Technische Universität Wien
Institut für Analysis and Scientific Computing
Prof. Dr. Ansgar Jüngel

Winter 2010/11

Seminar „Numerical Digit Challenge“

Dieses Dokument enthält die Seminarausarbeitungen der Teilnehmenden

1. Tobias Fellinger, Guido Schlögel
2. Florian Kain, Gregor Mitscha-Eibl
3. Florian Scheweder, Peter Regner
4. Karl Rupp (Gastbeitrag zur ersten Seminaraufgabe)

Seminar "'Numerical Digit Challenge"'

Tobias Fellinger, Guido Schlögel

28. Januar 2011

Aufgaben 3,4,5 hat Guido Schlögel bearbeitet, Aufgaben 1 und 2 Tobias Fellinger, die Numerischen Experimente zu Aufgabe 2 stammen von Guido Schlögel.

1 Aufgabe 1

Die Kempner Reihe $\sum_{n=1}^{\infty} b_n$ mit $b_n = 1$ wenn b_n nicht eine bestimmte Ziffer enthält und 0 sonst konvergiert bekannter weise und lässt sich mittels Rekurrenzen gut berechnen. Wenn man die Zusatzbedingung einführt, dass die Zahl noch dazu eine Primzahl sein muss konvergiert diese Summe als Summe über eine kleinere Menge natürlich auch, die Rekurenzen versagen aber. Es wurde keine bessere Methode gefunden als die Summe bis zu einem endlichen Glied "brute forceßu berrechnen und dann über die Differenz zur Partialsumme K_n der Kempnerreihe bis zum selben Glied den Fehler zu schätzen. Offensichtlich gilt ist die Kempnerreihe K_n über jede Teilmenge von $\mathbb{N}$ summiert größer als S_n. Damit gilt dann:

$$S - S_n < K - K_n$$

Mit der Rekurrenz für K_n und einer brute-force Berechnung von S_n ergibt sich dann für $N = 10^10$:
$S_N = 1.976243082$ und $K_N = 8.636667685$. Auf http://oeis.org/A082830 kann man nachlesen dass $\lim_{n\to\infty} K_n = 16.176969528$. Mit $S - S_n < K - K_n$ folgt dann $S < K - K_n + S_n$. Also $S < 9.516544925$ da die Partialsummen monoton wachsend sind gilt sicher auch $S > S_N = 1.976243082$. Die Folgen weiter explizit zu berechnen war vor allem wegen des Speicherbedarfs des verwendeten Primzahlsiebs nicht möglich, obwohl die Lauzeiten noch sehr gering waren. C Code für die Summation befindet sich im Anhang an die Abgabe.

2 Aufgabe 2

Bei dieser Aufgabe war erst einmal nicht klar in welchem Sinn die Folge überhaupt konvergiert. Aus der Literatur ist jedoch bekannt, dass die Folge fast sicher gegen den Grenzwert σ konvergiert. Für die Stochastische Fibonacci Folge ohne den Vorfaktor $\frac{3}{5}$

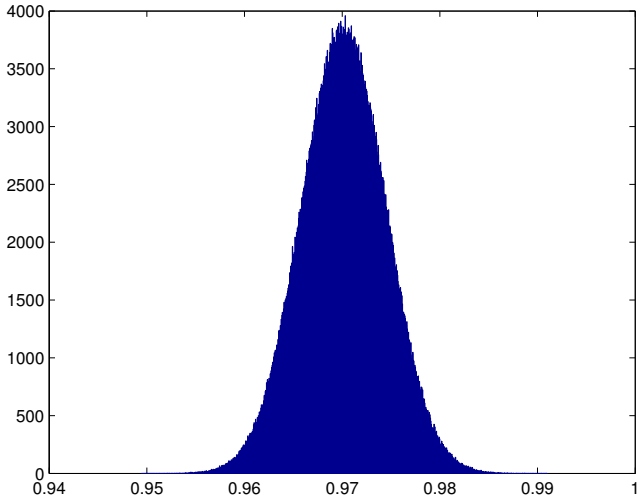


Abbildung 1: Histogramm

ist die Lyapunov Konstante bekannt, die die assymptotische Grow Rate σ beschreibt. Für die Fibonacci Folge mit der wir es hier zu zun haben ist uns die Konstruktion einer solchen Konstante aber nicht gelungen. Wir haben daher zur Approximation des Grenzwertes auf numerische Experimente zurückgegriffen. Auch hier ist eine Schätzung der Genauigkeit schwer da nicht klar ist ob der Erwartungswert der Verteilung auch der Grenzwert ist. Dazu wurde die Verteilung des $n$ten Folgenglieds von p Zufallsfolgen betrachtet. Bei $n = 10^4$ und $p = 10^6$ ergab das einen Erwartungswert von 0.97157. Die Abbildung zeigt das Histogramm dieses Versuches.

