeric Integration

To compute an integral numerically, we start by computing estimates or approximations by fitting geometric shapes to the function in question, shapes for which the areas are easily computable, and where the approximated integral is then the sum of these areas. Two kinds of shapes are used in the follwoing; first a rectangular shape is used in the midpoint rule of section 4.1, and a trapezoidal shape is then used in section 4.2.

4.1 Midpoint Rule

First denote the actual integral that we wish to approximate by I, that is:

$$I = \int_a^b f(x)\,dx \tag{7}$$

and then let I_m be the approximation calculated from the midpoint rule, which is given by:

$$I_m(n) = h\sum_{i=1}^{n} f(a + \frac{2i-1}{2}h) \tag{8}$$

where n is the number of rectangles we use in the approximation, so h is the socalled step-size between samples of the function f:

$$h = \frac{b-a}{n}$$

One may be convinced that this actually computes the area of a series of rectangular fittings of equal width, by writing out a part of the summation in Eq.(8).

4.2 Trapezoid Rule

Using the trapezoid rule, we instead approximate I by I_t, defined recursively as follows (see [1, p. 503]):

$$I_t(2^k) = \begin{cases} \frac{1}{2}h_k(f(a)+f(b)) & \text{, if } k=0 \\ \frac{1}{2}I_t(2^{k-1}) + h_k \sum_{i=1}^{2^{k-1}} f(a+(2i-1)h_k) & \text{, if } k \geq 1 \end{cases} \tag{9}$$

for some integer k, meaning the number of steps is $n = 2^k$, and with the step-size being:

$$h_k = \frac{b-a}{2^k}$$

Note that the recursive definition implies a doubling of the number of samples used for the approximation, each time we increase the resolution, which is not required in the midpoint rule above.

4.3 Advantages & Disadvantages

Mathematically speaking, both I_t and I_m converge to I with increasing resolution (that is, for $n \to \infty$ we have $\lim I_t = \lim I_m = I$), provided f is continuous over $[a, b]$; but there are different advantages and disadvantages to both kinds of approximations, also depending on the function to be integrated.

In section 7.3 below, it is briefly described how the base-case of the trapezoid rule in Eq.(9) evaluates invalid samples for some of the above functions, while section 4.4 determines that the trapezoid rule uses about half the number of function evaluations of the midpoint rule.

4.4 Number of Function Evaluations

The way we use the approximations I_m and I_t in section 5, is to compute a series of approximations to I, where the number of samples for each approximation, is the doubling of the previous number of samples. That is, we calculate $I_m(n)$ or $I_t(n)$ for $n = 1, 2, 4, 8, 16, 32, \cdots$, or stated in another way, we perform $k+1$ iterations, starting with $n = 2^0 = 1$, and then double the number of samples for each iteration, so for the $(k+1)$'th step we use 2^k samples of the function f.

To count the number of function evaluations required for $k+1$ such refinements to the approximation I_m, define $C_m(k)$ as follows:

$$C_m(k) = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

where the first identity is simply a count of the number of samples, required in Eq.(8) for each of the iterations. The last identity can be proven inductively. The total number of function evaluations required in $k+1$ steps of the trapezoid rule, can be written recursively directly from Eq.(9):

$$C_t(k) = \begin{cases} 2 & \text{, if } k=0 \\ C_t(k-1) + 2^{k-1} & \text{, if } k \geq 1 \end{cases}$$

which may be written out as follows:

$$C_t(k) = 2^{k-1} + 2^{k-2} + 2^{k-3} + \cdots + 2^2 + 2^1 = \sum_{i=1}^{k-1} 2^i = 2^k - 2$$

meaning that:

$$C_m(k) \simeq 2 \cdot C_t(k)$$

Or put in other words; refining the integral approximations with the midpoint rule by doubling the number of function-samples, requires about twice as many function evaluations as the trapezoid rule.

This difference can be understood more intuitively by drawing the range $[a, b]$ and marking the samples that are required by the midpoint rule, for the first few times the number of samples is doubled. Clearly the trapezoid rule re-uses all the previous samples, which is indeed why Eq.(9) is recursive, while the midpoint rule re-uses none of the samples.

It is possible to modify the iterations of the midpoint rule, so instead of doubling the number of samples n, we instead increase it to $2n - 1$. This way we may re-use the function evaluations from $I_m(n)$ in the calculation of $I_m(2n-1)$ so as to obtain a recursive formula similar to Eq.(9). However, this has other implications and is therefore not used presently. For example, the particular simple mathematical analysis in section 7.6, would no longer be valid, and moreover, the corrective factors used in Romberg integration, would appear unfamiliar.

5 Richardson Extrapolation

The idea behind *Richardson Extrapolation*, is to deduce from a number of approximations, an error-factor which can be used in making refinements to the existing approximations. The method does not require any further function evaluations, and is therefore very cheap to perform. Furthermore, it is not limited to numeric integration, and is therefore described in a more abstract context in the following.

5.1 Approximations & Errors

First let $M(f)$ denote the true or actual value of a given mathematical problem, for example the value I of the definite integral of f over some range, as given in Eq.(7). An approximation to this, is then given in $N(f, k)$ where f again indicates the underlying function, and k is a degree of refinement. Omitting the explicit reference to f and writing M as the approximation plus some error-term $E(k)$, we arrive at:

$$M = N(k) + E(k) \tag{10}$$

with $k = 0$ being the most coarse approximation, refined with increasing k. Consider two consecutive approximations, one for k and one for $k+1$ given by:

$$M = N(k+1) + E(k+1) \tag{11}$$

Assuming $E(k) \neq 0$ (otherwise $M = N(k)$ and the approximation would be completely accurate), there exists some $\alpha_k \in \mathbb{R}$ so that:

$$\alpha_k = \frac{E(k+1)}{E(k)} \;\Leftrightarrow\; \alpha_k \cdot E(k) = E(k+1) \tag{12}$$

which means Eq.(11) becomes:

$$M = N(k+1) + \alpha_k \cdot E(k)$$

and subtracting Eq.(11) from this, we have:

$$0 = N(k+1) - N(k) + (\alpha_k - 1) \cdot E(k) \;\Leftrightarrow\; E(k) = \frac{N(k) - N(k+1)}{\alpha_k - 1} \tag{13}$$

At first sight this does not appear very useful; we have found the error $E(k)$, provided we know α_k, which was defined in terms of $E(k)$.

However, the error $E(k)$ is generally an infinite series of lesser error terms, appearing with various weights for different k. So if $|E(k)|$ decreases with increasing k, then the α_k's converge to some limit α, which is clearly the dominating error-factor, wherefore its error-terms may be removed from the approximations $N(k)$, to generate a new sequence of approximations $N_1(k)$ as follows:

$$N_1(k) = N(k) + \frac{N(k) - N(k+1)}{\alpha - 1} \simeq N(k) + E(k)$$

which may then be subject to another iteration of Richardson extrapolation, where the next error-term is removed, and so on. The general formula for this process is:

$$N_{m+1}(k) = N_m(k) + \frac{N_m(k) - N_m(k+1)}{\alpha - 1} \tag{14}$$

with a new α for each iteration m, and $N_0(k)$ being the basic approximations.

5.2 Determining The Corrective Factor

The question is then how to find the corrective factor α to be used in Eq.(14). Consider the error in Eq.(13) for $k+1$:

$$E(k+1) = \frac{N(k+1) - N(k+2)}{\alpha_{k+1} - 1}$$

Since $E(k+1) = \alpha_k \cdot E(k)$ also, we have:

$$\alpha_k \cdot E(k) = \frac{N(k+1) - N(k+2)}{\alpha_{k+1} - 1}$$

and then using Eq.(13) and rewriting a bit, we finally obtain:

$$\alpha_k \cdot \frac{\alpha_{k+1} - 1}{\alpha_k - 1} = \frac{N(k+1) - N(k+2)}{N(k) - N(k+1)}$$

Now, assuming the α_k's indeed do converge to some number, then:

$$\frac{\alpha_{k+1}-1}{\alpha_k-1} \to 1$$

so that:

$$\alpha_k \simeq \frac{N(k+1)-N(k+2)}{N(k)-N(k+1)}$$

To determine α, we therefore compute a number of such quotients, and take a guess at what the true value for α is.

