Numerische Integration

Rechteck-, Trapez- und Simpsonregel mit einer Implementierung in Python

*Marcel Kropp**

Universität Innsbruck
Jänner 2015

*Dieses Abschlussprojekt entstand im WS 2014/15 im Zuge des Kurses „Einführung in das mathematische Arbeiten, mathematische Software und Programmieren“ an der Universität Innsbruck unter der Leitung von Dr. Stefan Rainer.

Inhaltsverzeichnis

1. Motivation **3**

2. Verfahren **3**
2.1. Die Rechteckregel 3
2.2. Die Trapezregel 5
2.3. Die Simpsonregel 6
2.4. Konvergenzverhalten & Verfahrensfehler 8
2.4.1. Eine erste Intuition 8
2.4.2. Fehlerabschätzung 10

3. Implementierung in Python **11**
3.1. Beispiel: Die (Standard)-Normalverteilung 11

4. Zusammenfassung und Ausblick **15**

A. Appendix **16**
A.1. Python-Code 16
A.1.1. numtint.py 16
A.1.2. konvergenz.py 20

Literatur & Quellen **25**

Abbildungsverzeichnis

1. Veranschaulichung der Rechteckregel. 4
2. Veranschaulichung der Trapezregel. 5
3. Veranschaulichung der Simpsonregel. 7
4. Vergleich der Verfahrensfehler. 9
5. Funktionsplotter des „numint.py“-scripts. 14

Tabellenverzeichnis

1. Gegenüberstellung der Rechteck-, Trapez- und Simpson-Regel. . 8

1. Motivation

Wenn es darum geht das bestimmte Integral

$$I := \int_a^b f(x)dx \tag{1}$$

zu berechnen, so treten häufig Fälle auf, in denen das Integral analytisch nicht integrierbar ist (eine Stammfunktion also nicht in geschlossener Form hingeschrieben werden kann)[1] oder die Berechnung der exakten Lösung zu aufwendig ist. In diesen Fällen wird auf die *numerische* Integration (auch numerische *Quadratur* genannt) zurückgegriffen, also, die näherungsweise Bestimmung von bestimmten Integralen.

Im Zuge dieser Arbeit befassen wir uns ausschließlich mit der univariaten Integration. Es sei also im folgenden f eine auf dem abgeschlossenen Intervall $[a, b]$ stetige Funktion, also $f : [a, b] \subset \mathbb{R} \to \mathbb{R}$. Weiters wählen wir eine äquidistante Zerlegung $Z = \{x_0, \ldots, x_n\}$ des Intervalls $I = [a, b]$ in n Teilintervalle $[x_{i-1}, x_i]$ der Länge $h = \frac{b-a}{n}$[2] mit $x_0 = a, x_1 = a + h, x_2 = a + 2h, \ldots, x_n = a + nh = b$.

Die grundlegendsten Methoden zur numerischen Integralberechnung sind

- ▷ die Rechteck-Regel
- ▷ die Trapez-Regel
- ▷ die Simpson-Regel,

welche wir nun im folgenden näher beleuchten. Das gemeinsame Ziel dieser Methoden ist es, das kontinuierliche Problem aus (1) in ein diskretes überzuführen, das mit angemessenem Rechenaufwand ausgewertet werden kann.
Dabei gibt es zur Verbesserung der Rechengenauigkeit prinzipiell die Möglichkeit entweder (i) zu einer vorgegebenen Unterteilung des Intervalls $[a, b]$ Interpolationspolynome höherer Ordnung zur Approximation der Funktion über den jeweiligen Teilintervallen zu verwenden, oder (ii) zu einem vorgegebenen Interpolationspolynom die Länge h der Teilintervalle zu verkleinern.

2. Verfahren

2.1. Die Rechteckregel

Beim Reckteck-Verfahren wird, wie in Abbildung 1 ersichtlich, die Funktion $f(x)$ (der Integrand) in jedem Teilintervall $[x_{i-1}, x_i]$ durch eine konstante

[1] Klassische Beispiele wären die Integrale $\int e^{-x^2} \mathrm{d}x$ oder $\int \frac{sin(x)}{x} \mathrm{d}x$.
[2] Die sogen. *Schrittweite*.

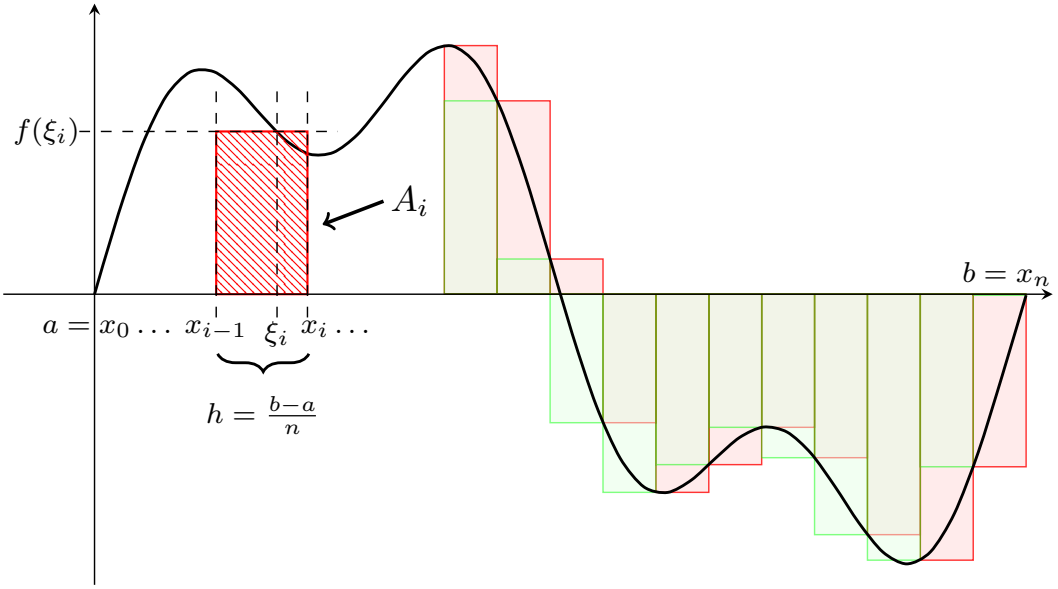

Abbildung 1: Veranschaulichung der Rechteckregel.

Funktion $f(\xi_i), \xi_i \in [x_{i-1}, x_i]$ ersetzt. In jedem Teilintervall wird dazu eine Stützstelle gewählt, d.h. es gilt $\forall i = 1, \ldots, n : \xi_i \in [x_{i-1}, x_i]$. Dadurch wird die i-te Fläche unter der Kurve durch $A_i = f(\xi_i) \cdot h$ approximiert. Das Integral I über das Gesamtintervall wird folglich durch

$$I_R = \sum_{i=1}^{n} A_i = \sum_{i=1}^{n} f(\xi_i) \cdot h = h \sum_{i=1}^{n} f(\xi_i)$$

approximiert, d.h. wir können schreiben

$$I = \int_a^b f(x)\mathrm{d}x \quad \approx \quad I_R = h \cdot (f(\xi_1) + f(\xi_2) + \cdots + f(\xi_n)).$$

Die Wahl der Stützstellen ξ in den Teilintervallen ist prinzipiell beliebig. Eine naheliegende Möglichkeit wäre, die Teilinterval*mitten* als Stützstellen zu wählen, d.h., $\xi_i = \frac{1}{2}(x_i + x_{i-1})$. Für die Approximation des Integrals ergibt sich dann entsprechend

$$I_{R_M} = h \sum_{i=1}^{n} f(\frac{1}{2}(x_{i-1} + x_i)).$$

Werden hingegen jeweils die linken *Teil*intervallgrenzen als Stützstellen verwendet, d.h., $\xi_i = x_{i-1}, \forall i = 1, \ldots, n$,[3] so ergibt sich

$$I_{R_L} = h \sum_{i=1}^{n} f(x_{i-1})$$

[3]Die Stützstellen lauten dann $\xi = \{\xi_1 = x_0, \xi_2 = x_1, \ldots, \xi_n = x_{n-1}\}$.