3 Aufgabe 3

Berechnen Sie

$$I = \lim_{\epsilon \to o} \int_{\epsilon}^{1} \sin\left(cos(x)e^{\frac{1}{x}}\right) dx \tag{1}$$

Das Problem beim Berechnen des Integrals ist, dass dieses für $x \to 0$ stark oszilliert. Die Folge der Nullstellen geht dabei nur logarithmisch gegen 0.

3.1 Aufteilung des Integrationsbereiches und Reihenapproximation

Das Integral lässt sich in eine Reihe umwandeln, bei der die Reihenglieder immer das Integral zwischen 2 Nullstellen sind. In diesem Intervall lässt sich die Funktion problemlos integrieren (sie verhält sich ähnlich wie der Sinus zwischen 2 benachbarten Nullstellen).

Zunächst müssen die Nullstellen bestimmt werden. Dabei wird verwendet, dass $sin(x)$ genau dann Null ist, wenn $x = n\pi$ für $n \in \mathbb{Z}$. In unserm Fall gilt also für ein Nullstelle

x_n

$$\cos(x_n)e^{\frac{1}{x_n}} = n\pi \qquad n \in \mathbb{N}$$

Die Nullstellen befinden sich für $n \geq 1$ im Intervall $(0, 1)$ und sie sind absteigend sortiert. Sie lassen sich nicht analytisch bestimmen, sind aber leicht numerisch mit Hilfe des Newtonverfahrens bestimmbar. Als Anfangswert für die Iteration wird dabei jeweils die vorrangegengene Nullstelle verwenden.

Setzt man $x_0 = 1$ gilt

$$\int_0^1 \sin\left(\cos(x)e^{\frac{1}{x}}\right) dx = \sum_{n=0}^{\infty} a_n \qquad a_n = \int_{x_{n+1}}^{x_n} \sin\left(\cos(x)e^{\frac{1}{x}}\right)$$

Die Quadratur wurde mit der Matlab-Funktion quadgk durchgeführt, die eine adaptive Quadratur hoher Ordnung implementiert. Als relative Fehlertoleranz wurde die 100-fache Maschinengenauigkeit (100 eps) gewählt. Könnte man die Reihe exakt berechnen würde man somit einen Gesammtfehler kleiner 10^{-14} erhalten.

Als nächstes müssen Abschätzungen für das Verhalten der Reihe gefunden werden. Asymptotisch kann das Verhalten der Nullstellen approximiert werden durch

$$e^{\frac{1}{x_n}} \sim n\pi \qquad \Rightarrow \qquad x_n \sim \frac{1}{\log(n\pi)}$$

Damit gilt für den Abstand zwischen 2 Nullstellen

$$x_n - x_{n+1} = \frac{1}{\log(n\pi)} - \frac{1}{\log((n+1)\pi)} = \frac{\log\left(1 + \frac{1}{n+1}\right)}{\log(n\pi)\log((n+1)\pi)}$$

weil $x_n \sim x_{n+1}$ folgt durch Taylorentwicklung

$$x_n - x_{n+1} = \mathcal{O}\left(\frac{1}{n(\log n)^2}\right)$$

Da der Integrand zwischen 0 und 1 oszilliert verhalten sich die $|a_n|$ wie die Intervalllänge. Die Reihe ist alternierend. Auf Grund der Abschätzung der Nullstellen erwarten wir mindestens eine logarithmische Konvergenz, erhoffen uns aber, weil die Reihe alternierend ist, eine bessere Konvergenzordnung.

Experimentall erhält man eine Konvergenzrate zwischen $\mathcal{O}(n)$ und $\mathcal{O}(n^2)$. Will man möglichst viele Stellen bestimmen wäre ein höhere Konvergenzrate wünschenswert. Eine Möglichkeit ist es die Reihe nicht durch Integration zwischen den Nullstellen zu berechnen, sondern zwischen 2 Extremstellen. Dies führt zu kleineren Summanden und auch zu einer besseren Konvergenzrate (experimentell zwischen $\mathcal{O}(n^2)$ und $\mathcal{O}(n^3)$).

Um die Konvergenzrate weiter zu verbessern wurde das Aitken'sche Δ^2-Verfahren zur Konvergenzbeschleunigung angewendet. Dies führt zu einer experimentellen Konvergenzrate besser als $\mathcal{O}(n^4)$ im Fall der Verwendung der Nullstellen bzw. $\mathcal{O}(n^5)$ bei Verwendung der Extrema. Abbildung 2 zeigt die erzielten Konvergenzraten für $n = 1000$.