5.3 Common Formulae

The reader may have noticed Eq.(14) being different from what is common in the litterature. The reason for this equation being turned around, begins with the relationship designated by α_k in Eq.(12). Let us instead consider $\beta_k \in \mathbb{R}$ to be the number which satisfies:

$$\beta_k \cdot E(k+1) = E(k)$$

We then have:

$$M = N(k) + E(k) = N(k) + \beta_k \cdot E(k+1)$$

and subtracting Eq.(11) from this, we get:

$$0 = N(k) - N(k+1) + (\beta_k - 1)E(k+1) \;\Leftrightarrow\; E(k+1) = \frac{N(k+1)-N(k)}{\beta_k - 1} \tag{15}$$

As with the α_k's we once more seek a limit β for the β_k's, which would make us able to use this error-estimate in a corrective manner, obtaining $N_1(k)$ from $N(k)$:

$$N_1(k+1) = N(k+1) + \frac{N(k+1)-N(k)}{\beta-1} \simeq N(k+1) + E(k+1)$$

or identically, but with adjusted indices k:

$$N_1(k) = N(k) + \frac{N(k)-N(k-1)}{\beta-1} \simeq N(k) + E(k)$$

which again may be subject to another iteration of Richardson extrapolation, with the general formula now being:

$$N_{m+1}(k) = N_m(k) + \frac{N_m(k)-N_m(k-1)}{\beta-1} \tag{16}$$

and again with a new β for each iteration of the process. This is the commonly used formula in the litterature, and is also used in the implementation below.

The reason for this, is that for well-behaved functions for which the approximations $N_0(k)$ have their resolutions doubled for each k, the values of β are certain positive powers of 2, as detailed in [4]. This makes it easier to recognize the corrective factors in play, than with α where they appear reciprocal.

Still we need a way to make an educated guess for the value of β. The error-term in Eq.(15) for k is:

$$E(k) = \frac{N(k) - N(k-1)}{\beta_{k-1} - 1}$$

but we had $\beta_k \cdot E(k+1) = E(k)$ from above, so this becomes:

$$\beta_k \cdot E(k+1) = \frac{N(k) - N(k-1)}{\beta_{k-1} - 1}$$

then using Eq.(15) and rearranging, we find:

$$\beta_k \cdot \frac{\beta_{k-1} - 1}{\beta_k - 1} = \frac{N(k) - N(k-1)}{N(k+1) - N(k)}$$

and assuming the β_k's converge, this means:

$$\beta_k \simeq \frac{N(k) - N(k-1)}{N(k+1) - N(k)}$$

6 Romberg Integration

Performing Romberg integration is now just a matter of using Eq.(16) with $N_0(k)$ being approximations to I from Eq.(7), for example:

$$N_0(k) = I_t(2^k)$$

with $I_t(2^k)$ being the Trapezoid approximation to I, as given in section 4.2. Similarly we have the following if the midpoint rule from section 4.1 is desired instead:

$$N_0(k) = I_m(2^k)$$

It is customary to change notation for Romberg integration, so using the trapezoid rule, we would write:

$$R_{k,m} = \begin{cases} I_t(2^k) & \text{, if } m = 0 \\ R_{k,m-1} + \frac{R_{k,m-1} - R_{k-1,m-1}}{\beta - 1} & \text{, if } m \geq 1 \end{cases}$$

with β being 2^{2m} for smooth functions [4].

7 Implementation

Implementation is carried out in *MS Visual C++ .NET* and compiles to an executable for *MS Windows*. The source-code relies on template classes for easy specialization and change of floating point datatypes. Socalled *assertions* are used throughout for easy debugging, and do not compile into the final *release-build*. Assertions also makes it easier to get an overview of valid parameters, states, etc., thus helping maintenance of the source-code.

7.1 Basics

The entry-point for the program is found in the file `na_exam2.cpp` along with a few definitions and auxiliary functions. The first of the two type definitions:

```
typedef double FTYPE;
typedef FTYPE (*FPTR)(FTYPE);
```

is used for the floating point datatype, and the second defines a datatype for a function taking such a floating point as an argument, and returning another.

The two functions `InputFunction()` and `InputRomberg()` are used to get the parameters from the user, that are to be used for Romberg integration. `GetFunction()` returns a pointer to the function which the user chose to integrate; their implementations are described in section 7.2. `GetRomberg()` creates a new object for performing Romberg integration, with either the trapezoid or the midpoint rule, depending on the user's choice.

All of this comes together in the main-function, which serves as the program's entry-point:

```
int _tmain(int argc, _TCHAR* argv[])
{
    cout << "Romberg Integration, 2004-2005 Magnus ...";
    InputRomberg();
    InputFunction();

    NA::LRichardsonExtrapolation<FTYPE>* romberg = GetRomberg();

    romberg->Extrapolate(gA, gB, gNumIterations, GetFunction());

    delete romberg;

    return 0;
}
```

7.2 Test Functions

The functions from Eqs.(1-5) are implemented as follows:

```
FTYPE Function1(FTYPE x) { return sin(x); }

FTYPE Function2(FTYPE x) { return pow(x, 1.0/3.0); }

FTYPE Function3(FTYPE x) { return sqrt(x)*log(x); }

FTYPE Function4(FTYPE x)
{
    FTYPE c = cos(x*x);
    return exp(-x)*c*c;
}

FTYPE Function5(FTYPE x) { return 1.0/sqrt(1-x*x); }
```

where we use $\sqrt[3]{x} = x^{1/3}$ in Eq.(2). Note that the functions are implemented directly, as we are only considering the Romberg process, and not whether there exist rewrites for the functions themselves, that improve their precision.

7.3 Function Sampler

Both the midpoint and trapezoid rules from Eqs.(8) and (9), make use of function samples. To compute these samples and fill them into an array, the following function is supplied in the file `FunctionSampler.h`. However, it computes $f(a + (2i - 1)h)$, so in order to use it with the midpoint rule which requires $f(a + (2i - 1)h'/2)$, we simply let $h = h'/2$. The function is as follows:

```
template <class T>
void FunctionSampler(T *array, T a, T h,
                     unsigned int n, T (*f)(T))
{
    assert(array && n>0);
    assert(f);

    for (int i=0; i<n; i++)
    {
        // Remember that C++ indexes arrays from 0 to n-1,
        // so we need to adjust for this.
        int j = i+1;
        T y = f(a+(2*j-1)*h);
        array[i] = ValidFloat(y);
    }
}
```

where it should be noted that the right end-point of the sampling-range, i.e. b, is not needed as it is implicitly given in the step-size h. The call to `ValidFloat()` is used to weed out invalid floating point numbers, resulting from improper integrals being approximated, for example when sampling $\ln x$ with $x = 0$. This auxiliaray function is given by:

```
template <class T>
T ValidFloat(T x)
{
    return x;
}
```

which is then specialized for the `double` datatype as follows, setting invalid numbers to zero (which naturally affects the approximation being calculated):

```
template <>
double ValidFloat(double x)
{
    return (_isnan(x) ||
            _fpclass(x) == _FPCLASS_PINF ||
            _fpclass(x) == _FPCLASS_NINF) ? (0) : (x);
}
```

where it should be noted that `_isnan()` and the other functions and constants, are specific to the MS Windows implementation, and are used to determine whether `x` is *not-a-number*, *positive infinite*, or *negative infinite*. These special erroneous floating point numbers, are really just encoded as simple bit-patterns (see for example [1, p. 42]), so similar tests could be made for any platform using IEEE floating point numbers, and not just MS Windows. Finally, the single-precision floating point specialization of `ValidFloat()` simply calls the double-precision specialization:[1]

```
template <>
float ValidFloat(float x)
{
    return (float) ValidFloat<double>(x);
}
```

7.4 Richardson & Romberg

The base class for performing Richardson extrapolation, is `LRichardsonExtrapolation`. Again, it is a template class allowing for easy change of datatypes. The class contains two primary functions, the first is for performing the extrapolation and is given by:

```
void Extrapolate (T a, T b, const int n, T (*f)(T))
{
    T *estimates = new T[n];
    T *estimates2 = new T[n];  // One more than needed, easens indexing.
    T *differences = new T[n]; // One more than needed.
    T *quotients = new T[n];   // Two more than needed.
```

[1]Naturally, automatic type promotion does not take place with template specializations, as the most general template will occur as an exact type-match.

```
    // Call specialization to get initial estimates.
    ComputeEstimates(estimates, a, b, n, f);

    for (int m=1; m<n; m++)
    {
        int k;

        // Compute differences.
        for (k=m; k<n; k++)
        {
            differences[k] = estimates[k] - estimates[k-1];
        }

        // Compute quotients.
        for (k=2; k<n; k++)
        {
            quotients[k] = differences[k-1] / differences[k];
        }

        Output(m, n, estimates, differences, quotients);

        // Get corrective factor from user.
        T factor;
        std::cout << "Enter factor: ";
        std::cin >> factor;

        // Compute next series of estimates/approximations.
        for (k=m; k<n; k++)
        {
            estimates2[k] = estimates[k] + (estimates[k] -
                                estimates[k-1]) / (factor-1);
        }

        // Swap arrays.
        std::swap(estimates, estimates2);
    }

    Output(n, n, estimates, differences, quotients);

    delete [] estimates;
    delete [] estimates2;
    delete [] differences;
    delete [] quotients;
}
```

The arguments to this function are the two endpoints, the number of iterations for the Richardson process, and a pointer to the function that we wish to compute the approximations for. The implementation could be done with just a single array instead of the above four, but the above approach is believed to be more readable. Some floating point units (FPU) have extended internal precision, so using just a single array and keeping the intermediate results in the FPU, could possibly increase precision somewhat.