und man spricht von der Näherung von I nach der *Linksregel*.
Entsprechend ergibt sich, wenn die rechten Intervallgrenzen als Stützstellen verwendet werden, d.h., $\xi_i = x_i, \forall i = 1, \ldots, n$,[4] die Näherung nach der *Rechtsregel*

$$I_{R_R} = h \sum_{i=1}^{n} f(x_i).$$

Abbildung 1 zeigt in der 2. Hälfte des Funktionsgraphen exemplarisch den Unterschied zwischen der Linksregel (rote Flächen) und der Rechtsregel (hellgrüne Flächen). Besonders bei der computerunterstützten Implementierung wird auf die Links- bzw. Rechtsregel zurückgegriffen, da ein händisches Eintippen der Stützstellen nicht praktikabel ist. Näheres dazu besprechen wir in Abschnitt 3.

2.2. Die Trapezregel

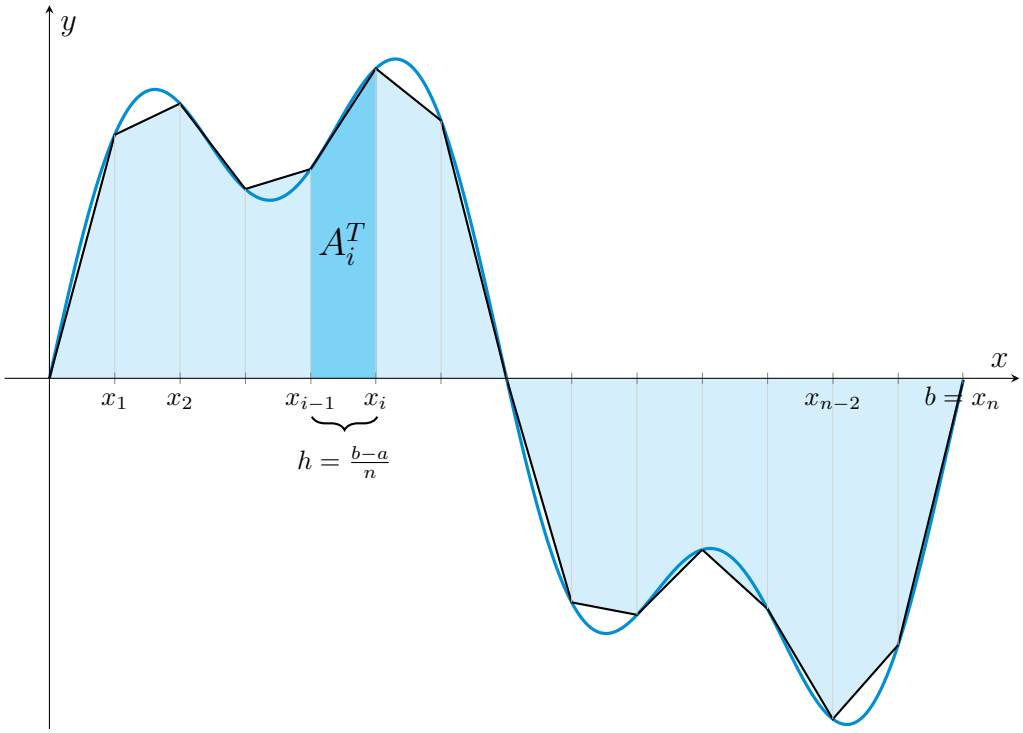

Abbildung 2: Veranschaulichung der Trapezregel.

Verglichen mit der Rechtecks-Methode liefert die Trapez-Regel genauere Näherungswerte. Dabei wird die Kurve $y = f(x)$, gemäß Abbildung 2, in jedem Teilintervall $[x_{i-1}, x_i]$ durch eine Sehne ersetzt, welche durch die Punkte $(x_{i-1}, f(x_{i-1})), (x_i, f(x_i))$ läuft. Dadurch wird die i-te Fläche unter der Kurve

[4] Die Stützstellen lauten dann $\xi = \{\xi_1 = x_1, \xi_2 = x_2, \ldots, \xi_n = x_n\}$.

entsprechend der Flächenformel für ein Trapez durch $A_i^T = \frac{1}{2} \cdot h \cdot (f(x_{i-1}) + f(x_i))$ approximiert, wobei h wieder der äquidistanten Schrittweite entspricht, welche durch $h = \frac{b-a}{n}$ bestimmt ist. Das Integral I über das Gesamtintervall $[a, b]$ wird dann durch die folgende Summe von Trapezflächen

$$\begin{aligned} I_T &= \sum_{i=1}^{n} A_i^T \\ &= \sum_{i=1}^{n} \frac{1}{2} \cdot h \cdot (f(x_{i-1}) + f(x_i)) = \frac{1}{2} h \sum_{i=1}^{n} (f(x_{i-1}) + f(x_i)) = \\ &= \frac{1}{2} h \cdot ((f(x_0) + f(x_1)) + (f(x_1) + f(x_2)) + \cdots + (f(x_{n-1}) + f(x_n))) = \\ &= \frac{1}{2} h \cdot (f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)) \end{aligned}$$

approximiert, d.h., $I = \int_a^b f(x)\mathrm{d}x \quad \approx \quad I_T$.

2.3. Die Simpsonregel

Die Simpson-Regel geht noch einen Schritt weiter als die Trapez-Regel und approximiert die zu $y = f(x)$ gehörige Kurve anstatt durch Sehnen durch Parabelbögen (Polynome 2. Grades). Da die einzelnen Parabelbögen jeweils durch drei Punkte eindeutig bestimmt sind, ist bei dieser Methode eine gerade Anzahl von Teilintervallen notwendig (bzw. eine ungerade Anzahl von Punkten), wobei die Schrittlänge, wie gehabt, $h = \frac{b-a}{n}$ beträgt.

In jedem Doppelstreifen $[x_{i-1}, x_{i+1}]$, in dessen Intervallmitte der Punkt x_i mit Abstand h zu x_{i-1} bzw. x_{i+1} liegt, wird ein Interpolationspolynom 2. Grades der Form $p_i(x) = a_2x^2 + a_1x + a_0$ angesetzt, welches demnach durch die Punkte $P = (x_{i-1}, f(x_{i-1}))$, $Q = (x_i, f(x_i))$ und $R = (x_{i+1}, f(x_{i+1}))$ verlaufen muss. Der Flächeninhalt des i-ten Flächenstücks unter dem i-ten Parabelbogen wird durch Integration

$$A_i^S = \int_{x_{i-1}}^{x_{i+1}} p_i(x)\mathrm{d}x$$

ermittelt. Um nun eine allgemeine Formel für dieses Integral zu erhalten, können wir festhalten, dass A_i^S auch nach Verschiebung des mittleren Punktes auf die y-Achse (sodass $x_i = 0$) unverändert bleibt. Wenn wir annehmen, dass *nach* Verschiebung $x_i = 0$ gilt, so folgt daraus, dass $x_{i-1} = -h$ und $x_{i+1} = h$, woraus wir die folgenden drei Gleichungen ableiten können:

$$\begin{aligned} I &: f(x_{i+1}) = p_i(-h) = a_2h^2 - a_1h + a_0 \\ II &: f(x_i) = p_i(0) = a_0 \\ III &: f(x_{i+1}) = p_i(h) = a_2h^2 + a_1h + a_0 \end{aligned}$$