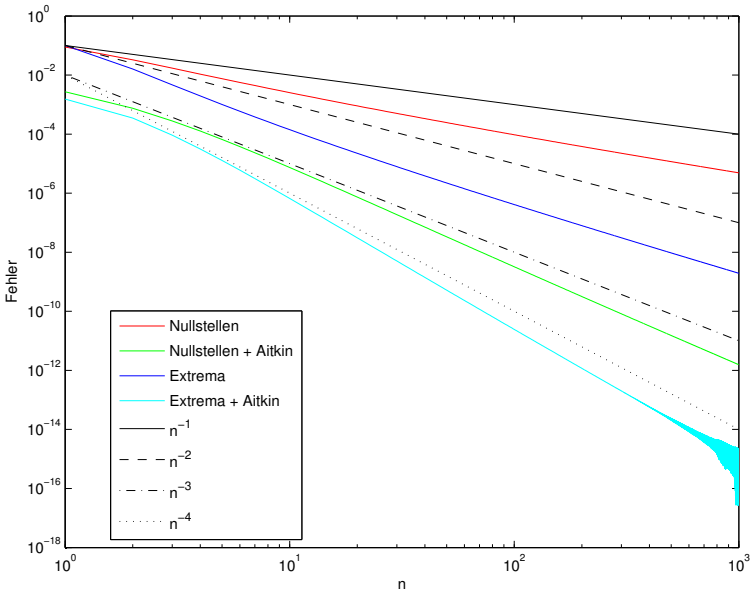


Abbildung 2: Konvergenzraten der Reihen

Verwendung der Nullstellen	**0.126212**125655828
Verwendung der Nullstellen (beschleunigt)	**0.126217015840**340
Verwendung der Extrema	**0.126217017**785096
Verwendung der Extrema (beschleunigt)	**0.126217015841907**

Tabelle 1: Ergebnisse der Approximation der Reihe

Zur Schätzung des Fehlers wurde der konvergenzbeschleunigte Wert bei Verwendung der Extremstellen verwendet.

In Tabelle 1 wurden die Ergebnisse für $n = 1000$ zusammengefasst. Die fett gedruckten Ziffern sind diejenigen, die als richtig angesehen werden.

Zu beachten ist, dass für $n = 1000$ das Integral nur im Intervall $x \in (0.124, 1)$ ausgewertet wird. Auf den Rest des Integrals wird mittels Extrapolation geschlossen. Da wir aber wissen, wie sich der Integrand und damit dir Reihe für kleine x verhält, können die erhalten Ziffern als richtig angesehen werden.

4 Aufgabe 4

Lösen Sie das Integral

$$J = \int_{\mathbb{R}^8} \sin\left(\prod_{n=1}^{8} |x_n|^n\right) \exp\left(-\frac{1}{10}\sum_{n=1}^{8} x_n^2\right) dx$$

Beim Exponentialterm handelt es sich bis auf einen Vorfaktor um die Dichte einer Normalverteilung. Er ist auch dahingehend „gutartig“, dass er ein Produkt von Funktion zerfällt, das jeweils von nur einer Variablen abhängt. Man kann eine einfache obere und untere Abschätzung erhalten, wenn man den Sinus mit 1 bzw. -1 abschätzt

$$|J| \leq \left(\int_{\mathbb{R}} \exp\left(-\frac{1}{10}x^2\right) dx\right)^8 = (20\pi)^4 \sim 1.55\ 10^7$$

4.1 Versuch die Struktur des Integranden auszunutzen

Ziel ist es den Sinus-Term so zu approximieren, dass es nicht mehr nötig ist ein 8-dimensionales Integral zu lösen, sondern nur noch 1-dimensionale Integrale. Ein Ansatz ist den Sinus durch seine Taylorreiche auszudrücken. Es ergibt sich

$$J = \int_{\mathbb{R}^8} \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} \prod_{n=1}^{8} |x_n|^{n(2k+1)} \exp\left(-\frac{x_n^2}{10}\right) dx$$

Hier ist es aber nicht möglich die Reihe mit dem Integral zu vertauschen. Schränkt man jedoch den Integrationsbereich ein ein endliches 8-dimensionales Rechteck ein ($\Omega := [a_n, b_n]_{n=1}^8$), kann man das Integral und den Grenzwert vertauschen und es ergibt sich

$$J|_\Omega = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} \prod_{n=1}^{8} \int_{a_n}^{b_n} |x_n|^{n(2k+1)} \exp\left(-\frac{x_n^2}{10}\right) dx_n$$

Die Reihe konvergiert zwar, aber die einzelnen Reihenglieder sind im Vergleich zum erwarteten Ergebnis so groß, dass jede numerische Berechnung scheitern muss.

