Info I Übung 08.12.22

Exercise 4

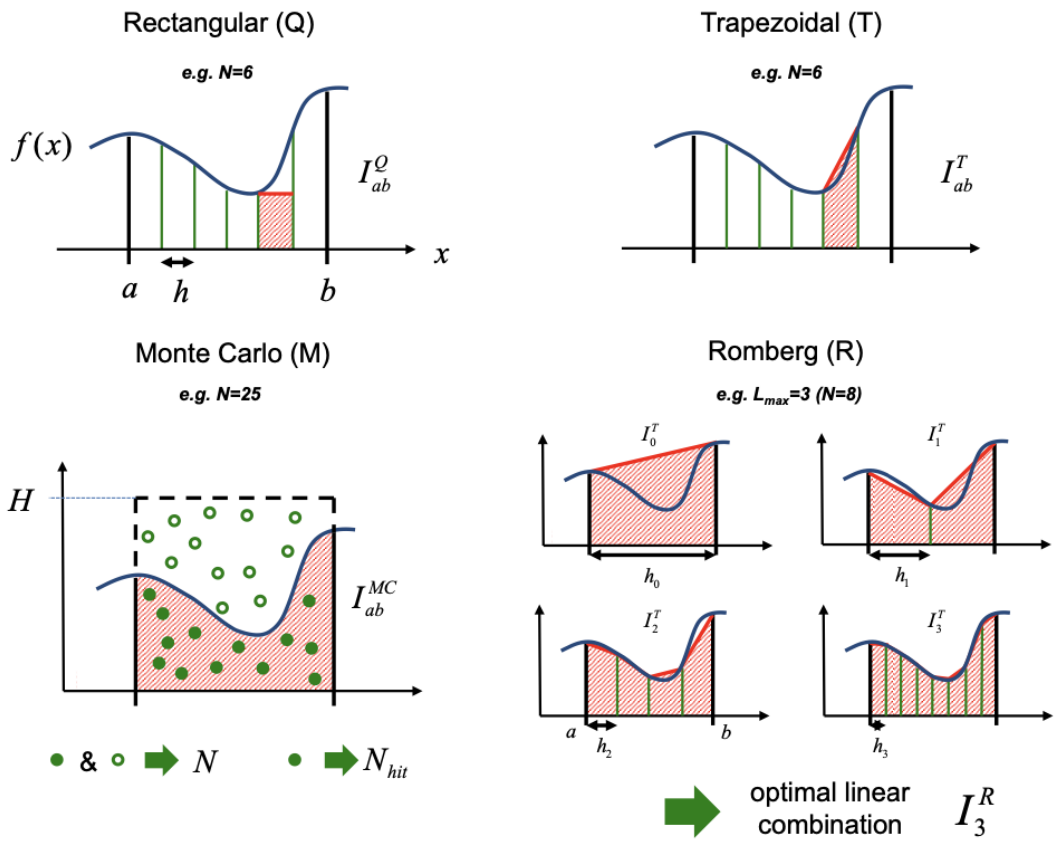
- Abgabe ist der 12.12.22
- Ich denke ihr seid alle mit ex4 fertig, wie liefs?

Exercise 5

- Abage am 19.12.22
- Der Termin für die Abgabe von Exercise 5 hat sich nicht verändert.
- Sandbox ist optional
    - Sie ist also nicht wichtig für die Serie oder die Prüfung
    - Enthält aber sehr interessante Elemente von C++ und falls ihr euch etwas dafür interessiert empfehle ich euch die Sandbox zumindest durchzulesen.

Quadratur

- In dieser Task geht es darum, die Wärmekapazität über die Temperatur zu integrieren. Analytische Integration ist in manchen fällen nicht möglich, daher müssen wir das Integral über verschiedene Arten abschätzen. Hier werden 4 verschieden Arten der Approximation betrachtet.
    - Das Approximieren einer Kurve wird Quadratur gennant. Da man versucht die Fläche unter der Kurve in eine geometrische form umzuwandeln, deren Fläche man leicht errechnen kann, wie zum Beispiel mit einem Quadrat.