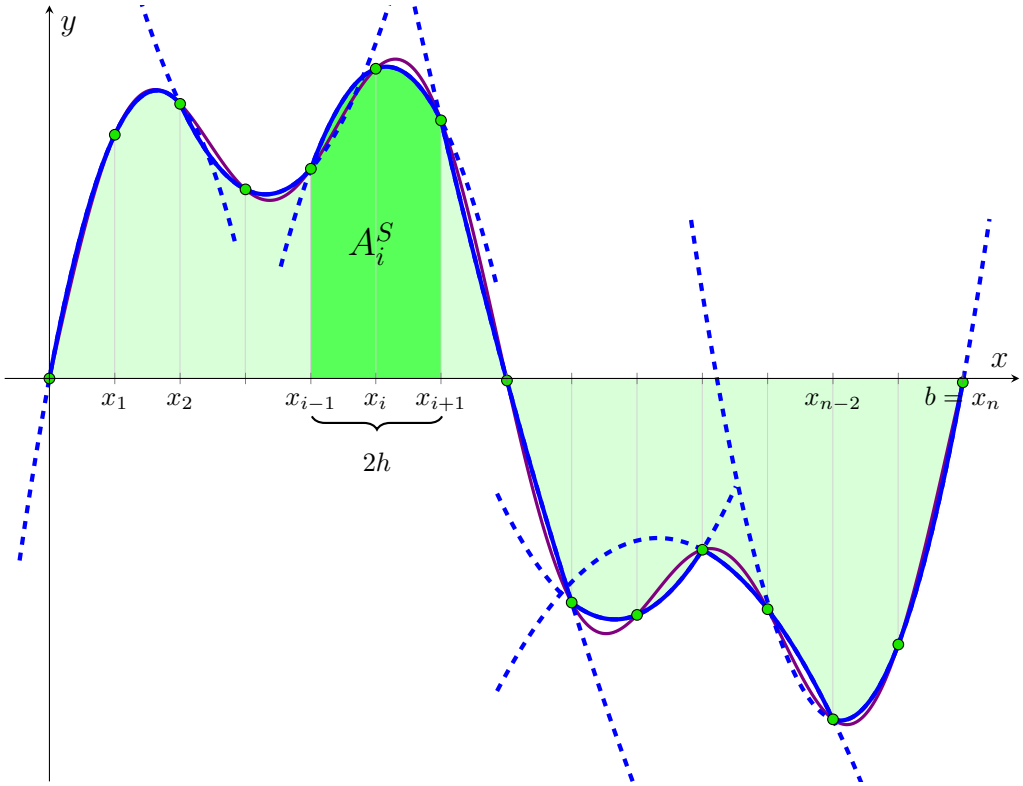

Abbildung 3: Veranschaulichung der Simpsonregel.

Daraus folgt sofort, dass $a_0 = f(x_i)$ und aus Gleichungen $I + III$ ergibt sich

$$2a_2h^2 + 2a_0 = f(x_{i+1}) + f(x_{i+1}) \implies a_2 = \frac{f(x_{i-1}) + f(x_{i+1}) - 2f(x_i)}{2h^2}.$$

Eingesetzt in die Flächenformel für das i-te Flächenstück A_i^S ergibt sich

$$\begin{aligned}
A_i^S &= \int_{x=x_{i-1}}^{x_{i+1}} (a_2x^2 + a_1x + a_0)\mathrm{d}x = [x_{i-1} = -h, x_{i+1} = h] = \\
&= \int_{x=-h}^{h} (a_2x^2 + a_1x + a_0)\mathrm{d}x = \left(a_2\frac{x^3}{3} + a_1\frac{x^2}{2} + a_0x\right)\Bigg|_{x=-h}^{h} = \\
&= \frac{2a_2h^3}{3} + 2a_0h = \left[a_2 = \frac{f(x_{i-1}) + f(x_{i+1}) - 2f(x_i)}{2h^2}, a_0 = f(x_i)\right] = \\
&= \frac{2h^3}{3} \cdot \frac{f(x_{i-1}) + f(x_{i+1}) - 2f(x_i)}{2h^2} + 2f(x_i)h = \\
&= \frac{h}{3}(f(x_{i-1}) + f(x_{i+1}) - 2f(x_i) + 6f(x_i)) = \\
&= \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1})).
\end{aligned}$$

Das Integral über die zu berechnende Gesamtfläche ergibt sich dann zu

$$
\begin{aligned}
I_S &= \sum_{\substack{i=1 \\ i=2n+1 \\ \forall n \in \mathbb{N}_0}}^{n-1} A_i^S = \\
&= \sum_{i=1}^{n-1} \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1})) = \\
&= \frac{h}{3} \cdot (\underbrace{f(x_0) + 4f(x_1) + f(x_2)}_{\int \text{ vom 1. Parabelbogen}} + \underbrace{f(x_2) + 4f(x_3) + f(x_4)}_{\int \text{ vom 2. Parabelbogen}} + \cdots + \\
&+ \underbrace{f(x_{n-2}) + 4f(x_{n-1}) + f(x_n))}_{\int \text{ vom n-ten Parabelbogen}} = \\
&= \frac{h}{3}(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n))
\end{aligned}
$$

Die Formeln für die drei vorgestellten Methoden lassen sich sehr übersichtlich, wie in Tabelle 1 dargestellt, veranschaulichen[5]

	Gesamt-faktor	Faktoren vor $f(x_0)$	$f(x_1)$	$f(x_2)$	$f(x_3)$	$\ldots$	$f(x_{n-2})$	$f(x_{n-1})$	$f(x_n)$
I_{R_L}	h	1	1	1	1	$\ldots$	1	1	0
I_{R_R}	h	0	1	1	1	$\ldots$	1	1	1
I_T	$h/2$	1	2	2	2	$\ldots$	2	2	1
I_S	$h/3$	1	4	2	4	$\ldots$	2	4	1

Tabelle 1: Gegenüberstellung der Rechteck-, Trapez- und Simpson-Regel.

2.4. Konvergenzverhalten & Verfahrensfehler

2.4.1. Eine erste Intuition

Intuitiv würde man vermuten, dass die Simpsonregel mit einer zunehmenden Anzahl an Teilintervallen n am schnellsten gegen den wahren Wert des bestimmten Integrals konvergiert. Um zunächst ein Gefühl für das Konvergenzverhalten zu bekommen, wollen wir die Funktion

$$f : \mathbb{R} \to \mathbb{R} : x \mapsto 2\sin(x) + 0.8\sin(\pi x)$$

betrachten, welche wir in Abschnitt 2 als Funktion in den Abbildungen 1, 2 und 3 verwendet haben.

[5] Wobei wir bei der Rechteck-Regel sowohl die Links- als auch die Rechtsregel betrachten.