The call to `ComputeEstimates()` in `Extrapolate()` above, is specialized in the two classes `LRombergTrapezoid` and `LRombergMidpoint` which compute the initial approximations using the trapezoid and midpoint rules, respectively. The first specialization using the trapezoid rule, is as follows:

```
// Compute n integrals of f and place them in the array for
// 'estimates'.
virtual void ComputeEstimates (T *estimates, T a, T b, int n,
                               T (*f)(T))
{
    assert(estimates && n>0);
    assert(a < b);
    assert(f);

    // Allocate an array that is big enough to hold samples for
    // all integrals, that is, allocate enough samples for the
    // integral with finest resolution.
    T *samples = new T[1<<(n-2)];

    // Initialize the step-size.
    T h = b-a;

    // Compute the first trapezoid integral.
    estimates[0] = 0.5*h*(ValidFloat(f(a)) + ValidFloat(f(b)));

    // Compute the rest of the trapezoid integrals, by doubling
    // the number of samples, and thus halving the step-size for
    // the value sampling.
    for (int i=1; i<n; i++)
    {
        int numSamples = 1<<(i-1); // numSamples = pow(2, i-1)

        // Halve the step-size for the sampling.
        h *= 0.5;

        // Compute the samples of f.
        FunctionSampler(samples, a, h, numSamples, f);

        // Sum the samples, and compute the integral.
        estimates[i] = 0.5 * estimates[i-1] +
                       h*ArrayOps::BinarySummation<T>(samples,
                                                      numSamples);
    }

    // Delete the locally used array for the samples.
    delete [] samples;
}
```

And the specialization in `LRombergMidpoint` is similar:

```
virtual void ComputeEstimates (T *estimates, T a, T b, int n,
                               T (*f)(T))
{
    assert(estimates && n>0);
    assert(a < b);
    assert(f);

    // Allocate an array that is big enough to hold samples for
    // all integrals, that is, allocate enough samples for the
    // integral with finest resolution.
    T *samples = new T[1<<(n-1)];

    // Initialize the step-size.
    T h = b-a;

    // Compute the midpoint integrals, by doubling the number of
    // samples in each iteration, and thus halving the step-size
    // for the value sampling.
    for (int i=0; i<n; i++)
    {
        int numSamples = 1<<i;  // numSamples = pow(2, i)
        T h2 = h;               // Store the step-size.

        // Halve the step-size for the sampling.
        h *= 0.5;

        // Compute the samples of f.
        FunctionSampler(samples, a, h, numSamples, f);

        // Sum the samples, and compute the integral.
        estimates[i] = h2*ArrayOps::BinarySummation<T>(samples,
                                                       numSamples);
    }

    // Delete the locally used array for the samples.
    delete [] samples;
}
```

Note the use of `BinarySummation()` in both of these functions, which is described next.

7.5 Binary Summation

Looking at Eqs.(8) and (9) and how they are used in section 6, we see that the summations are of 2^k numbers for some integer k. Although it is possible to perform the following, socalled *binary summation*, with any number of terms, a number which is a power of 2, is particularly well suited. The idea behind binary summation, is to lower the rounding error, which can be understood by considering the summation:

$$\sum_{i=1}^{n} x_i$$

where the direct implementation would be a simple for-loop, similar to:

```
T sum = 0;
for (int i=0; i<n; i++)
{
    sum += x[i];
}
```

The second iteration adds `x[1]` to `x[0]` and may generate a rounding-error ϵ,[2] which will propagate through the rest of the summation, possibly creating rounding-errors ϵ_i on top of ϵ. This means the error could, in the worst case, depend linearly on the number of elements to be summed.

Re-organizing the calculation as described in [3], so the summation occurs in a manner resembling a binary tree, each error may at most be part of $\log_2 n$ summation operations, and therefore improves the worst-case error accumulation. The implementation is found in `BinarySummation.h` and is as follows:

```
template <class T>
T BinarySummation(T *array, int n)
{
    assert(array && n>0);

    int k=n;

    while (k>1)
    {
        // Ensure n is a power of 2.
        assert(k%2 == 0);
```

[2]In IEEE floating point arithmetics, the first iteration adding $x[0]$ to 0 is exactly $x[0]$, which is a (very sound) requirement.

```
        for (int i=0, j=0; i<k; i+=2, j++)
        {
            T t = array[i] + array[i+1];
            array[j] = t;
        }

        k >>= 1; // k /= 2
    }

    return array[0];
}
```

Note that this implementation assumes n is a power of 2, and it also destroys the contents of the supplied array of numbers, neither of which is strictly necessary.

7.6 Floating Point Halving

Let $x = S \times 2^E$ and $y = T \times 2^F$ be two floating point numbers. Their product is then given by:

$$x \times y = (S \times T) \times 2^{E+F}$$

whose machine-realization is discussed in [2, p. 37]. Clearly we may set $y = 2^{-1}$, thus obtaining:

$$x \times y = S \times 2^{E-1} \tag{17}$$

Similarly, we have that division by $y = 2^1$ also yields Eq.(17) as we take the difference $E - F$ for the exponent, and the quotients of the mantissas (which is S). So it does not matter whether we multiply by 0.5 or divide by 2. Note that both operations could be implemented by simple decrementing of x's exponent (doubling x would then be the incrementing of its exponent), instead of a full-blown multiplication (or division). However, such a special instruction for halving floating point numbers, should not be assumed to exist, and we should therefore use multiplication of 0.5, as it is generally expected to be much less expensive than division by 2.

The above naturally assumes x to be normalized, but fortunately generalizes well to denormalized numbers, in which case the mantissa of x is merely bit-shifted. An interesting and most relevant question, is then how many times a given floating point number can be halved, before it becomes zero. This of course depends on the initial value of the number, but let us assume it is 1, which is not just a nice round number, but is also common for the initial step-size in the above integrals (i.e. $b - a$ with $b = 1$ and $a = 0$).

So starting with $x = 1 = 1 \times 2^0$, we may halve it by decrementing the exponent which is zero, until x underflows and becomes denormalized. From [2, Table 4.3, p. 22] we find the lowest normalized number in single-precision to be 2^{-126}, and 2^{-1022} in double-precision. As we typically only require step-sizes in the area of 2^{-10} for Romberg integration, we will not encounter any problems with underflowing step-sizes, and thus disregard denormalized step-sizes altogether.

8 Experimental Results

The following is an experimental study of Romberg integration on the five integrals from section 2. Again, it should be stressed that in the course material, there is no algorithmic strategy for choosing the corrective factors β, that will work for all functions, although we may fall back on $\beta = 2^{2m}$ whenever the convergence becomes unclear. This means that the following merely uses sensible arguments to justify why the given factors are chosen. As other factors could prove to lower the approximation error even further, the results are indeed experimental. This however, is believed to be in accord with expectations.