Das auftretende Problem stellt sich nicht nur bei der Taylorentwicklung, sondern bei jeder Polynomapproximation, etwa durch Interpolation. Damit das Integral in 1-dimensionale Integrale zerfällt ist es notwendig das approximierende Polynom in der Monombasis anzuschreiben und es treten die gleichen Probleme auf.

Mittelwert	Standardabweichung	Konfidenzintervall 95 %	Konfidenzintervall 99 %
2.2712851 10^4	59.18376	$[2.259685, 2.282884]$	$[2.256040, 2.286529]$

Tabelle 2: Ergebnisse der Monte-Carlo Quadratur

4.2 Monte-Carlo Methode

Da alle anderen Ansätze scheiterten musste das Monte-Carlo Verfahren verwendet werden. Das Verfahren konvergiert nur mit der Rate $\mathcal{O}(\sqrt{n})$, daher war es nicht erste Wahl.

Um nicht das Integral über einen Raum mit unendlichen Maß betrachten zu müssen wird das Integral so umgeschrieben, dass bezüglich des Maßes der Normalverteilung mit Mittelwert 0 und Varianz 10 integriert wird. Es ergibt sich

$$J = (20\pi)^4 \int_{\mathbb{R}} \sin\left(\prod_{n=1}^{8} x_n^n\right) dN_{0,10}$$

Man geht jetzt in folgenden Schritten vor

1. erzeuge 8-dimensionale normalverteilte Zufallszahlen
2. berechne $(20pi)^4 \sin\left(\prod_{n=1}^{8} x_n^n\right)$ für alle Zufallszahlen
3. berechne den Mittelwert $(\bar{x})$ und die Standardabweichen(s);

Die Standardabweichung für den Mittelwert ergibt mit Anwendung des zentralen Grenzwertsatzes durch $\frac{s}{\sqrt{n}}$.

Bei der Implementierung in Matlab hat es sich als zweckmäßig erwiesen sich immer 10^6 Zufallszahlen gleichzeitig zu generieren um den Speicherbedarf begrenzt zu halten.

Die Ergebnisse werden in Tabelle 4.2 dargestellt. Das Ergebnis kann also als 2.2 10^4 angegeben werden.

5 Aufgabe 5

Die Lotka-Volterra-Differentialgleichungen

$$x' = (1 - by)x, \qquad y' = (-1 + x)y, \qquad t > 0 \tag{2}$$

mit den Anfangswerten $x(0) = 1/5$ und $y(0) = 1/5$ besitzen periodische Lösungen $(x(t), x(t))$. Bestimmen Sie den Parameter $b > 0$ so, dass die Periode $T = 10$ ist.

5.1 Vorbemerkungen

Für die Gleichung ist bekannt, dass sie periodische Lösungen besitzt, und eine Erhaltungsgröße (erstes Integral) hat.

$$\text{erstes Integral } = -x + \log(x) - by + \log(y)$$

Bei festes b kann das Problem leicht mit Hilfe eines Differentialgleichungslösers berechnet und visualisiert werden. Man erkennt rasch, dass es 2 mögliche Werte von b gibt, einen in der Nähe von $b = 0.16$ und einen bei $b = 25$.

5.2 Verwendung von Lösern von Differentialgleichungen

Für jedes feste b kann die Gleichung für $t = 10$ numerisch gelöst werden, etwa mit Hilfe von Runge-Kutta-Verfahren. Wenn man zusätzlich noch die Ableitungen $\partial_b x(10)$ bzw. $\partial_b y(10)$ kennt kann das Problem mit dem Newtonverfahren gelöst werden.

Um dies zu erreichen gehen wir ähnlich vor wie beim Schießverfahren zur Lösung von Randwertproblemen. Wir leiten die Differentialgleichung nach b ab, und erhalten dadurch eine Differentialgleichung für die Ableitung nach b.

Zunächst definieren wir

$$u := \begin{pmatrix} x \\ y \end{pmatrix} \qquad f(u) := \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} (1-by)x \\ (-1+x)y \end{pmatrix} \qquad v := \partial_b u$$

Leitet man nun die Gleichung $u = f(u)$ nach b ab erhält man

$$v' = \partial_b f(u) + \partial_u f(u)v = \begin{pmatrix} -xy \\ 0 \end{pmatrix} + \begin{pmatrix} 1+by & -1+x \\ -bx & x \end{pmatrix} v =: g(u,v)$$

mit der Anfangsbedingung $v(0) = 0$. Es ergibt sich also das Gleichungssystem

$$\begin{aligned} u' &= f(u) \\ v' &= g(u,v) \\ u(0) &= \left(\frac{1}{5}, \frac{1}{5}\right)^T \\ v(0) &= 0 \end{aligned}$$