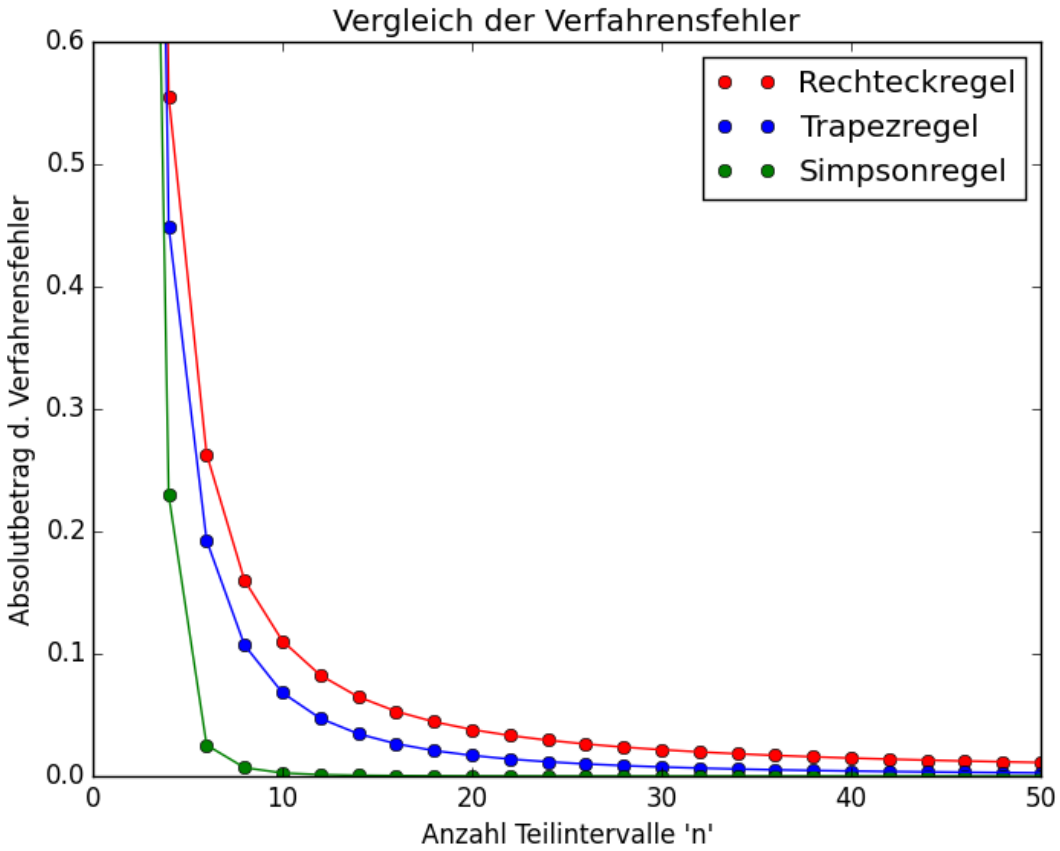

Abbildung 4: Vergleich der Verfahrensfehler.

Diese Funktion kann analytisch integriert werden und die Menge der Stammfunktionen kann wie folgt hingeschrieben werden:

$$\int (2\sin(x) + 0.8\sin(\pi x))\,\mathrm{d}x = -2\cos(x) - \frac{0.8}{\pi}\cos(\pi x) + C$$

Die Berechnung des bestimmten Integrals in den Grenzen von $x = 0$ bis $x = 3$ liefert

$$\int_{x=0}^{3} (2\sin(x) + 0.8\sin(\pi x))\,\mathrm{d}x = \frac{1.6}{\pi} - 2\cos(3) + 2 \approx 4.48928081109496,$$

gerundet auf 14 Stellen. An dieser Stelle greifen wir der computerunterstützten Implementierung der vorgestellten Methoden, welche wir in Abschnitt 3 genauer besprechen werden, vor, um zunächst einen Vergleich der Verfahrensfehler in diesem konkreten Beispiel anzustellen.[6]

[6] Für das hier besprochene Beispiel siehe das Python-Skript `konvergenz.py` in Abschnitt A.1.2.

Abbildung 4 zeigt sehr schön den Verlauf des Absolutbetrags der Verfahrensfehler für eine steigende Anzahl an Teilintervallen n und auch, dass die Simpsonregel, wie vermutet, deutlich am schnellsten konvergiert.[7] Nur der Grafik folgend, scheint in diesem konkreten Beispiel der Verfahrensfehler bei Verwendung der Simpsonregel schon ab $n = 10$ sehr gute Ergebnisse zu liefern, wohingegen die Rechteckregel für $n = 50$ und die Trapezregel für $n = 30$ noch eine deutliche Abweichung vom wahren Wert zeigt. Das können wir aber natürlich noch genauer festhalten: Eine Auswertung der Abweichungswerte mit steigendem n zeigt, dass für eine Ergebnisgenauigkeit auf 3 Nachkommastellen die Simpsonregel in diesem Beispiel lediglich 14 Teilintervalle benötigt, wohingegen die Rechteckregel mit 440 und die Trapezregel mit 84 deutlich mehr Intervalle brauchen. Für eine Genauigkeit auf 4 Nachkommastellen benötigt die Simpsonregel lediglich 24, die Trapezregel schon 260 und die Rechteckregel bereits 4250 Teilintervalle.

2.4.2. Fehlerabschätzung

Nach einem ersten Vergleich der drei Verfahren im vorhergehenden Abschnitt, wollen wir die Fehlerabschätzung für den allgemeinen Fall betrachten, wo wir keine exakte analytische Lösung zur Verfügung haben. Die in der Literatur als „Verfahrensfehler“ bzw. „Quadraturfehler“ bekannten Formeln erlauben eine Abschätzung der maximalen Abweichung des numerisch berechneten Integrals vom 'wahren' Wert. Sie geben also eine *obere Schranke* des Verfahrensfehlers an, welcher nur von der Anzahl der Teilintervalle n und Eigenschaften der konkret betrachteten Funktion f abhängen. Für die Trapezformel gilt

$$|e_T| = \left| \int_a^b f(x)\mathrm{d}x - I_T \right| \leq \frac{(b-a)^3}{12n} \cdot \max_{x\in[a,b]} |f^{(2)}(x)|$$

und für die Simpsonregel

$$|e_S| = \left| \int_a^b f(x)\mathrm{d}x - I_S \right| \leq \frac{(b-a)^5}{2880n^4} \cdot \max_{x\in[a,b]} |f^{(4)}(x)|,$$

wobei $f^{(2)}$ und $f^{(4)}$ für die 2. bzw. 4. Ableitung der Funktion f stehen. Da die Herleitung dieser Abschätzung etwas langwieriger ist, verweisen wir den Leser an dieser Stelle auf entsprechende Fachbücher.

[7] Da wir für die Simpsonregel nur eine gerade Anzahl an Teilintervallen verwenden können, beschränkt sich auch der Vergleich der Verfahrensfehler nur auf gerade n. Darüber hinaus haben wir die Verfahrensfehler für $n = 2$ aus der Grafik ausgeschnitten, um die Unterschiede für größere n besser sehen zu können. Berechnungen mittels der Rechteckregel wurden nach der Linksregel vorgenommen.

3. Implementierung in Python

3.1. Beispiel: Die (Standard)-Normalverteilung

Der Python-Code zur Realisierung der hier vorgestellten Quadraturmethoden findet sich im Skript `numint.py` in Abschnitt A.1.1 wieder. Da wir das Programm so realisiert haben, dass der Benutzer sowohl alle Parameter (zu integrierende Funktion, Integrationsgrenzen, Anzahl der Teilintervalle) als auch die Integrationsmethode auswählen kann, möchten wir diesen Abschnitt dazu nutzen, anhand eines Beispiels Schritt für Schritt durch das Programm zu gehen. Erst in Abschnitt 3.1 gehen wir auf Teile des Programmcodes ein.