Integral 1

The value of Eq.(1) is only known in terms of the cosine. A small C++ program printing an approximation is given by:

```
double d = 1-cos(1.0);
cout << setprecision(25) << d << endl;
```

which prints the number

$$0.45969769413186029 \tag{18}$$

Using the trapezoid rule and 10 computations of $I_t(2^k)$, we first have table 1 where it is clear that the corrective error-factor should be $\beta = 4$, which yields table 2. There, choosing $\beta = 16$ results in table 3, where the erratic behaviour due to rounding errors, starts to show up. It is still clear however, that $\beta = 64$, which indeed also lowers the difference between successive approximations, as can be seen in table 4. Guessing that $\beta = 256$ we arrive at table 5 where the quotients are impossible to deduce anything from, so we try with $\beta = 2^{2m} = 1024$ which appears to be correct as the differences are brought down slightly, as shown in table 6. There is no point in continuing because the error appears to be completely dominated by rounding errors, and the different approximations for the integral agree on so many decimals. The best approximation found is therefore:

$$R_{6,5} = 0.45969769413186023$$

that is, we only required $k = 6$ and $m = 5$, which means we needed a total of $C_t(k) = C_t(6) = 2^6 - 2 = 62$ function evaluations. Had we used the midpoint rule instead, this number would have been $C_m(6) = 2^{6+1} - 1 = 127$.

We used 10 iterations because it is easier to read the convergence of the quotients that way. But since the function $\sin(x)$ is about as smooth as they come, the error-factors are expected to be 2^{2m}, and the convergence does not need to be ascertained from the calculations. So it would indeed have been sufficient to iterate up until $R_{6,5}$.

The approximation is not identical to Eq.(18) in the last digit, so which one should we believe to be the true value of the integral, if any of them? The system-call to `std::cos()` is of course also an approximation, but it could be expected to be rather accurate to this degree and beyond, while we will see below for some of the other functions, that the Romberg integral is often not exact to that many digits. So the last digit should probably be closer to 9 as in Eq.(18) than the 3 for $R_{6,5}$.

Integral 2

The value of Eq.(1) is known to be 3/4 (see section 3). Let us use the trapezoid rule once more, and again with 10 computations of $I_t(2^k)$. These results are shown in table 7 where the quotients do not converge to anything we would expect. However, we may try and remove another error-term first, which could reveal the point of convergence for the dominating term. Using $\beta = 4$ we arrive at table 8 which indeed shows convergence of the quotients, to a value of $\beta = 2.5198$. This brings us to table 9 where the quotients begin to behave oddly, as the first few quotients could indicate an error-factor of $\beta = 16$, where the last few seem to suggest one closer to 2.5. Trying with $\beta = 16$ we get table 10, where the quotients appear to converge to $\beta = 2.5198$ (again). Using this corrective error-factor, table 11 results, where a qualified guess for β is 64. This yields table 12, where the quotients are hard to deduce anything from, but $\beta = 256$ does not seem to be completely off charts. The error is brought down slightly, but it is difficult to deduce anything from the two quotients in table 13. However, the last quotient is 1.8098, which is not that far from 2.5198, so let us use this once more. Fortunately, this brought down the error even more as shown in table 14. By now there appears to be a system to the errors in this integral, as $\beta = 2.5198$ exists at several levels, and interleaved by $\beta = 2^{2m'}$ for some m'. Now trying $\beta = 1024$ results in table 15. As there are only two approximations available in this table, there are not enough to calculate even a single quotient. A shot in the dark is therefore $\beta = 2.5198$ which means

$$R_{9,9} = 0.75000000000000588$$

but we should not place any confidence in this value at all (even if it was good), as it originates from a wild guess. The best value obtained is therefore:

$$R_{9,8} = 0.74999999999999989$$

which is quite close to the actual value of 3/4. Even if we did not have a reference-value, the Romberg process was well-behaved in terms of differences between approximations getting smaller, and the quotients converging. Since the last difference is of the order $\sim 9e-15$, the above result is by itself (that is, without a reference value), believed to be of high accuracy.

To summarize, the corrective quotients used, were as follows: 4, 2.5198, 16, 2.5198, 64, 256, 2.5198, 1024, 2.5198. The number of function evaluations required were: $C_t(k) = C_t(9) = 2^9 - 2 = 510$.

k	$R_{k,m}$	difference	quotient
0	0.42073549240394825		
1	0.45008051550407563	0.029345023100127376	
2	0.45730093757150209	0.0072204220674264574	4.0642
3	0.45909897349172157	0.0017980359202194873	4.0157
4	0.45954804321221476	0.00044906972049318927	4.0039
5	0.45966028322883579	0.00011224001662102356	4.001
6	0.4596883415202746	2.8058291438814997e-005	4.0002
7	0.45969535598609934	7.0144658247395242e-006	4.0001
8	0.45969710959586596	1.7536097666193129e-006	4
9	0.45969754799788953	4.3840202357259273e-007	4

Table 1: Romberg integration of Eq.(1). Trapezoid approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
1	0.45986218987078475		
2	0.45970774492731092	-0.00015444494347383042	
3	0.4596983187984614	-9.4261288495212092e-006	16.385
4	0.45969773311904583	-5.856794155767453e-007	16.094
5	0.45969769656770948	-3.6551336346501984e-008	16.023
6	0.45969769428408752	-2.2836219581989781e-009	16.006
7	0.45969769414137424	-1.4271328563353336e-010	16.001
8	0.45969769413245481	-8.9194207575360451e-012	16
9	0.45969769413189737	-5.574429806642911e-013	16.001

Table 2: Romberg integration of Eq.(1) with $m = 1$.

k	$R_{k,m}$	difference	quotient
2	0.45969744859774603		
3	0.45969769038987146	2.417921254327382e-007	
4	0.45969769407375144	3.6838799788441179e-009	65.635
5	0.45969769413095374	5.7202298453518097e-011	64.401
6	0.45969769413184608	8.9234175604246957e-013	64.104
7	0.45969769413186001	1.3933298959045715e-014	64.044
8	0.45969769413186018	1.6653345369377348e-016	83.667
9	0.45969769413186023	5.5511151231257827e-017	3

Table 3: Romberg integration of Eq.(1) with $m = 2$.

k	$R_{k,m}$	difference	quotient
3	0.45969769422784168		
4	0.45969769413222572	-9.5615959594397282e-011	
5	0.45969769413186173	-3.6398661862335757e-013	262.69
6	0.45969769413186023	-1.4988010832439613e-015	242.85
7	0.45969769413186023	0	-Inf
8	0.45969769413186018	-5.5511151231257827e-017	0
9	0.45969769413186023	5.5511151231257827e-017	-1

Table 4: Romberg integration of Eq.(1) with $m = 3$.

k	$R_{k,m}$	difference	quotient
4	0.45969769413185074		
5	0.45969769413186029	9.5479180117763462e-015	
6	0.45969769413186023	-5.5511151231257827e-017	-172
7	0.45969769413186023	0	-Inf
8	0.45969769413186018	-5.5511151231257827e-017	0
9	0.45969769413186023	5.5511151231257827e-017	-1

Table 5: Romberg integration of Eq.(1) with $m = 4$.

k	$R_{k,m}$	difference	quotient
5	0.45969769413186029		
6	0.45969769413186023	-5.5511151231257827e-017	
7	0.45969769413186023	0	-Inf
8	0.45969769413186018	-5.5511151231257827e-017	0
9	0.45969769413186023	5.5511151231257827e-017	-1

Table 6: Romberg integration of Eq.(1) with $m = 5$.

k	$R_{k,m}$	difference	quotient
0	0.5		
1	0.64685026299204984	0.14685026299204984	
2	0.70805533683690158	0.061205073844851743	2.3993
3	0.7330999621532317	0.025044625316330116	2.4438
4	0.74322952026024447	0.01012955810701277	2.4724
5	0.74729720168302616	0.0040676814227816882	2.4903
6	0.74892341037274324	0.0016262086897170791	2.5013
7	0.74957175924145636	0.00064834886871312225	2.5082
8	0.74982980356996798	0.00025804432851161963	2.5125
9	0.74993239525876632	0.00010259168879833691	2.5153

Table 7: Romberg integration of Eq.(2). Trapezoid approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
1	0.69580035065606649		
2	0.72845702811851887	0.032656677462452377	
3	0.7414481705920084	0.012991142473489536	2.5138
4	0.74660603962924876	0.0051578690372403591	2.5187
5	0.74865309549062009	0.0020470558613713274	2.5197
6	0.74946547993598223	0.00081238444536213539	2.5198
7	0.74978787553102744	0.00032239559504521065	2.5198
8	0.74991581834613852	0.00012794281511108174	2.5198
9	0.74996659248836572	5.0774142227205665e-005	2.5198

Table 8: Romberg integration of Eq.(2) with $m = 1$.