Dieses lösen wir in Matlab mit dem Löser ode45. Als Bedingung für das Newtonverfahren wählen wir $x(10) - 1/5 = 0$, und als Startwert 25. dadurch erhalten wir als Lösungen:

$$\begin{aligned} b &= 25.439649734309953 \\ x(10,b) &= 0.200000000000003 & \partial_b x(10,b) &= 0.154888212244724 \\ y(10,b) &= 0.199992544157243 & \partial_b y(10,b) &= 0.030310897943013 \end{aligned}$$

Als Abbruchbedingung für das Newtonverfahren wurde gewählt $|x_b(10) - 1/5| < 10^{-12}$ und als relativer Fehler für die Funktion ode45 wurde 10^{-12} verwendet. In Abbildung 3 ist die Lösung der Differentialgleichung und die Bahnkurve dargestellt.

Zum einen sehen wir, dass es gut war nach $x(0,b) = x(10,b)$ zu suchen statt nach $y(0,b) = y(10,b)$, weil die Ableitung nach b größer ist. Zum anderen sehen wir aber, dass es nur schlecht gelungen ist die Periodizität aufrecht zu erhalten. Das liegt daran,

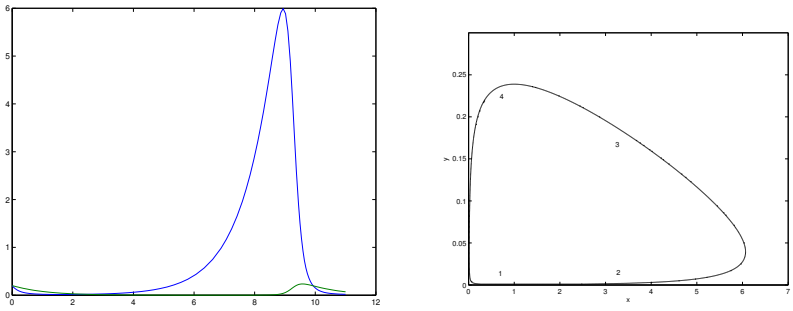


Abbildung 3: Lösung der Differentialgleichung und Bahnkurve für $b = 25.4$

dass sich der Fehler von der Lipschitzkonstanten L abhängt und sich durch err $\leq$ tol e^{Lt} abschätzen lässt.

Vergleicht man das Ergebnis mit dem unter Verwendung des MatLab-Lösers ode113 (Ergebnis: $b = 25.439019269142964$), so sieht man, dass nur 5 Stellen übereinstimmen.

Um den Fehler abschätzen zu können muss noch die Lipschitzkonstante bestimmt werden. Aus Abbildung 3 sieht man, dass $x \leq 7$ und $y \leq 0.3$. Es gilt

$$\begin{aligned} \|f(u)\|_2^2 &= (1-by)^2x^2 + (-1+x)^2y^2 \\ &\leq \underbrace{\max\{(1-by)^2, (-1+x)^2\}}_{:=L^2} \|u\|_2 \\ L &\leq \max\{6.7, 6\} \leq 6.7 \end{aligned}$$

Damit gilt e^{Lt} ist ungefähr 10^{29}. Da die Fehlerschätzung sehr grob ist, besteht zwar die Hoffnung, dass einige Stellen richtig sind, es kann aber nicht ausgesagt werden wie viele. Spätere Erbebnisse zeigen, dass 5 Stellen richtig sind.

Um den Fehler zu verringern müsste t verkleinert werden, also das Intervall $[0, 10]$ aufgeteilt werden. Teilt man zum Beispiel in 10 Teilintervalle so gilt $e^{L|t-t_0|} \leq 10^3$. Man würde damit maximal 3 Stellen „verlieren“. Das Problem dieser Vergehensweise ist aber, dass t für jedes Teilintervall bestimmt werden muss. Auf Grund dieser Schwierigkeit wurde zu einem anderen Verfahren gegriffen.

5.3 Verwendung der Kurve im Phasenportrait

Dadurch, dass wir das erste Integral kennen, kennen wir in 2D auch die genaue Strecke, die die Lösung im Phasenportrait zurücklegt, zumindest als implizite Funktion. Zudem kennen wir auch $t_x := \frac{\partial}{\partial x}t$ bzw. $t_y := \frac{\partial}{\partial y}t$.

$$t_x = \frac{1}{x'} = \frac{1}{(1-by)x} \qquad t_y = \frac{1}{y'} = \frac{1}{(-1+x)y}$$