Die Funktion, die wir numerisch mittels unserer Python-Implementierung integrieren möchten, ist die Dichtefunktion der (Standard-)Normalverteilung, welche durch

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

gegeben ist. Eine Zufallsvariable X mit eben jener Wahrscheinlichkeits - Dichtefunktion heißt entsprechend $\mathcal{N}(0,1)$-verteilt, also normalverteilt mit Mittelwert $\mu = 0$ und Streuung $\sigma = 1$. Die Funktion verläuft symmetrisch um 0, wo sie auch ihr Maximum mit $\phi(0) = \frac{1}{\sqrt{2\pi}}$ annimmt, besitzt zwei Wendepunkte bei $x = \pm\sigma = \pm 1$ und es gilt: $x \longrightarrow \pm\infty \Rightarrow \phi(x) \longrightarrow 0$.
Eine beliebige normalverteilte Zufallsvariable mit $\mu \in \mathbb{R}$ und $\sigma > 0$ hätte als Dichtefunktion

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad \text{i.e.} \quad X \sim \mathcal{N}(\mu, \sigma).$$

Da es sich (in beiden Fällen) um eine Dichtefunktion handelt, mit deren Hilfe man Aussagen über die Wahrscheinlichkeit für das Eintreten bestimmter Ereignisse treffen kann, beträgt die Gesamtfläche (die Wahrscheinlichkeitsmasse) unter der Kurve genau 1 und entspricht somit der Wahrscheinlichkeit des sicheren Ereignisses. Um Aussagen über das Eintreten bestimmter Ereignisse machen zu können, braucht man regelmäßig die entsprechende Fläche unter der Dichtefunktion, was schließlich auf die Auswertung eines Integrals der Form

$$P([a,b]) = \frac{1}{\sigma\sqrt{2\pi}} \int_{x=a}^{b} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \mathrm{d}x$$

hinausläuft, wobei $P([a,b])$ für die Wahrscheinlichkeit steht, dass die normalverteilte Zufallsvariable mit bekannten Parametern $\mu \in \mathbb{R}$ und $\sigma > 0$ Werte zwischen a und b annimmt. In einem ersten Vereinfachungsschritt sieht man,

dass der Ausdruck durch die Substitution $z = \frac{x-\mu}{\sigma}$ in eine standardnormalverteilte Zufallsvariable wie folgt umgerechnet werden kann:

$$\begin{aligned} P([a,b]) &= \frac{1}{\sigma\sqrt{2\pi}} \int_{x=a}^{b} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \mathrm{d}x = \\ &= \left[z = \frac{t-\mu}{\sigma} \Rightarrow \frac{\mathrm{d}z}{\mathrm{d}x} = \frac{1}{\sigma} \Leftrightarrow \mathrm{d}z = \mathrm{d}x\frac{1}{\sigma} \right] = \\ &= \frac{1}{\sqrt{2\pi}} \int_{z=\frac{a-\mu}{\sigma}}^{\frac{b-\mu}{\sigma}} e^{-\frac{1}{2}z^2} \mathrm{d}z. \end{aligned}$$

Das bedeutet, dass jede normalverteilte Zufallsvariable durch die oben vorgenommene Normierung auf die Standard-Normalverteilung zurückgeführt werden kann. Problematisch gestaltet sich jedoch die Auswertung des angeschriebenen Integrals, welche sich nicht auf eine elementare Stammfunktion zurückführen lässt und daher in Normalverteilungstabellen nachgeschlagen werden muss bzw. in jedem einschlägigen Software-Paket implementiert ist. In diesen Implementierungen bzw. Tabellen finden sich Auswertungen der zur Dichtefunktion gehörenden Verteilungsfunktion, die wie folgt gegeben ist:

$$\begin{aligned} F(x) &= \frac{1}{\sigma\sqrt{2\pi}} \int_{t=-\infty}^{x} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} \mathrm{d}x = \\ &= \left[z = \frac{t-\mu}{\sigma} \right] = \frac{1}{\sqrt{2\pi}} \int_{t=-\infty}^{\frac{x-\mu}{\sigma}} e^{-\frac{1}{2}z^2} \mathrm{d}z = \\ &= \Phi(\frac{x-\mu}{\sigma}), \end{aligned}$$

wobei Φ die Verteilungsfunktion der eingangs besprochenen Standardnormalverteilung ist. Durch entsprechende Standardisierung mittels $\frac{x-\mu}{\sigma}$ lässt sich folglich der zur berechneten Zahl gehörende Wert der Verteilungsfunktion aus einer Tabelle ablesen bzw. einem Softwarepaket auswerten. Im Hinblick auf unser Beispiel bleibt noch zu erwähnen, dass im Intervall der Abweichung $\pm\sigma = \pm 1$ vom Mittelwert 68.27% und im Intervall der Abweichung $\pm 2\sigma = \pm 2$ vom Mittelwert bereits 95.45% aller Messwerte zu finden sind.[8]

Das `numint.py`-Skript kann in der Python-Shell mittels `execfile("numint.py")` ausgeführt werden. Dem User wird daraufhin folgender Screen angezeigt, der ihm/ihr die Funktionsweise des Programms erläutert.

```
------------------------------------------------------------------------
Der folgende Integralrechner berechnet das bestimmte Integral einer vom
Benutzer eingegebenen Funktion f: R --> R in einer Variablen mittels
numerischer Integration.
Dazu müssen neben der interessierenden Funktion auch die Integrationsgrenzen
'a' und 'b', die Anzahl der Teilintervalle 'n' sowie die Berechnungsmethode
eingegeben bzw. ausgewählt werden.
```

[8] Die Fläche unter der Dichtefunktion entspricht also gerundet 0.6827 bzw. 0.9545.

```
      Die implementierten Berechnungsmethoden aus denen gewählt werden kann sind
      die Rechteckregel (1), die Trapezregel (2) und die Simpsonregel (3).
      Bei der Rechteckregel kann wiederum zwischen der Links- und der Rechtsregel
      ausgewählt werden.
      ------------------------------------------------------------------------
Drücken Sie 'Enter' um fortzufahren...
```

Nach Eingabe der Enter-Taste landet der Benutzer bei folgendem Screen

```
      ------------------------------------------------------------------------
      Achten Sie bei der Eingabe der Funktion bitte darauf, dass Hochzahlen in
      python mittels '**' erzeugt werden. Spezielle Funktionen wie beispielsweise
      Winkelfunktion oder die Exponentialfunktion zur Basis 'e' können mittels
      'sin()', 'cos()', 'tan()' bzw. 'exp()' eingegeben werden.
      ------------------------------------------------------------------------
Bitte geben Sie jetzt Ihre Funktion ein: y =
```

an deren Ende man schließlich die zu integrierende Funktion eingeben kann. An dieser Stelle möchten wir darauf hinweisen, dass dem Benutzer bei der Funktionsauswahl natürlich alle Freiheiten gegeben sind (sofern die Eingabe entsprechend der Python-Syntax erfolgt) was jedoch nicht verhindert, dass in entsprechenden Fällen Integrale 2. Art entstehen können. Dahingehend könnte unser Programm noch verfeinert werden indem ein „Integrabilitätscheck“ durchgeführt wird.
In unserem Fall würden wir an dieser Stelle also `(1/sqrt(2*pi))*exp(-x**2/2)` eingeben. Im Anschluss daran werden vom Benutzer noch drei nötige Parameter abgefragt,

```
 Bitte geben Sie die linke Integrationsgrenze 'a' ein, wobei a<b sein muss:
 Bitte geben Sie die rechte Integrationsgrenze 'b ' ein, wobei a<b sein muss:
 Bitte geben Sie die Anzahl der Teilintervalle 'n' ein:
```

welche wir in unserem Beispiel mit -2 (linke Integrationsgrenze), 2 (rechte Integrationsgrenze) und 1000 (Teilintervalle)[9] angeben.
Es folgt eine Zusammenfassung der eingegebenen Funktion und der Parameter

```
      ------------------------------------------------------------------------
      Die von Ihnen eingegebenen Daten lauten:
      'y'= (1/sqrt(2*pi))*exp(-x**2/2)
      'a' = -2
      'b' = 2
      'n' =  1000
      ------------------------------------------------------------------------
Wenn Sie nun 'Enter' drücken erhalten Sie einen plot der Funktion...
```

Damit sich der/die Benutzer/in ein ungefähres Bild von der Funktion machen kann (sofern er/sie das nicht schon vorher gemacht hat) wird nach erneuter Eingabe der Enter-Taste ein Plot der Funktion innerhalb der Integrationsgrenzen generiert. In unserem Fall lässt der Graph der Funktion erahnen, dass es sich um die Standardnormalverteilung handeln könnte wohingegen bei anders liegenden Integrationsgrenzen der Informationsgehalt des Plots möglicherweise eher gering ist. In unserem Fall ergibt sich ein Plot wie in Abbildung 5.