k	$R_{k,m}$	difference	quotient
2	0.74994451164428044		
3	0.74999609957838131	5.1587934100871813e-005	
4	0.74999982107235996	3.7214939786478496e-006	13.862
5	0.75000001999474653	1.9892238656282757e-007	18.708
6	0.75000001372027103	-6.2744754947274828e-009	-31.703
7	0.75000000580806736	-7.9122036700240983e-009	0.79301
8	0.75000000232765651	-3.4804108484109975e-009	2.2734
9	0.75000000092515162	-1.4025048900023762e-009	2.4816

Table 9: Romberg integration of Eq.(2) with $m = 2$.

k	$R_{k,m}$	difference	quotient
3	0.74999953877398806		
4	0.75000006917195849	5.3039797043297199e-007	
5	0.75000003325623898	-3.5915719509560518e-008	-14.768
6	0.75000001330197263	-1.9954266350374894e-008	1.7999
7	0.75000000528058708	-8.0213855557786928e-009	2.4876
8	0.75000000209562911	-3.1849579640308434e-009	2.5185
9	0.7500000008316513	-1.2639778113054945e-009	2.5198

Table 10: Romberg integration of Eq.(2) with $m = 3$.

k	$R_{k,m}$	difference	quotient
4	0.75000041816391172		
5	0.75000000962436675	-4.0853954497155343e-007	
6	0.75000000017243829	-9.4519284621341626e-009	43.223
7	0.75000000000266531	-1.6977297345732723e-010	55.674
8	0.74999999999998634	-2.6789681584205027e-012	63.373
9	0.74999999999997757	-8.7707618945387367e-015	305.44

Table 11: Romberg integration of Eq.(2) with $m = 4$.

k	$R_{k,m}$	difference	quotient
5	0.75000000313961201		
6	0.75000000002240763	-3.117204383507044e-009	
7	0.74999999999997047	-2.2437163238464564e-011	138.93
8	0.74999999999994382	-2.6645352591003757e-014	842.07
9	0.74999999999997746	3.3639757646142243e-014	-0.79208

Table 12: Romberg integration of Eq.(2) with $m = 5$.

k	$R_{k,m}$	difference	quotient
6	0.7500000000101833		
7	0.74999999999988243	-1.0300871267077127e-011	
8	0.74999999999994371	6.1284310959308641e-014	-168.08
9	0.74999999999997757	3.3861802251067274e-014	1.8098

Table 13: Romberg integration of Eq.(2) with $m = 6$.

k	$R_{k,m}$	difference	quotient
7	0.74999999999310463		
8	0.74999999999998401	6.87938594978732e-012	
9	0.74999999999999989	1.5876189252139739e-014	433.31

Table 14: Romberg integration of Eq.(2) with $m = 7$.

k	$R_{k,m}$	difference	quotient
8	0.74999999999999079		
9	0.74999999999999989	9.1038288019262836e-015	

Table 15: Romberg integration of Eq.(2) with $m = 8$.

Integral 3

For Eq.(3), the error for the first subinterval in calculating I_m with step-size h, is given by:

$$\int_0^h \sqrt{x}\ln x \, dx - h\sqrt{\frac{h}{2}} \cdot \ln\frac{h}{2}$$

which is known from the analytical integration above, to be:

$$\left(\frac{2}{3}\right)^2 x\sqrt{x}\Big|_0^h - h\sqrt{\frac{h}{2}} \cdot \ln\frac{h}{2} = \left(\frac{2}{3}\right)^2 h\sqrt{h} - h\sqrt{\frac{h}{2}} \cdot \ln\frac{h}{2}$$

For $h \to 0$, this approaches a dominating factor of $h\sqrt{h} = h^{3/2}$, meaning that we should expect to see at least one quotient converging to $2^{3/2} \simeq 2.828427$.

As the first trapezoid approximation attempts to evaluate the function at $x = 0$, which is invalid due to $\ln x$, we expect $I_t(2^0)$ to be useless. The trapezoid approximations for 10 iterations are shown in table 16, while the midpoint approximations are found in table 17. Interestingly, the quotients are more consistent for the trapezoid approximations, so we proceed with those. Here, the quotients certainly do not converge to 4, but could very well converge to $2^{3/2}$ as described above. Trying this value for β, we arrive at table 18 where the differences are improved, and where the quotients again suggest a corrective factor of $\beta = 2^{3/2}$. Using this factor again, table 19 results, which also shows an improvement in the differences, and with a clear convergence of the quotients, to $\beta = 4$. Proceeding with this choice of β, gives us table 20, with a clear improvement in differences, but less clear behaviour of the quotients. A guess which is sufficiently close however, would be $\beta = 16$, resulting in table 21. Now it is even more difficult to see which number the quotients would converge to, if it had not been for rounding errors. However, as we have just tried 4 and 16 for β, and 64 is not that far from the quotients in table 21, let us try that. This brings us to table 22, where the last quotient suggests $\beta = 2^{3/2}$ once more. Using this yields table 23, whose quotients could appear to converge to $\beta = 256$, which is the next regular factor that we should try anyway. Table 24 does not reveal anything really, about the true value of the quotient, so let us just continue with $\beta = 1024$ which is next in line, and finally let us use $\beta = 4096$ for table 25, which brings us to the final value:

$$R_{9,9} = -0.4444444444444442$$

which printed with the given precision, was already obtained at $R_{9,7}$. Ignoring that this is very close to our analytic value for this integral, it should be considered of good precision by itself, because the first (and most influential) handful of error-terms that were removed, were so wellbehaved. For the 10 gradual refinements of trapezoid approximations, we required a total of $C_t(9) = 2^9 - 2 = 510$ function evaluations.

The corrective factors β that were used, are summarized as follows: $2^{3/2}$, $2^{3/2}$, 4, 16, 64, $2^{3/2}$, 256, 1024, and 4096.

k	$R_{k,m}$	difference	quotient
0	0		
1	-0.24506453586713681	-0.24506453586713681	
2	-0.35810405881270413	-0.11303952294556732	2.168
3	-0.40809003951951328	-0.049985980706809152	2.2614
4	-0.42947458484537149	-0.021384545325858206	2.3375
5	-0.43838948606976558	-0.0089149012243940895	2.3987
6	-0.4420306836608825	-0.0036411975911169181	2.4483
7	-0.443493654930254	-0.0014629712693715047	2.4889
8	-0.44407363666147892	-0.00057998173122492114	2.5224
9	-0.44430103789420883	-0.00022740123272990687	2.5505

Table 16: Romberg integration of Eq.(3). Trapezoid approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
0	-0.49012907173427361		
1	-0.47114358175827148	0.018985489976002135	
2	-0.45807602022632243	0.013067561531949046	1.4529
3	-0.45085913017122975	0.007216890055092684	1.8107
4	-0.44730438729415961	0.0035547428770701384	2.0302
5	-0.44567188125199936	0.0016325060421602533	2.1775
6	-0.4449566261996255	0.00071525505237385323	2.2824
7	-0.44465361839270379	0.00030300780692171791	2.3605
8	-0.44452843912693873	0.00012517926576505189	2.4206
9	-0.44447771818135728	5.0720945581450572e-005	2.468

Table 17: Romberg integration of Eq.(3). Midpoint approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
1	-0.37909479021534803		
2	-0.41992743100397417	-0.04083264078862614	
3	-0.43542828201254646	-0.015500851008572292	2.6342
4	-0.44117018183986906	-0.0057418998273225941	2.6996
5	-0.44326520778268824	-0.0020950259428191842	2.7407
6	-0.44402212088594911	-0.00075691310326087002	2.7679
7	-0.4442937805406128	-0.0002716596546636918	2.7863
8	-0.44439083922478878	-9.705868417597685e-005	2.7989
9	-0.44442540776663464	-3.4568541845858558e-005	2.8077

Table 18: Romberg integration of Eq.(3) with $m = 1$.

k	$R_{k,m}$	difference	quotient
2	-0.44225954532334605		
3	-0.44390597951349797	-0.0016464341901519242	
4	-0.44431053134647919	-0.00040455183298121478	4.0698
5	-0.44441101559608148	-0.00010048424960229507	4.026
6	-0.44443609043642113	-2.5074840339645021e-005	4.0074
7	-0.44444235614940863	-6.2657129875076656e-006	4.0019
8	-0.44444392238543778	-1.5662360291490707e-006	4.0005
9	-0.44444431393124817	-3.9154581038536307e-007	4.0001

Table 19: Romberg integration of Eq.(3) with $m = 2$.