- Wie hier zu erkennen ist, gibt es unterscheidliche Ansätze. Drei der vier werten die zu integrierende Funktion an verschiedenen Punkten aus und berechnen daraus eine Teilfläche die dann zu der Gesamtfläche addiert wird. Die Monte Carlo Quadratur funktioniert da etwas anders. Es werden beleibing viele Punkte, zufällig, generiert und die Zahl an Punkten unter der Kurve im Verhältnis zur gesamt Zahl Punkte wird für die Approximation der Fläche Benutzt
  - Für eine visueller Erklärung empfehle ich diese video:

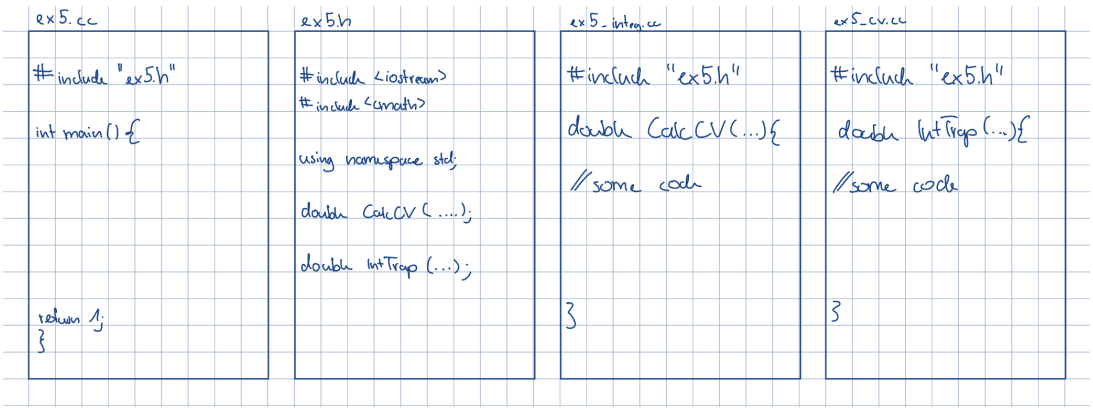
https://www.youtube.com/watch?v=8276ZswRw7M

Tipps fürs Programmieren

Wie linked man files?

- Ziel ist es, das ihr ein porgramm schreibt und die einzelnen Funktionen, Fähigkeiten, eures Programms in verschiedenen Files speichert.
- In eurem ex5.cc file müsst ihr also nur die Funktionen aufrufen und nicht definieren. Aber wo sind sie dann definiert?

  ⇒ Dafür müsst ihr eure h-file, also ex5.h, importieren mit dem bekannten #include, aber dieses mal wird der filename nicht von <> umklammert sondern von “ “.
- Im h-file importiert ihr alle Bibliotheken die ihr verwenden wollt. Importiert ihr diese nochmals im .cc file gibt es einen Fehler. Denn ihr könnt durch den import des h-files auf alles was dort definiert und importiert wurde zugreifen. Ausserdem wollt ihr hier die Header eurer Funktionen deklarieren. Die Funktionen definiert ihr aber in dem dazu gehörigen file.
  - dort könnt ihr auch file übergreifende, globale Variablen defninieren.
- Im file für die Funktion selbst importiert ihr wieder das h-file, so weiss das h-file schon die definition zu den Funktionen die ihr dort deklariert habt.



Exercise 7

- Abgabe 02.01.23
- Nur eine Woche Zeit!
- Ihr müsst dazu VMD/NAMD benutzten, diese Programme hat es auf den Computern der ETH. Es ist möglich VMD auf euren Computern zu installieren, ich würde euch aber empfehlen es an vor ort zu machen.
- Falls ihr plant es von euren Computer aus zu machen, installirt VMD bitte eine Woche vor beginn der Exerise 7, damit ich eucu ggf. helfen kann.

- Für Hilfe bei der Installation siehe 0C.4.1: https://ethz.ch/content/dam/ethz/special-interest/chab/physical-chemistry/csms-dam/doc/InfoI_doc-res/infoI_cmp_connect_HS22.pdf

4 Algorithms and programming (S2021.4)

Die Funktion $sin^2(x)$ kann als Taylorreihe entwickelt werden zu

$$\begin{aligned} sin^2(x) &= \sum_{n=1}^{\infty} (-1)^{n+1} \frac{2^{2n-1}}{(2n)!} x^{2n} \\ &= \frac{2}{2!}x^2 - \frac{2^3}{4!}x^4 + \frac{2^5}{6!}x^6 - \frac{2^7}{8!}x^8 + ... \end{aligned}$$