[9] Bei unseren Testläufen haben Berechnungen mit knapp 1 Mio. Teilintervallen problemlos funktioniert.

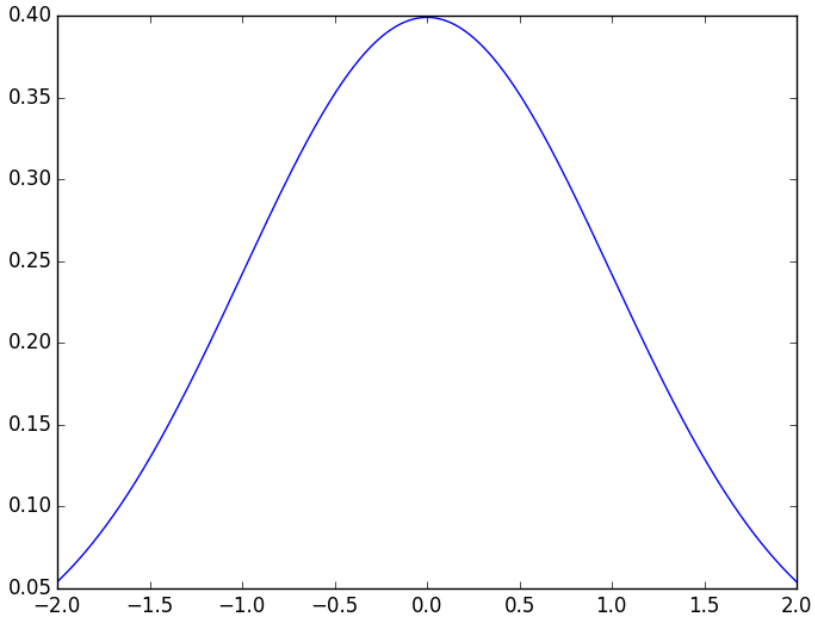

Abbildung 5: Funktionsplotter des „numint.py“-scripts.

Um mit dem Programm fortfahren zu können, muss der Benutzer das Plotfenster schließen und mit einer erneuten Enter-Eingabe fortsetzen. Damit gelangt man schließlich zum Kern des Programms: dem Integrationsrechner. Dabei hat der Nutzer 4 Optionen aus denen er auswählen kann. Folgender Text erscheint am Screen:

```
    ------------------------------------------------------------------------
    Wälen Sie nun die zu verwendende Integrationsmethode. Geben Sie die Zahl '1'
    für die Rechteckregel, die Zahl '2' für die Trapezregel oder die Zahl '3' für
    die Simpsonregel.
    Sollten Sie hingegen einen Vergleich der Ergebnisse aller drei Methoden
    benötigen, so geben Sie die Zahl '4' ein.
    ------------------------------------------------------------------------
Wählen Sie nun die zu verwendende Integrationsmethode:
```

Um einen Vergleich zwischen den Methoden zu erhalten, wählen wir hier Option (4). Sowohl bei Wahl der Option (4) als auch der Option (1)[10] wird dem Benutzer anschließend noch folgende Wahlmöglichkeit gestellt:

```
    ------------------------------------------------------------------------
    Die Rechteckregel kann das zu evaluierende Integral entweder nach der
    Links- oder nach der Rechtsregel berechnen.
    ------------------------------------------------------------------------
Geben Sie bitte '1' für die Links- bzw. '2' für die Rechtsregel ein:
```

Im Zuge der Rechteckregel ist also die Wahl der Stützstellen nicht völlig beliebig, sondern es wird entweder nach der Links- oder nach der Rechtsregel

[10] Wo also nur nach der Rechteckregel berechnet wird.

vorgegangen. Wir wählen hier Option (1), die Linksregel, und erhalten schließlich folgenden Output:

```
Fläche Rechteckregel =  0.954499448152
Fläche Trapezregel =  0.954499448152
Fläche Simpsonregel =  0.954499736103
```

Wir sehen, dass sich das Ergebnis zwischen Rechteck- und Trapezregel nicht unterscheidet; mittels Simpsonregel ergibt sich ab der 7. Nachkommastelle ein etwas anderes Ergebnis. Ein entsprechender Vergleich mit der Auswertung am PC für die wir den Wert 0.954499736104 erhalten, zeigt, dass der Unterschied zu dem von uns mittels der Simpsonregel berechneten Wert mit $\Delta = 9.9997787828 \cdot 10^{-13}$ verschwindend gering ist.

4. Zusammenfassung und Ausblick

In dieser Arbeit haben wir, um den Umfang überschaubar zu halten, vom höherdimensionalen Fall der multivariaten Quadratur abgesehen und mit der Rechteck-, Trapez- und Simpsonregel die Grundlagen der univariaten numerischen Quadratur besprochen. Eine entsprechende Implementierung dieser Verfahren in Python ließ uns relativ zügig einfache wohldefinierte Integrale berechnen wobei die Simpsonregel, betreffend der Konvergenzgeschwindigkeit, eindeutig am effizientesten funktioniert.
Tatsächlich ist es gerade die multivariate Quadratur, die die größeren Herausforderungen hinsichtlich der Verfahren selbst als auch der Implementierung in sich birgt. Die in dieser Arbeit vorgestellten Methoden sind daher als Grundlage immer komplexerer und aufwendigerer Methoden Verfahren zu sehen.