k	$R_{k,m}$	difference	quotient
3	-0.44445479091021528		
4	-0.44444538195747291	9.408952742373522e-006	
5	-0.4444445103459489	8.7161152401149522e-007	10.795
6	-0.44444444871653432	6.1629414571662267e-008	14.143
7	-0.44444444472040445	3.9961298714530358e-009	15.422
8	-0.44444444446411419	2.5629026678686273e-010	15.592
9	-0.44444444444651832	1.7595869206132875e-011	14.565

Table 20: Romberg integration of Eq.(3) with $m = 3$.

k	$R_{k,m}$	difference	quotient
4	-0.44444475469395672		
5	-0.44444445223851398	3.0245544274665193e-007	
6	-0.44444444460790666	7.6306073126808371e-009	39.637
7	-0.44444444445399578	1.5391088403760023e-010	49.578
8	-0.44444444444702819	6.9675931690937887e-012	22.09
9	-0.44444444444534525	1.6829315718780435e-012	4.1402

Table 21: Romberg integration of Eq.(3) with $m = 4$.

k	$R_{k,m}$	difference	quotient
5	-0.44444444743763395		
6	-0.44444444448678594	2.9508480103856982e-009	
7	-0.44444444445155273	3.5233205242235499e-011	83.752
8	-0.44444444444691761	4.6351256166587973e-012	7.6013
9	-0.44444444444531855	1.599054222367613e-012	2.8987

Table 22: Romberg integration of Eq.(3) with $m = 5$.

Integral 4

Using Romberg integration for Eq.(4), is done without any reference value. In the most general setting, we would not know anything about the function to be integrated, so all we have to deduce the error-terms by, are the differences and quotients. Here however, we do have the function, and may start by checking that at a glance, it appears to be smooth, in the sense that its first derivative exists and is continuous, and without proving it formally, it appears that the derivatives of all orders exist and are continuous, although they will grow in complexity.

Since the function is thought to be wellbehaved, let us use trapezoid approximations as they need fewer function evaluations than the midpoint rule. The result for 10 iterations is shown in table 26 in which there appears to be a convergence of the quotients to $\beta = 4$. Using this corrective factor, table 27 results in which a few of the quotients are close to the expected corrective factor of $\beta = 16$. Using this β yields table 28, where the quotients appear rather meaningless, although the last few quotients ending in 93.507 lend a tiny bit of confidence to our notion that the corrective factor should be $\beta = 2^{2m} = 64$. Since we do not have much choice, this β is used, and gives us table 29. This did not do wonders for the approximations and their differences, but having little other choice, let us continue with $\beta = 256$. If we continue this way with $\beta = 2^{2m}$, we obtain tables 30 through 34. The last value is:

$$R_{9,9} = 0.69918094691553256$$

This value is believed to be close enough[3] to the actual integral, because it agrees with the other approximations in many of its most significant decimals. A total number of $C_t(9) = 2^9 - 2 = 510$ function evaluations were used in the trapezoid approximations.

Notice the first few differences in table 26 are rather large. This is because the function is sampled over a larger range where the function decreases exponentially, and the first few refinements to I_t therefore improve greatly. Unfortunately this error propagates in the next few iterations of the Romberg process, until it finally diminishes for $m = 4$.

Integral 5

According to [5], the first many digits of π are:

$$\pi = 3.14159265358979323846264338327950288 4197\ldots$$

which means:

$$\frac{\pi}{2} \simeq 1.5707963267948966$$

And since this is the value of the integral in Eq.(5), the numeric integration below, is a curious way of approximating π.

[3]Of course this depends on the situation in which the value of the integral must be used.

k	$R_{k,m}$	difference	quotient
6	-0.44444444287291346		
7	-0.44444444443228304	-1.5593695779081429e-009	
8	-0.44444444444438258	-1.2099543589272344e-011	128.88
9	-0.44444444444444398	-6.1395333261771157e-014	197.08

Table 23: Romberg integration of Eq.(3) with $m = 6$.

k	$R_{k,m}$	difference	quotient
7	-0.4444444444383982		
8	-0.44444444444443004	-6.0318416927884755e-012	
9	-0.4444444444444442	-1.4155343563970746e-014	426.12

Table 24: Romberg integration of Eq.(3) with $m = 7$.

k	$R_{k,m}$	difference	quotient
8	-0.44444444444443593		
9	-0.4444444444444442	-8.2711615334574162e-015	

Table 25: Romberg integration of Eq.(3) with $m = 8$.

k	$R_{k,m}$	difference	quotient
0	2.5165497961937295		
1	1.4632614843433662	-1.0532883118503633	
2	0.73181898082973884	-0.73144250351362738	1.44
3	0.7666087037154502	0.034789722885711361	-21.025
4	0.70901055266718616	-0.057598151048264046	-0.60401
5	0.70124114953478156	-0.0077694031324045953	7.4135
6	0.69969535792630366	-0.0015457916084778978	5.0262
7	0.69930951924786289	-0.00038583867844077524	4.0063
8	0.69921308820186279	-9.6431046000100551e-005	4.0012
9	0.69918898212824532	-2.4106073617469193e-005	4.0003

Table 26: Romberg integration of Eq.(4). Trapezoid approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
1	1.1121653803932452		
2	0.48800481299186305	-0.62416056740138215	
3	0.77820527801068728	0.29020046501882424	-2.1508
4	0.68981116898443151	-0.088394109026255774	-3.283
5	0.6986513484906467	0.0088401795062151844	-9.9991
6	0.69918009405681103	0.00052874556616433477	16.719
7	0.69918090635504926	8.122982382285926e-007	650.93
8	0.69918094451986279	3.8164813531693653e-008	21.284
9	0.69918094677037279	2.2505100005787426e-009	16.958

Table 27: Romberg integration of Eq.(4) with $m = 1$.

k	$R_{k,m}$	difference	quotient
2	0.44639410849843758		
3	0.79755197567860892	0.35115786718017133	
4	0.68391822838268113	-0.11363374729592779	-3.0903
5	0.69924069379106102	0.015322465408379893	-7.4162
6	0.69921534376122196	-2.535002983905521e-005	-604.44
7	0.69918096050826517	-3.4383252956793342e-005	0.73728
8	0.69918094706418366	-1.3444081514712991e-008	2557.5
9	0.69918094692040678	-1.4377687929112426e-010	93.507

Table 28: Romberg integration of Eq.(4) with $m = 2$.

k	$R_{k,m}$	difference	quotient
3	0.80312591007829415		
4	0.68211451810814261	-0.12101139197015154	
5	0.69948390752770195	0.017369389419559345	-6.9669
6	0.6992149413797959	-0.00026896614790605788	-64.578
7	0.69918041474234527	-3.4526637450627007e-005	7.7901
8	0.69918094685078558	5.3210844030981974e-007	-64.886
9	0.6991809469181246	6.7339023246404395e-011	7901.9

Table 29: Romberg integration of Eq.(4) with $m = 3$.

k	$R_{k,m}$	difference	quotient
4	0.68163996362982826		
5	0.69955202278032769	0.017912059150499426	
6	0.69921388661058848	-0.00033813616973921157	-52.973
7	0.69918027934376703	-3.3607266821444881e-005	10.061
8	0.69918094893748539	6.6959371836095016e-007	-50.191
9	0.69918094691838872	-2.0190966676381095e-009	-331.63

Table 30: Romberg integration of Eq.(4) with $m = 4$.

k	$R_{k,m}$	difference	quotient
5	0.69956953212456086		
6	0.69921355607669822	-0.00035597604786263837	
7	0.69918024649208876	-3.3309584609453324e-005	10.687
8	0.6991809495920247	7.0309993593919984e-007	-47.375
9	0.69918094691641497	-2.6756097337710116e-009	-262.78

Table 31: Romberg integration of Eq.(4) with $m = 5$.

Preliminary experiments with Romberg integration of this function, suggest that a larger number of iterations should be used. Because the integral is improper (for $x = 1$), we use the midpoint rule, as the trapezoid rule would cause a division by zero for this x, and hence set the value of the function to zero at this point. This is not essential, but it does offset the initial differences and quotients slightly.