Damit ist es möglich die Zeit zwischen 2 Punkten auf der Kurve zu bestimmen. Am unteren Ast gilt:

$$t(x_1, x_2) = \int_{x_1}^{x_2} t_x \, dx$$

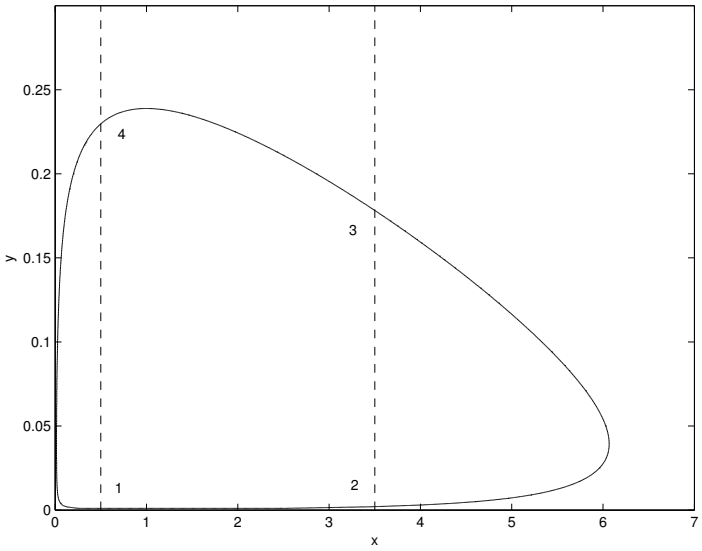


Abbildung 4: Festlegung der Integrationsgrenzen

t_x hängt dabei von y ab. Wenn dieses Integral mittels Quadratur gelöst wird, kann für jeden Quadraturpunkt $y(x)$ mit Hilfe des ersten Integrals und des Newtonverfahrens bestimmt werden.

Es würde also ausreichen 2 Integrale zu lösen. Weil jedoch t_x in der Nähe des minimalen und maximalen x-Wertes gegen unendlich strebt, ist es zweckmäßig in diesen Bereichen nach y zu integrieren.

Um die Periodendauer t für festes b zu bestimmen geht man wie in Algorithmus 1 beschrieben vor.

Algorithmus 1. Bestimme die Konstante c für das 1. Integral mit Hilfe der Anfangswerte.
Bestimme die Funktion des ersten Integrals F und deren Ableitungen F_x und F_y in Abhängigkeit von c.
Bestimme die Punkte $(x_i, y_i)_{i=1}^4$, die als Integrationsgrenzen verwendet werden in 2 Schritten

(i) Lege die Punkte x_i fest. Diese wurden durch $x_1 = x_4 = \frac{1+x_{min}}{2}$ und $x_2 = x_2 = \frac{x_{max}+1}{2}$ festgelegt. x_{max} und x_{min} sind die minimalen und maximalen Werte des Verfahrens aus Kapitel 5.2

(ii) Bestimme y_i mit Hilfe des Newtonverfahrens. Die definierende Gleichung ist $F(x, y) - c = 0$

Bestimme t durch Quadratur der folgende Integrale

$$t_1 := \int_{x_1}^{x_2} \frac{1}{(1-by(x))x} dx$$
$$t_2 := \int_{y_2}^{y_3} \frac{1}{(-1+x(y))y} dy$$
$$t_3 := -\int_{x_4}^{x_3} \frac{1}{(1-by(x))x} dx$$
$$t_4 := -\int_{y_1}^{y_4} \frac{1}{(-1+x(y))y} dy$$
$$t = \sum_{i=1}^{4} t_i$$

Für jeden Quadraturpunkt muss die zweite Variable mittels Newtonverfahren bestimmt werden. Dabei muss ein gedämpftes Newtonverfahren verwendet werden um zu verhindern, dass y kleiner 0 wird und somit das erste Integral F im reellen nicht mehr definiert ist. Die Verwendung des gedämpften Verfahrens ermöglicht es auch sicherzustellen, dass immer der „richtige“ Wert für y verwendet wird. Es muss gelten

$$y < \frac{1}{b} \quad \text{für } t_1 \qquad\qquad x > 1 \quad \text{für } t_2$$
$$y > \frac{1}{b} \quad \text{für } t_3 \qquad\qquad y < 1 \quad \text{für } t_4$$

Um nun die gesuchte Konstante b zu bestimmen muss wieder eine Nullstelle bestimmt werden.

$$t(b) - 10 = 0$$

Um das Newtonverfahren anwenden zu können muss man die Ableitung kennen. Diese kann aus dem oben beschriebenen Verfahren aber nicht erhalten werden. Um das Newtonverfahren trotzdem anwenden zu können approximiert man die Ableitung durch den Differenzenquotienten, man führt ein Sekantenverfahren aus. Dieses konvergiert mit einer Rate von ca. 1.6. Zudem ist die Approximation in der Nähe der gesuchten Nullstelle ungenau, weil es zu Auslöschung kommt. Dies kann zwar die Konvergenzrate negativ beeinflussen, aber nicht die Konvergenz des Verfahrens verhindern. Das Vorgehen wird in Algorithmus 2 beschrieben.

