# Numerieke Integratie - Oefening 1

Ruben Van Assche

December 16, 2016

## 1 Opgave

De opdracht was de approximatie van ln(2) d.m.v. de trapezium regel. We weten dat:

$$ln(x) = \int_a^b (\frac{1}{t} dt)$$

En dus over het interval [1,2]

$$ln(2) = \int_1^2 (\frac{1}{t} dt)$$

Hierop laten we de trapeziumregel los welke telkens opnieuw wordt berekend met $n = 2^k$ waarbij $k = 1, ...$ dit levert de functie $T(k)$ op. Hierbij wordt $k$ telkens verhoogd tot

$$\left| \frac{T(k) - T(k+1)}{T(k+1)} \right| \leq 2^{-40}$$

.

## 2 Trapeziumregel

We berekenen de integraal d.m.v. de trapeziumregel, we integreren over het interval [1,2] en zullen het interval opdelen in $2^k$ subintervallen waarbij $k = 1, 2, ....$ We noteren $T(k)$ voor de benadering.
Om de juiste k te vinden voor de juiste benadering zullen we vereisen dat de relatieve fout $\leq 2^{-40}$.Dus:

$$\frac{T(k) - T(k+1)}{T(K+1)} \leq 2^{-40}$$

Via numerical recipes vinden we algauw een methode om d.m.v. de trapeziumregel de integraal uit te rekenen. Hierbij is a de ondergrens en b de bovengrens van de integraal.

[0]http://e-maxx.ru/bookz/files/numerical_recipes.pdf

```
Doub next() {
    Doub x,tnm,sum,del;
    Int it,j;
    n++;
    if (n == 1) {
       return (s=0.5*(b-a)*(func(a)+func(b)));
    } else {
       for (it=1,j=1;j<n-1;j++) it <<= 1;
       tnm=it;
       del=(b-a)/tnm;
       x=a+0.5*del;
       for (sum=0.0,j=0;j<it;j++,x+=del) sum += func(x);
       s=0.5*(s+(b-a)*sum/tnm);
       return s;
    }
}
```

Wanneer we deze functie herschrijven dat ze ietwat leesbaarder is bekomen we volgende code:

```
double trapezium(std::function<double(double)> f, int k, double a, double b) {
    int n = pow(2, k);
    double h = (b - a) / n;

    double sum = 0.0;
    double x = a;

    for (int i = 1; i < n; i++) {
        x += h;
        sum += h * f(x);
    }

    sum += (h / 2) * f(a + h);
    sum += (h / 2) * f(b - h);

    return sum;
}
```

# 3 Efficiëntie

We kunnen nu de approximatie van de integraal berekenen. Maar vermits onze deelintervallen telkens verdubbelen wordt het aantal keer dat we f(x) berekenen drastisch groter. Het probleem met de huidige vorm van de trapeziumregel is dat bij elke vergroting van k we een hoop punten opnieuw moeten berekenen. We roepen dus meerdere malen f(x) op met dezelfde x-coordinaat. Een mogelijkheid bestaat erin om deze x en bijhorende f(x) in een map op te slaan en later telkens op te roepen maar het zoeken van de x-waarde in een map kost aanzienlijk meer tijd dan het berekenen van $f(x) = 1/x$. Daarom opteren we voor een versie die de punten ertussen telkens extra berekent.

Wanneer we de trapeziumregel uitschrijven voor a = 1 en b = 2 en h = $\frac{a-b}{2}$(normaal gedeeld door n maar voor dit voorbeeld houden we hier de n constant op 2) met:

**n=2**

$h(\frac{1}{2}f(1) + \frac{1}{2}f(2)) + h(f(1.5))$

**n=4**

$\frac{h}{2}(\frac{1}{2}f(1) + \frac{1}{2}f(2)) + \frac{h}{2}(f(1.5)) + \frac{h}{2}(f(1.25) + f(1.75))$

**n=8**

$\frac{h}{4}(\frac{1}{2}f(1) + \frac{1}{2}f(2)) + \frac{h}{4}(f(1.5)) + \frac{h}{4}(f(1.25) + f(1.75)) + \frac{h}{4}(f(1.125) + f(1.375) + f(1.65) + f(1.875))$

Wat opvalt is dat bij elke vergroting van n de volledige uitdrukking van de vorige n opnieuw voorkomt maar dan gedeeld door 2. Vervolgens worden er nog enkele termen f(x) bijgeteld. Dit vanaf n = 4.

Het aantal termen dat extra berekend worden hangt af van n. Voor n zijn er namelijk $2^{log_2(n)-1}$. De eerste 2 termen zullen altijd f(a+h) en f(b-h). De termen daarop zullen altijd in koppel de regel volgen namelijk f(a+h+2*ih) en f(b-h-2ih) voor i = 1 ... $\frac{2^{log_2(n)-1}-2}{2}$.

Voor n = 8 bekomen we dan f(1.125), f(1.375), f(1.65), f(1.875).

Voor elk niveau n tellen we deze termen op en vermenigvuldigen we ze met $\frac{b-a}{n}$. Wanneer we dan (zoals eerder aangehaald) de uitdrukking van het niveau n/2 erbij optellen en delen door 2, bekomen we de trapeziumregel voor niveau n.