Using 20 iterations, we get table 35 where the quotients clearly converge to 1.4142, which appears to be $\sqrt{2} \simeq 1.414213562$, although this is not immediately evident from the analytic integration above. But trying this value for β, brings us to table 36, which naturally improves greatly on the differences between the approximations, as so many of the quotients were close to $\sqrt{2}$. This time, the obvious choice seems to be $2 \cdot \sqrt{2} \simeq 2.828427$, which also proves to lessen the differences, as can be seen in table 37. There, the first quotients could quite likely be converging to $5 \cdot \sqrt{2} \simeq 5.656854$, which is therefore used as the corrective factor β. The behaviour of the quotients in table 38 becomes harder to read, but the first few suggest an error-term of $8 \cdot \sqrt{2} \simeq 11.313708$, which does improve the consistency of the approximations as seen in table 39. But determining the corrective factor from the new quotients, is not too obvious. The first few quotients suggest $8 \cdot \sqrt{2}$, but the last many quotients seem to be $\sqrt{2}$. So let us try with the latter, as most of the quotients are very close to it. Table 40 shows the results having improved differences, but with the quotients bouncing wildly, thus making it difficult to make a sensible guess at what the corrective factor β should be.

Let us therefore stop the process here, and take the value of the integral to be:

$$I \simeq R_{9,5} = 1.5707963267979401$$

Naturally we could have taken the approximation using most partitions for the midpoint rule:

$$I \simeq R_{19,5} = 1.570796326794903$$

but the only reason we had 20 refinements of I_m instead of just 10, was to be able to better deduce the convergence of the quotients. If we had known in advance, the system with which the error-terms appeared, then $k = 9$ would give us sufficient precision with a handful steps of the Romberg process. This would mean that we had only used $C_m(k) = C_m(9) = 2^{10} - 1 = 1023$ function evaluations, where we in fact used $C_m(20) = 2^{21} - 1 = 2097151$ in the computations described above.

To summarize, the corrective error-factors were: $\sqrt{2}$, $2 \cdot \sqrt{2}$, $5 \cdot \sqrt{2}$, and $8 \cdot \sqrt{2}$. If we boldly try and use $\beta = 11 \cdot \sqrt{2}$ for table 40, then table 41 results, which improves the first few approximations. Still, this should be considered a lucky guess, so the results are not used.

k	$R_{k,m}$	difference	quotient
6	0.69921346914726035		
7	0.69918023835788001	-3.323078938033408e-005	
8	0.69918094976372192	7.114058419022129e-007	-46.711
9	0.6991809469157616	-2.8479603120246111e-009	-249.79

Table 32: Romberg integration of Eq.(4) with $m = 6$.

k	$R_{k,m}$	difference	quotient
7	0.69918023632950976		
8	0.69918094980714529	7.1347763552775945e-007	
9	0.69918094691558774	-2.8915575489563139e-009	-246.75

Table 33: Romberg integration of Eq.(4) with $m = 7$.

k	$R_{k,m}$	difference	quotient
8	0.69918094981803225		
9	0.69918094691554367	-2.9024885828121683e-009	

Table 34: Romberg integration of Eq.(4) with $m = 8$.

k	$R_{k,m}$	difference	quotient
0	1.1547005383792517		
1	1.2723267255127766	0.11762618713352491	
2	1.3583103474292781	0.085983621916501551	1.368
3	1.4200532525650962	0.061742905135818038	1.3926
4	1.4640335803727482	0.043980327807652042	1.4039
5	1.4952436452458082	0.031210064873059995	1.4092
6	1.5173513912216203	0.02210774597581211	1.4117
7	1.5329976001387431	0.015646208917122761	1.413
8	1.5440659598894451	0.011068359750701973	1.4136
9	1.5518941734562799	0.0078282135668348296	1.4139
10	1.55743015741164	0.0055359839553601464	1.4141
11	1.5613449016327183	0.0039147442210782923	1.4141
12	1.5641131189076813	0.0027682172749630141	1.4142
13	1.5660705706602864	0.0019574517526050883	1.4142
14	1.5674547074532705	0.0013841367929841031	1.4142
15	1.5684334432836544	0.00097873583038388112	1.4142
16	1.5691255151993762	0.00069207191572173699	1.4142
17	1.5696148843587912	0.00048936915941499848	1.4142
18	1.5699609207565492	0.00034603639775809825	1.4142
19	1.5702056054917839	0.00024468473523464596	1.4142

Table 35: Romberg integration of Eq.(5). Midpoint approximations ($m = 0$).

k	$R_{k,m}$	difference	quotient
1	1.556301462036553		
2	1.5658931737890329	0.00959171175247997	
3	1.569113811658567	0.003220637869534082	2.9782
4	1.5702114843392341	0.0010976726806670722	2.9341
5	1.5705914072127618	0.00037992287352772713	2.8892
6	1.5707242114379998	0.0001328042252379813	2.8608
7	1.5707708899402073	4.6678502207520012e-005	2.8451
8	1.5707873441368829	1.6454196675619315e-005	2.8369
9	1.5707931528355086	5.8086986256533635e-006	2.8327
10	1.5707952049697886	2.0521342800616793e-006	2.8306
11	1.5707959302329801	7.2526319150334473e-007	2.8295
12	1.5707961866025122	2.5636953204788426e-007	2.829
13	1.5707962772333732	9.0630861038221155e-008	2.8287
14	1.5707963092740822	3.2040708974889753e-008	2.8286
15	1.5707963206014759	1.1327393689697374e-008	2.8286
16	1.570796324605954	4.0044780824644022e-009	2.8287
17	1.5707963260215221	1.4155681071770232e-009	2.8289
18	1.5707963265218441	5.0032200604732679e-010	2.8293
19	1.5707963266986251	1.767810342556686e-010	2.8302

Table 36: Romberg integration of Eq.(5) with $m = 1$.

k	$R_{k,m}$	difference	quotient
2	1.5711390554963585		
3	1.5708752371185575	-0.00026381837780098394	
4	1.5708118215037297	-6.3415614827810884e-005	4.1601
5	1.5707991939461274	-1.2627557602229444e-005	5.022
6	1.5707968444855527	-2.3494605747576003e-006	5.3747
7	1.5707964192625197	-4.2522303300707165e-007	5.5252
8	1.5707963432364782	-7.6026041462284866e-008	5.5931
9	1.5707963297184937	-1.3517984509547887e-008	5.6241
10	1.570796327319371	-2.3991226782982267e-009	5.6346
11	1.5707963268926179	-4.267530773205408e-010	5.6218
12	1.5707963268156746	-7.6943340587831699e-011	5.5463
13	1.5707963268010405	-1.4634071732189113e-011	5.2578
14	1.570796326797729	-3.3115732378519169e-012	4.4191
15	1.5707963267966336	-1.0953460360951794e-012	3.0233
16	1.5707963267960758	-5.5777604757167865e-013	1.9638
17	1.5707963267957221	-3.5371705564557487e-013	1.5769
18	1.5707963267954792	-2.4291679778798425e-013	1.4561
19	1.5707963267953098	-1.6942003355779889e-013	1.4338

Table 37: Romberg integration of Eq.(5) with $m = 2$.

k	$R_{k,m}$	difference	quotient
3	1.5708185854868333		
4	1.5707982038092887	-2.0381677544589749e-005	
5	1.5707964823391922	-1.7214700964718332e-006	11.84
6	1.5707963399688607	-1.4237033152397771e-007	12.091
7	1.5707963279512969	-1.2017563832955602e-008	11.847
8	1.5707963269108556	-1.0404412886799719e-009	11.55
9	1.570796326815679	-9.5176533321250645e-011	10.932
10	1.57079632680419	-1.148903194803097e-011	8.2841
11	1.5707963268009781	-3.2118752102405779e-012	3.577
12	1.570796326799152	-1.8260948309034575e-012	1.7589
13	1.5707963267978982	-1.2538858840116518e-012	1.4563
14	1.5707963267970178	-8.8040685852774914e-013	1.4242
15	1.5707963267963985	-6.1928240313591232e-013	1.4217
16	1.5707963267959562	-4.4231285301066237e-013	1.4001
17	1.5707963267956462	-3.0997426847534371e-013	1.4269
18	1.570796326795427	-2.191580250610059e-013	1.4144
19	1.5707963267952734	-1.5365486660812167e-013	1.4263

Table 38: Romberg integration of Eq.(5) with $m = 3$.