A. Appendix

A.1. Python-Code

A.1.1. numtint.py

```
# -*- coding: utf-8 -*-

##########Kurze Beschreibung#########
#Der Integrationsrechner ermöglicht die Berechnung von bestimmten Integralen mittels
#numerischer Integration. Vom Benutzer werden dazu die Funktion, die Integrationsgrenzen,
#die Anzahl der Teilintervalle und die Methode, nach der integriert werden soll, erfragt.
#Die implementierten Integrationsmethoden sind die Rechteck-, die Trapez- und die
#Simpson-Regel.
############################################

############################################
############Import Pakete#################
############################################
from math import *
import numpy as np #um das plotten zu erleichtern
import matplotlib.pyplot as plt #um Funktionen zu plotten
import parser #um die eingegebene mathematische Funktion zu evaluieren
############################################


############################################
#Umwandlung der vom User eingegebenen Funktion
#in einen mathematischen Ausdruck,
#der weiterverarbeitet werden kann:

#Innerhalb des Programms kann der User eine von ihm gewünschte Funktion eingeben,
#welche als "string" gespeichert wird. Unser Ziel ist es dann diese Funktion so zu
#manipulieren, dass sie für beliebige x-Werte ausgewertet werden kann.
#Das kann mit Hilfe eines parsers realisiert werden, der die Struktur des eingegebenen
#Strings (der eingegebenen Symbolfolge) erkennt, sodass die Funktion dann als
#mathematische Funktion weiterverarbeitet werden kann, welcher man x-Werte als
#Argumente übergeben kann.

#Die geparste Funktion nennen wir "fx".
#Die Funktion "func()" nimmt als Übergabeargumente beliebige x-Werte und wertet die
#Funktion f(x), welche wir auch als "fx" bezeichnet haben, an der Stelle "x" aus.

##die folgende Funktion 'func()' nimmt als Übergabeargumente x-Werte und wertet dabei die
##geparste Funktion fx an der Stelle x aus.
##die Umwandlung der eingebenen Funktion als 'string' in eine 'evaluierbare' Funktion
##fx erfolgt weiter unten im Programmcode.
def func(x):
    return eval(fx)
############################################
```

```
##########################################
############****Rechteckregel****###############
##########################################
def integrate_rectangle(a, b, n):
    #Berechnung der Intervalllänge
    h = ((b - a) / n)
    #Fläche A auf 0 gesetzt
    A = 0

    #Auswahl: Links oder Rechtsregel
    print("\t ------------------------------------------------------------------------")
    print("\t Die Rechteckregel kann das zu evaluierende Integral entweder nach der     ")
    print("\t Links- oder nach der Rechtsregel berechnen."                               )
    print("\t ------------------------------------------------------------------------")
    choice1 = input("Geben Sie bitte '1' für die Links- bzw. '2' für die Rechtsregel ein: ")
    if choice1 == 1:
        for i in range(1, n+1):
            #linke Intervallgrenze
            x0 = a+(i-1)*h
            #ite Fläche
            Ai = h * func(x0)
            #Summe über alle Flächen
            A = A + Ai
        #Ergebnis
        print "Fläche Rechteckregel = ", A

    if choice1 == 2:
        for i in range(1, n+1):
         #rechte Intervallgrenze
            x0 = a + i*h
            #ite Fläche
            Ai = h * func(x0)
            #Summe über alle Flächen
            A = A + Ai
        #Ergebnis
        print "Fläche Rechteckregel = ", A
##########################################


##########################################
#############****Trapezregel****###############
##########################################
def integrate_trapezoid(a, b, n):
    #Berechnung der Intervalllänge
    h = ((b - a) / n)
    #Fläche A auf 0 gesetzt
    A = 0

    #Loop um die Fläche jedes der i Trapezoide zu berechnen
    #am Schluss wird aufsummiert.
    for i in range(1, n+1):
```

```
        #Berechnung der linken und rechten Intervallegrenze des i-ten Trapezoids
        x0 = a+(i-1)*h
        x1 = a+i*h

        #Flächenberechnung für das i-te Trapezoid
        Ai = h * (func(x0) + func(x1))/ 2.

        #Summe über alle i Flächen
        A = A + Ai
    #Ergebnis
    print "Fläche Trapezregel = ", A
##############################################


##############################################
#############****Simpsonregel****##############
##############################################
def integrate_simpson(a, b, n):
    #Überpfügun ob n gerade ist:
    if n % 2 == 0:
        #Berechnung der Intervalllänge
        h = ((b - a) / n)

        A = 0

        #Loop um die i-te Fläche zu berechnen
        #am Schluss wird aufsummiert.
        for i in range(1, n+1, 2):
            #Berechnung der linken und rechten Intervallegrenze des i-ten
            #Parabelbogens
            x0 = a + (i-1) * h
            x1 = a + i * h
            x2 = a + (i+1)*h

            #Fläche unter dem iten Parabelbogen
            Ai = h/3 * (func(x0) + 4*func(x1) + func(x2))

            #Summe über alle Flächen
            A = A + Ai

        #Ergebnis
        print "Fläche Simpsonregel = ", A
    else:
        print("\t ------------------------------------------------------------------------")
     print("Achtung! Da bei der Simpsonregel Parabelbögen zur Approximation an die zu")
     print("integrierende Kurve gelegt werden, ist eine gerade Anzahl an")
     print("Teilintervallen nötig! Bitte korrigieren Sie Ihre Eingabe!")
     n = input("Bitte geben Sie erneut die Anzahl der Teilintervalle 'n' ein:")
     print("\t ------------------------------------------------------------------------")
     integrate_simpson(a, b, n)
##############################################
```

```
print("\t -------------------------------------------------------------------------------")
print("\t Der folgende Integralrechner berechnet das bestimmte Integral einer vom"       )
print("\t Benutzer eingegebenen Funktion f: R --> R in einer Variablen mittels"          )
print("\t numerischer Integration."                                                      )
print("\t Dazu müssen neben der interessierenden Funktion auch die Integrationsgrenzen"  )
print("\t 'a' und 'b', die Anzahl der Teilintervalle 'n' sowie die Berechnungsmethode"   )
print("\t eingegeben bzw. ausgewählt werden."                                           )
print("\t Die implementierten Berechnungsmethoden aus denen gewählt werden kann sind"    )
print("\t die Rechteckregel (1), die Trapezregel (2) und die Simpsonregel (3)."          )
print("\t Bei der Rechteckregel kann wiederum zwischen der Links- und der Rechtsregel"   )
print("\t ausgewählt werden."                                                            )
print("\t -------------------------------------------------------------------------------")
raw_input("Drücken Sie 'Enter' um fortzufahren..."                                      )

print("\t -------------------------------------------------------------------------------")
print("\t Achten Sie bei der Eingabe der Funktion bitte darauf, dass Hochzahlen in "     )
print("\t python mittels '**' erzeugt werden. Spezielle Funktionen wie beispielsweise"   )
print("\t Winkelfunktion oder die Exponentialfunktion zur Basis 'e' können mittels"      )
print("\t 'sin()', 'cos()', 'tan()' bzw. 'exp()' eingegeben werden."                     )
print("\t -------------------------------------------------------------------------------")

user_fx = raw_input("Bitte geben Sie jetzt Ihre Funktion ein: y = ")
fx = parser.expr(user_fx).compile()
a = input("Bitte geben Sie die linke Integrationsgrenze 'a' ein, wobei a<b sein muss:"   )
b = input("Bitte geben Sie die rechte Integrationsgrenze 'b ' ein, wobei a<b sein muss:" )
n = input("Bitte geben Sie die Anzahl der Teilintervalle 'n' ein:")
print("\t -------------------------------------------------------------------------------")
print "\t Die von Ihnen eingegebenen Daten lauten:"
print "\t 'y'=", user_fx
print "\t 'a' =", a
print "\t 'b' =", b
print "\t 'n' = ", n
print("\t -------------------------------------------------------------------------------")

#wir machen Kommazahlen aus den eingegebenen Grenzen
a = float(a)
b = float(b)


raw_input("Wenn Sie nun 'Enter' drücken erhalten Sie einen plot der Funktion...")


##############################################
############****Plot der Funktion****############
##############################################
def drange(start, stop, jump):
    while start <= stop:
        yield start
        start += jump
xs = drange(a, b, 0.001)
xs = [x for x in xs]
```

```
ys = []
for x in xs:
    ys.append(func(x))
plt.plot(xs, ys)
plt.show()
##########################################


raw_input("Drücken Sie 'Enter' um fortzufahren...")


print("\t -------------------------------------------------------------------------------")
print("\t Wälen Sie nun die zu verwendende Integrationsmethode. Geben Sie die Zahl '1'")
print("\t für die Rechteckregel, die Zahl '2' für die Trapezregel oder die Zahl '3' für ")
print("\t die Simpsonregel.")
print("\t Sollten Sie hingegen einen Vergleich der Ergebnisse aller drei Methoden"       )
print("\t benötigen, so geben Sie die Zahl '4' ein."                                     )
print("\t -------------------------------------------------------------------------------")
choice2=input("Wählen Sie nun die zu verwendende Integrationsmethode: ")
if choice2==1:
    integrate_rectangle(a, b, n)

if choice2==2:
    integrate_trapezoid(a, b, n)

if choice2==3:
    integrate_simpson(a, b, n)

if choice2==4:
    integrate_rectangle(a, b, n)
    integrate_trapezoid(a, b, n)
    integrate_simpson(a, b, n)
```