Vermits we bij deze opgave telkens de n verhogen(en dus telkens al het vooraande niveau van de trapeziumregel hebben berekend) zorgt deze techniek ervoor dat het algoritme een stuk sneller zal werken.

# 4 Uitkomsten

| k | n | uitkomst | fout |
|---|---|---|---|
| 2 | 2 | 0.70833333333333325932 | / |
| 3 | 4 | 0.69702380952380948997 | 0.016225448334756562702 |
| 4 | 8 | 0.69412185037185025749 | 0.0041807632916390901484 |
| 5 | 16 | 0.69339120220752681334 | 0.0010537315183655395368 |
| 6 | 32 | 0.69320820826924878233 | 0.00026398120520660999736 |
| 7 | 64 | 0.69316243888340334234 | 6.6029812462384222392e-05 |
| 8 | 256 | 0.69315099522810807997 | 1.6509613885061783162e-05 |
| 9 | 512 | 0.69314741897841058993 | 4.1275385793745183181e-06 |
| 10 | 1024 | 0.69314724016458284517 | 1.0318930903303908621e-06 |
| 11 | 2048 | 0.69314719546110592496 | 2.5797380034623445163e-07 |
| 12 | 4096 | 0.69314718428523591776 | 6.4493483076803758563e-08 |
| 13 | 8192 | 0.69314718149126797186 | 1.6123372150363317769e-08 |
| 14 | 16384 | 0.69314718079277626295 | 4.030843694523827 41e-09 |
| 15 | 32768 | 0.69314718061815094874 | 1.0077105242181248858e-09 |
| 16 | 65536 | 0.69314718057449753452 | 2.5193107480149093119e-10 |
| 17 | 131072 | 0.69314718056358781695 | 6.2978564207053126936e-11 |
| 18 | 262144 | 0.69314718056086133124 | 3.9334874040291391555e-12 |
| 19 | 524288 | 0.6931471805601696623 | 9.9786735593802382027e-13 |

Wanneer we dus k = 20 en dus n = 1048576 hebben we een fout 2.5659446295555767756e-13 $\leq 2^{-40}$

# 5 Code

```
#include <functional>
#include <iostream>
#include <math.h>
#include <map>
#include <vector>
#include <limits>
#include <list>



double trapezium(std::function<double(double)> f, int k, double a, double b) {
    int n = pow(2, k);
    double h = (b - a) / n;

    double sum = 0.0;
    double x = a;

    for (int i = 1; i < n; i++) {
        x += h;
```

```
        sum += h * f(x);
    }

    sum += (h / 2) * f(a + h);
    sum += (h / 2) * f(b - h);

    return sum;
}


double calculateSum(std::function<double(double)> f, int k, double a, double b)
    if (k == 1) {
        double n = pow(2, k);
        double h = (b - a) / n;

        return h * f(a + h);
    }

    double nprev = pow(2, k - 1);
    double n = nprev * 2;
    double h = (b - a) / n;

    double incremental = h * 2;
    double toCalc = (nprev - 2) / 2;

    double sum;
    double posa = a + h;
    double posb = b - h;
    sum += h * f(posa);
    sum += h * f(posb);
    for (int i = 0; i < toCalc; i++) {
        posa += incremental;
        posb -= incremental;
        sum += h * f(posa);
        sum += h * f(posb);
    }

    return sum;
}

double trapeziume(std::function<double(double)> f, int k, double a, double b) {
    double sum = 0.0;

    for (int i = 1; i <= k; i++) {
        double n = pow(2, i);
```

```
        if (i == 1) {
            double h = (b - a) / n;

            sum += (h/2) * (f(a) + f(b));
            sum += calculateSum(f, i, a, b);
        } else {
            sum /= 2;
            sum += calculateSum(f, i, a, b);
        }
    }


    return sum;
}

// Calculates the trapeoid rule till the relative error is smaller then maxError
int trapeziumewitherror(std::function<double(double)> f, double a, double b, dou
    int k = 1;
    double prevsum = 0.0;
    double sum = 0.0;

    while(true){
        // Set prevsum equal to the previous sum
        prevsum = sum;

        double n = pow(2, k);

        if (k == 1) {
            double h = (b - a) / n;

            sum += (h/2) * (f(a) + f(b));
            sum += calculateSum(f, k, a, b);
        } else {
            sum /= 2;
            sum += calculateSum(f, k, a, b);
        }

        // Calculate error
        double error = (prevsum-sum)/(sum);

        if(fabs(error) <= maxError){
            std::cout << "Found integration using " << n << " intervals(n = " <<
            std::cout << ") with an error of " << error << " and solution: " <<
            break;
        }
```

```
        // Raise intervals
        std::cout << "Using " << n << " intervals: " << sum << " error: " << err
        k++;
    }

    return k;+
}

int main() {
    std::cout.precision(20);

    int a = 1;
    int b = 2;
    std::function<double(double)> f = [](double x) {
        return 1.0 / x;
    };

    int k = trapeziumewitherror(f, 1, 2, pow(2, -40));

    std::cout << "Calculate with n = " << k << " using basic trapezoid rule" <<
    std::cout << trapezium(f, k, 1, 2);
}
```