k	$R_{k,m}$	difference	quotient
4	1.5707962276356813		
5	1.5707963154283104	8.7792629166472125e-008	
6	1.5707963261648699	1.0736559419299851e-008	8.177
7	1.5707963267860938	6.2122396116137679e-010	17.283
8	1.5707963268099761	2.3882229527316667e-011	26.012
9	1.5707963268064509	-3.525180147789797e-012	-6.7748
10	1.570796326803076	-3.3748559502555509e-012	1.0445
11	1.5707963268006666	-2.4094060080415147e-012	1.4007
12	1.5707963267989751	-1.6915358003188885e-012	1.4244
13	1.5707963267977765	-1.198596777385319e-012	1.4113
14	1.5707963267969325	-8.43991543320044e-013	1.4202
15	1.5707963267963385	-5.9396931817445875e-013	1.4209
16	1.5707963267959133	-4.2521541843143495e-013	1.3969
17	1.5707963267956162	-2.9709568138969189e-013	1.4312
18	1.5707963267954057	-2.1049828546892968e-013	1.4114
19	1.5707963267952585	-1.4721557306529576e-013	1.4299

Table 39: Romberg integration of Eq.(5) with $m = 4$.

k	$R_{k,m}$	difference	quotient
5	1.5707965273854065		
6	1.5707963520860659	-1.7529934059901109e-007	
7	1.5707963282859103	-2.3800155624087438e-008	7.3655
8	1.5707963268676348	-1.4182754970448741e-009	16.781
9	1.5707963267979401	-6.9694694460054052e-011	20.35
10	1.5707963267949281	-3.0120350658080497e-012	23.139
11	1.5707963267948497	-7.8381745538536052e-014	38.428
12	1.5707963267948912	4.1522341120980855e-014	-1.8877
13	1.5707963267948828	-8.4376949871511897e-015	-4.9211
14	1.5707963267948948	1.1990408665951691e-014	-0.7037
15	1.5707963267949046	9.7699626167013776e-015	1.2273
16	1.5707963267948868	-1.7763568394002505e-014	-0.55
17	1.570796326794899	1.2212453270876722e-014	-1.4545
18	1.5707963267948974	-1.5543122344752192e-015	-7.8571
19	1.570796326794903	5.5511151231257827e-015	-0.28

Table 40: Romberg integration of Eq.(5) with $m = 5$.

k	$R_{k,m}$	difference	quotient
6	1.5707963400432563		
7	1.5707963266508742	-1.3392382092192179e-008	
8	1.5707963267702014	1.1932721477592168e-010	-112.23
9	1.5707963267931522	2.2950752409656161e-011	5.1993
10	1.5707963267947211	1.5689671784002712e-012	14.628
11	1.5707963267948444	1.2323475573339238e-013	12.732
12	1.5707963267948941	4.9737991503207013e-014	2.4777
13	1.5707963267948821	-1.1990408665951691e-014	-4.1481
14	1.5707963267948957	1.354472090042691e-014	-0.88525
15	1.5707963267949052	9.5479180117763462e-015	1.4186
16	1.5707963267948857	-1.9539925233402755e-014	-0.48864
17	1.5707963267948999	1.4210854715202004e-014	-1.375
18	1.5707963267948974	-2.4424906541753444e-015	-5.8182
19	1.5707963267949034	5.9952043329758453e-015	-0.40741

Table 41: Romberg integration of Eq.(5) with $m = 6$.

9 Discussion

As was shown in the experiments of section 8, the Romberg process often required a significant number of *educated* guesses, some of which were less obvious than others, and some even requiring additional mathematical analysis. Full automation of this process is desirable, but requires further development of the theory behind Romberg integration.

9.1 Aitken Extrapolation

A simple attempt at automating Romberg integration with corrective factors other than 2^{2m}, is known as *Aitken* extrapolation and is briefly described in [4]. The main idea is to select one of the β_k's for β, depending on an estimate of which one is better – the simplest way is of course to select the first quotient, which has the least likelihood of being numerically imprecise due to rounding errors, but also the highest likelihood of being mathematically imprecise, because the first approximations may differ greatly in their values.

Take for example table 26, where the 6th quotient is close to 4, as are the remaining two quotients while the others are just a ragrug. In this case, selecting the first quotient would lead us to $\beta = 1.44$ because the approximations I_t improve greatly in the first few iterations, thus creating misleading quotients. At any rate, Aitken extrapolation is not sufficient in general to automate Romberg integration.

9.2 Improper Corrective Factor

Another interesting topic, is the effect of an improper choice of corrective factor β. This is also briefly discussed in [4], but a more detailed study of how it affects the rest of the process, and in particular its effect on automatically finding the true factors β for future iterations, is of interest. Furthermore, the *degree of incorrectness* within which we may still save the rest of the process, is important when using a β that has been automatically discovered.

9.3 Multiple Convergence Limits

When the series of quotients appear to have several sub-series that exhibit convergence towards different limits, then there are several seemingly valid choices for β. Take for example table 9 where we chose a corrective factor of $\beta = 16$. Although this resulted in table 10 which shows clear convergence to 2.5198 (as was also the case in table 8), this choice for β is already suggested by the last few quotients in table 9.

The question is then, in case of multiple convergences for the quotients as illustrated there, which one is the better choice? One argument would be that we should take the last of the limits, in the case of table 9 this would be 2.5198. The reason for this, is that we then as quickly as possible refine our best approximation $I_t(2^k)$ for the highest k, and may therefore yield more precise results with fewer iterations of the Romberg process. Conversely, the error introduced by selecting a wrong factor for the first approximations, will propagate through the Romberg process, which arguably means that we should choose the first appearing convergence limit as the corrective factor. For this example however, the difference in the first approximations of table 9 are already very low, so an erroneous choice of β is not catastrophic.

9.4 Number of Iterations

The number of iterations that are required, is chosen experimentally in the above. In a more robust setting, some theory on the required number of iterations of the Richardson process, would be desirable. It is naturally possible to set as termination criterion, in form of a limit on the difference $N_m(k) - N_m(k-1)$ that must be obtained, but without proper theoretical foundation, we can not be sure that the algorithm will terminate, with this criterion alone.[4]

9.5 Floating Point Rounding Errors

After a number of iterations of the Romberg process, the differences become very small and the quotients (often) become very large, e.g. 2^{2m}. The error estimates that are added to the approximations are therefore smaller still, and are thus highly susceptible to floating point rounding errors.

Two suggestions may alleviate this. The first considers extrapolation of only the errors and not the approximations. In the above Eq.(16) for Richardson extrapolation, new approximations $N_m(k)$ are generated from existing ones (that is, from $N_{m-1}(k)$). Again, this means that as the precision of the approximations increase, the differences and quotients will degenerate with rounding errors. Extrapolation of only the errors, so that we generate $E_m(k)$ instead of $N_m(k)$, could prove useful in this regard.

The other suggestion, which could be used by itself or in conjunction to extrapolation of only the errors, is to use rewrites of the extrapolation formulae, that improve numeric precision on finite and discrete computers. The first and most obvious attempt would be to use logarithms, possibly with normalization-factors. This choice stems from the nature of logarithms, which is to continuously *condense* or *compress* a range onto a (logarithmic) fraction of that range, meaning that instead of e.g. a difference between approximations of $d = 1e-17$, we would have $\log(d) = -17$. Naturally, some research must be conducted to find the best way to employ such a rewrite – and whether it is useful at all.

9.6 Heuristics

Another even more experimental suggestion, would be to use a heuristic optimization approach. What we wish to find, is the corrective factor β, which

[4]In numerical algorithms with termination limits, it is customary to simply set a maximum count also, to avoid this problem altogether.

minimizes the differences in the next iteration of the Richardson process. Summing all of these differences to obtain a single fitness measure, we find that it equals the last $N_m(k)$ minus the first $N_m(k')$ for the first $k' < k$ in that iteration. Finding the β which minimizes $N_m(k) - N_m(k')$ should leave us with a good guess of what β should be for the next iteration of the Richardson process.

References

[1] David Kincaid and Ward Cheney: Numerical Analysis, Third Edition, Brooks/Cole, 2002, ISBN 0-534-38905-8.

[2] Michael L. Overton: Numerical Computing with IEEE Floating Point Arithmetic, SIAM, 2001, ISBN 0-89871-482-6.

[3] Ole Østerby: Numerical Analysis, Supplementary Notes, University of Aarhus, Autumn 2002.

[4] Ole Østerby: Pearls of Programming, Romberg integration, extrapolation and error estimation, University of Aarhus, February 10 2000.

[5] Sloane, N. J. A., Sequence A000796 (M2218) in "An On-Line Version of the Encyclopedia of Integer Sequences.", `http://www.research.att.com/~njas/sequences/eisonline.html`.