A.1.2. konvergenz.py

```
# -*- coding: utf-8 -*-
##########################################
###***Beispiel: Konvergenzverhalten***#####
##########################################

##########################################
############****Import Pakete****###############
##########################################
from math import *
import numpy as np #um das plotten zu erleichtern
import matplotlib.pyplot as plt #um Funktionen zu plotten
import parser #um die eingegebene mathematische Funktion zu evaluieren
##########################################

#tatsächlicher Wert des bestimmten Integrals
Aacc = 1.6/pi - 2*cos(3) + 2
```

```
#in diesem Beispiel betrachtete Funktion
example_function = "2*sin(x) + 0.8*sin(pi*x)"
#Parser, der die Funktion, die wir oben als String
#eingegeben haben in eine Funktion in Abhängigkeit
#von x umwandelt
parsed_function = parser.expr(example_function).compile()
#Funktion, die die 'parsed_function' für beliebige
#x-Werte auswertet
def func(x):
    return eval(parsed_function)


##########################################
##########****Rechteckregel****###########
##########################################
def integrate_rectangle2(a, b, n):
    a = float(a)
    b = float(b)
    #Berechnung der Intervalllänge
    h = ((b - a) / n)
    #A wird auf 0 gesetzt
    A = 0

    for i in range(1, n+1):
        #linke Intervallgrenze des iten Intervalls
        x0 = a+(i-1)*h

        #Fläche des iten Rechtecks
        Ai = h * func(x0)

        #Summe über alle Rechtecke
        A = A + Ai
    return A
##########################################


##########################################
##########****Trapezregel****############
##########################################
def integrate_trapezoid2(a, b, n):
    a = float(a)
    b = float(b)
    #Berechnung der Intervalllänge
    h = ((b - a) / n)
    #A wird auf 0 gesetzt
    A = 0

    for i in range(1, n+1):
        #Berechnung der linken und rechten Intervallegrenze des i-ten Trapezoids
        x0 = a+(i-1)*h
        x1 = a+i*h
```

```
        #Flächenberechnung für das i-te Trapezoid
        Ai = h/2 * (func(x0) + func(x1))

        #Summe über alle i Flächen
        A = A + Ai
    return A
##########################################


##########################################
#############****Simpsonregel****##########
##########################################
def integrate_simpson2(a, b, n):
    a = float(a)
    b = float(b)
    #Berechnung der Intervalllänge
    h = ((b - a) / n)
    #A wird auf 0 gesetzt
    A = 0

    for i in range(1, n+1, 2):
        #Berechnung der linken und rechten Intervallegrenze des i-ten Objekts
        x0 = a + (i-1) * h
        x1 = a + i * h
        x2 = a + (i+1)*h

        #i-te Flächenberechnung
        Ai = h/3 * (func(x0) + 4*func(x1) + func(x2))

        #Summe über alle i Flächen
        A = A + Ai
    return A
##########################################


##########################################
############****Plot der Funktion****##########
##########################################
a = 0
b = 3
def drange(start, stop, jump):
    while start <= stop:
        yield start
        start += jump
xs = drange(a, b, 0.001)
xs = [x for x in xs]
ys = []
for x in xs:
    ys.append(func(x))
plt.plot(xs, ys)
plt.show()
##########################################
```

```
##########################################
## Funktion, die die Abweichung zum
## wahren Wert des Integrals, abhängig
## von der Integrationsmethode, für
## n = 2 bis 50 berechnet.
##########################################
def convergence1(Atrue):
    rect = []
    trap = []
    simp = []
    error_rect = []
    error_trap = []
    error_simp = []
    for j in range(2, 51, 2):
        rect.append(integrate_rectangle2(0, 3, j))
        trap.append(integrate_trapezoid2(0, 3, j))
        simp.append(integrate_simpson2(0, 3, j))
    for k in range(0, 25, 1):
        error_rect.append(abs(Atrue - rect[k]))
        error_trap.append(abs(Atrue - trap[k]))
        error_simp.append(abs(Atrue - simp[k]))
    print error_rect
    print error_trap
    print error_simp
    return error_rect, error_trap, error_simp

#Anwendung der Funktion
error1, error2, error3 = convergence1(Aacc)

x = np.linspace(2, 50, 25)
plt.xlabel("Anzahl Teilintervalle 'n'")
plt.ylabel("Absolutbetrag d. Verfahrensfehler")
plt.title("Vergleich der Verfahrensfehler")
plt.ylim(ymax = 0.6)
plt.plot(x, error1, 'ro', label = 'Rechteckregel')
plt.plot(x, error2, 'bo', label = 'Trapezregel')
plt.plot(x, error3, 'go', label = 'Simpsonregel')
plt.plot(x, error1, 'r')
plt.plot(x, error2, 'b')
plt.plot(x, error3, 'g')
plt.legend(loc='upper right')
plt.show()

##########################################
## Funktion, die die Anzahl an Teilintervallen
## für eine beliebige eingegebene Genauigkeit
## berechnet für unser konkretes
## Beispiel berechnet.
##########################################
def convergence2(Atrue, tol):
```

```
    j = 2
    k = 2
    l = 2
    Arect = integrate_rectangle2(0, 3, j)
    Atrap = integrate_trapezoid2(0, 3, k)
    Asimp = integrate_simpson2(0, 3, l)
    while abs(Arect-Atrue) > tol:
        j = j + 2
        Arect = integrate_rectangle2(0, 3, j)

    while abs(Atrap-Atrue) > tol:
        k = k + 2
        Atrap = integrate_trapezoid2(0, 3, k)

    while abs(Asimp-Atrue) > tol:
        l = l + 2
        Asimp = integrate_simpson2(0, 3, l)
    return j, k, l


#n11, n22, n33 = convergence2(Aacc, 10**(-6))
#print "Nach der Rechteckregel werden %d Iterationen benötigt." % (n11)
#print "Nach der Trapezregel %d Iterationen benötigt."  % (n22)
#print "Nach der Simpsonregel %d Iterationen benötigt."  % (n33)

#Anwendung der Funktion
n1, n2, n3 = convergence2(Aacc, 10**(-4))
print "Nach der Rechteckregel werden %d Iterationen benötigt." % (n1)
print "Nach der Trapezregel %d Iterationen benötigt."  % (n2)
print "Nach der Simpsonregel %d Iterationen benötigt."  % (n3)
```

Literatur

Götz, S., Reichel, H.-C., Müller, R. & Hanisch, G. (2004), *Lehrbuch der Mathematik*, 4te edn.

Huckle, T. (2006/07), 'Numerische Quadratur', `http://www5.in.tum.de/lehre/vorlesungen/konkr_math/06_07/vorl/Handout_03.pdf`. [Online; aufgerufen am 9. Jänner, 2015].

Kuhlmann, H.-C. (2012), 'Numerische Integration', `http://www.fluid.tuwien.ac.at/322036?action=AttachFile&do=get&target=Integration.pdf`. [Online; aufgerufen am 26. Dezember, 2014].

Kun, J. (2012), 'Numerical Integration', `http://jeremykun.com/2012/01/08/numerical-integration/`. [Online; aufgerufen am 26. Dezember, 2014].

Meyberg, K. & Vachenauer, P. (2003), *Höhere Mathematik 1*, 6th edn.

Nestler, B. (2012), 'Numerische Integration', `https://www.iam.kit.edu/zbs/download/lehre/ModSim-Skript6.pdf`. [Online; aufgerufen am 26. Dezember, 2014].

Wagner, P. (1996/1997), Mathematik A - Skriptum zur Vorlesung, Technical report, Institut für Technische Mathematik, Geometrie und Bauinformatik, Universität Innsbruck.