Algorithmus 2. n = 1
while Fehler > Toleranz
Wähle mit dem in Abschnitt 5.2 beschriebenen Verfahren die Startwerte für b und t_b.
Bestimme t mit Algorithmus 1.
Wurden bereits 2 Approximation für b und t berechnet berechne

$$t_b = \frac{t_n - t_{n-1}}{b_n - b_{n-1}};$$

Bestimme die Approximation für b durch

$$b = b - \frac{t - 10}{t_b};$$

Fehler $= |10 - b|$
$n = n + 1$
end while

Dieser Algotithmus wurde in Matlab implementiert und alle Fehlerschranken (Quadratur und verwendete Newtonverfahren) auf 100-fache Maschinengenauigkeit gesetzt. Dadurch ergeben sich die folgenden Ergebnisse

$$b = \mathbf{25.43905182026}156$$
$$b = \mathbf{0.1621732203771}028$$

Auch in diesem Fall ist nicht genau geklärt wie viele Stellen wirklich richtig sind. Da es aber in keinem Schritt zu einer wesentlichen Fehlerverstärkung kommt werden die ersten 13 Stellen als richtig angesehen.

NUMERICAL DIGIT CHALLENGE

FLORIAN KAIN, GREGOR MITSCHA-EIBL

Beispiel 1

Gregor Mitscha-Eibl

Aufgabenstellung, Erste Näherung mit Partialsummmen

Sei $a(n) = \frac{1}{n}$, falls $n \in \mathbb{N}$ eine Primzahl ist und *nicht* die Ziffer 1 enthält, und 0 sonst. Gesucht ist der Wert

$$S = \sum_{n=1}^{\infty} a(n).$$

Zunächst bemerken wir, dass die Reihe konvergiert, weil es sich um eine Teilsumme der Kempner-Reihe K_1 handelt, welche nicht nur aus Primzahlen, sondern allen Zahlen ohne 1 besteht. Bekanntermaßen gilt ungefähr $K_1 = 16.179$, womit wir bereits eine erste obere Schranke für S gefunden hätten.

Eine untere Schranke sowie eine erste Näherung bekommen wir, indem wir die Partialsummen S_N für möglichst hohe N berechnen. Dies wurde in Octave realisiert, indem zunächst die nicht in der Summe vorkommenden Zahlen mit Erathostenes sowie einer ähnlichen Methode zum Streichen der 1er ausgesiebt wurden. Der zugehörige Code *ssum.m* ist am Ende des Kapitels zu finden, dort wird auch der Algorithmus näher erläutert. Bis $N = 10^8$ schafft es das Programm noch in "benutzerfreundlicher" Zeit (also im Minutenbereich), für die nachfolgenden Werte bis 10^{10} musste der PC über Nacht laufen. Da beim Summieren positiver Zahlen in dieser Größe praktisch keine Rundungsfehler auftreten, können sie für unsere Zwecke als exakt angenommen werden:

	S_{10^N}	ΔS_{10^N}
N=1	1.1762	
2	1.4364	2.6021e-001
3	1.6065	1.7006e-001
4	1.7233	1.1685e-001
5	1.8046	8.1319e-002
6	1.8647	6.0103e-002
7	1.9108	4.6015e-002
8	1.9468	3.6005e-002
9	1.9754	2.8669e-002
10	1.9986	2.3123e-002

Date: 26.Jänner 2011.

Anhand dieser Tabelle sieht man sehr gut, wieso ich dieses Verfahren nur als "erste Näherung" bezeichnet habe. Offenbar konvergiert die Summe äußerst langsam, ihr Verhalten unterscheidet sich nicht wesentlich von der reinen Primzahlsumme inklusive 1er, welche jedoch divergiert. Wir können daher bis jetzt nicht viel mehr sagen als

$$1.998 < S < 16.17.$$

Eine Bessere Näherung

Da wir uns keine Hoffnung machen, durch weiteres Ausreizen der Rechenzeit oder der Effizienz des Algorithmus wesentlich mehr zu erreichen, werden wir versuchen, der Reihe analytisch bzw. durch heuristische Überlegungen auf die Schliche zu kommen.

Für die reziproke Primzahlsumme findet man in der Literatur eine asymptotische Näherung:

$$\sum_{p\leq n} \frac{1}{p} = \log\log n + M + o(1) \tag{0.1}$$

wobei $M \doteq 0.26149$ die sogenannte Mertens-Konstante ist. (Auf den Fehlerterm werde ich noch zu sprechen kommen.)