Schreiben Sie eine **effiziente C++ Funktion** `sin_sqr`, welche $sin^2(x)$ auf Basis der oben stehenden Entwicklung zurückgibt. **Brechen Sie die Serie ab**, wenn der zuletzt hingefügte Term im Betrag kleiner ist als ein gegebener Grenzwert `eps` > 0. Die Funktionsdeklaration lautet

```
double sin_sqr (double x, double eps);
```

$$\sin^2(x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{2^{2n-1}}{(2n)!} x^{2n} = \frac{2}{2!}x^2 - \frac{2^3}{4!}x^4 + \frac{2^5}{6!}x^6 - \frac{2^7}{8!}x^8 + \dots$$

mit header: double sin_sqr (double x, double eps);

① einzelne Terme finden

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{2^{2n-1}}{(2n)!} x^{2n} = \frac{2}{2!}x^2 - \frac{2^3}{4!}x^4 + \dots \quad (2^3 = 8)$$

es wird immer um x^2 erhöht
=> Hilfsvariable

② Variablen für jeden Term definieren für n=1, da die Summe bei n=1 anfängt:

```
double x2 = x * x;
double pow_x = x^2;
int sign = 1;
double pow2 = 2;
double fac2 = 2;
int n = 1;
double sum = term = x2;
```

▽ Namen wie 2pow oder 2fac akzeptiert der Compiler nicht ▽

Brauchen variablen die das Ergebnis und den aktuellen Term speichern.

③ Summe als loop darstellen, for loop würde auch gehen, dann muss man aber mit "n" aufpassen.
Loop muss nur den Schritt von n nach n+1 ausführen.
Wie ändern sich die oberen Variablen in jedem Schritt?

```
while ( term * term > eps * eps ){ // damit umgeht man das nervige Vorzeichen aus der Musterlösung
  n++; // wir berechnen jetzt für n = 2
  sign *= -1;
  powx *= x2;
  pow2 *= 4; // 4 = 2*2
  fac2 *= 2*n * (2n-1); // um von 2! nach 4! müssen wir mit 3 und 4 multiplizieren.
  term = sign * pow2 * powx / fac2;
  sum += term;
}
```

4 Algorithms and programming (S2021.4)

The key to efficiency in this type of problems is to construct each new term from the previous one by means of simple operations. Looking at the equation, we observe that to construct a term, we need $(-1)^{n+1}$, 2^{2n-1}, $(2n)!$ and x^{2n}. So, we define 4 variables, one for each of them. Their initial values, corresponding to the $n = 1$ term, should be set to $+1$, 2, 2 and x^2, respectively. So, we initialize the 4 variables to these values. We also define a variable for storing the current term and one for storing the sum. We initialize the current term and the sum to the value of the first term to add at $n = 1$, *i.e.* x^2. Now we can start a `while`-loop, testing the size of the last added term (at first, it will be the term $n = 1$ that we already included at initialization; then it will be the successive additional terms). In the loop body, we need to increment n and then to update the values of the 4 variables accordingly, *i.e.* from their values appropriate for term $n-1$ to new values appropriate for term n. For the 4 variables, this gives $\cdot(-1)$ for $(-1)^{n+1}$, $\cdot 2^2$ for 2^{2n-1}, $\cdot 2n(2n-1)$ for $(2n)!$, and $\cdot x^2$ for x^{2n}. We use these updated variables to construct the new term n, we add this new term n to the sum, and we are ready for the next iteration. A possible C++ code for the function `sin_sqr` thus reads

```
double sin_sqr (double x, double eps) {
    double x2 = x * x;           // for the constant x^2
    int sgn = +1;                // for (-1)^n
    int pow2 = 2;                // for 2^(2n-1)
    int fac2 = 2;                // for (2n)!
    double powx = x2;            // for x^(2n)
    double sum = term = x2;      // first term with n = 1
    int n = 1;
    while ( term < -eps || eps < term ) {   // magnitude of last term larger than eps
        n++;
        sgn *= -1;
        pow2 *= 4;
        fac2 *= 2*n * (2*n-1);
        powx *= x2;
        term = sgn * pow2 * powx / fac2;
        sum += term;
    }
    return sum;
}
```

Important remark: A common student mistake in this exercise is to use variable names like `2_pow` or `2_fac`, which start by a digit. Keep in mind that this is **not allowed** in C++.