Wenn wir die eine Art Dichte w_p für die Wahrscheinlichkeit kennen, dass p keine 1 enthält, so könnten wir S annähern durch einen Ausdruck der Form $\sum \frac{w_p}{p}$. Diese Wahrscheinlichkeit hängt natürlich von der Anzahl der Dezimalziffern ab, also davon, in welchem Intervall $\{10^k, ..., 10^{k+1} - 1\}$ wir uns befinden. Wir betrachten die Summe also über jedes dieser Intervalle separat. Um Zahlen mit 1 als erster Ziffer von vornherein auszuschließen, lassen wir die Teilsummen immer bei 2.10^k starten. Da diese im weiteren Verlauf immer wieder gebraucht wird, möchte ich dafür eine Kurzschreibweise einführen:

$$\Sigma_{k,p} := \sum_{p=2.10^k, p\in\mathbb{P}}^{10^{k+1}-1} \frac{1}{p}$$

Analog dazu verwende ich $\Sigma^1_{k,p}$,wenn ich darzustellen wünsche, dass die Summe nur Zahlen ohne 1er durchläuft.

Es gibt, wie man sich überlegen kann, 9^k Zahlen ohne 1 zwischen 0 und 10^k, und folglich auch in jedem Intervall $\{j\cdot 10^k, \ldots, (j+1)\cdot 10^k - 1\}$ für $j = 2, \ldots, 9$. Daraus ergibt sich naiv die Vermutung:

$$\Sigma^1_{k,p} \sim \left(\frac{9}{10}\right)^k \Sigma_{k,p}$$

Nach einigen Experimenten wird klar, dass diese Behauptung noch präzisiert werden muss, aber schon in die richtige Richtung geht. Denn wenn wir den Quotienten $r_k := \Sigma^1_{k,p}/\Sigma_{k,p}$ für kleine k errechnen, beobachten wir tatsächlich $r_k \approx \frac{9}{10} r_{k-1}$. Setzen wir also $R_k := \left(\frac{10}{9}\right)^k r_k$, erhalten wir eine Folge, die sich sehr schnell in der Nähe eines konstanten Wertes einzupendeln scheint, $R_k \approx R$.

	r_k/r_{k-1}	R_k	$\lvert 1-\frac{R_{k-1}}{R_{10}}\rvert$
$k=2$	0.74916	0.83240	2.1403e-001
3	0.91151	0.84304	1.0561e-002
4	0.89771	0.84090	2.3479e-002
5	0.88237	0.82443	2.0879e-002
6	0.89816	0.82275	8.8627e-004
7	0.90071	0.82340	1.1573e-003
8	0.90008	0.82347	3.7209e-004
9	0.90038	0.82381	2.8453e-004
10	0.89988	0.82370	1.3472e-004

Damit und mit (0.1) erhalten wir asymptotisch

$$\Sigma^1_{k,p} = R_k \left(\frac{9}{10}\right)^k \Sigma_{k,p} \sim R \left(\frac{9}{10}\right)^k \Sigma_{k,p} \sim R \left(\frac{9}{10}\right)^k (\log\log 10^{k+1} - \log\log 2\cdot 10^k)$$

Wir vereinfachen den Ausdruck rechts, mit $\beta := \log_{10} 2$, zu

$$\log\log 10^{k+1} - \log\log 2\cdot 10^k = \log\left(\frac{(k+1)\log 10}{\log 2 + k\log 10}\right) = \log\left(1+\frac{1-\beta}{k+\beta}\right)$$

Wir fügen alles zusammen und und gelangen zur neuen Näherungsformel für S:

$$S \approx S_{10^n} + R\sum_{k\geq n}\left(\frac{9}{10}\right)^k \log\left(1+\frac{1-\beta}{k+\beta}\right) \tag{0.2}$$

Jetzt sind wir glücklich. Denn wir haben eine Darstellung für den Reihenrest gefunden, die nicht nur sehr einfach und auf beliebige Genauigkeit zu berechnen ist, sondern auch noch erstaunlich gute Ergebnisse liefert.

In der Umsetzung variieren wir die Formel, indem wir R durch (bereits berechnete) empirische Werte R_j ersetzen, und jeweils $j, n = 1, \ldots, 10$ laufen lassen. Die unendliche Reihe ersetzen wir durch eine endliche, bis m laufende, wobei es kaum merkbare Unterschiede macht, ob wir $m = 100$, 1000 oder 10000 wählen, weder was die benötigte Zeit noch was das Rechenergebnis anbelangt. Das zugehörige Programm *ssum2.m* findet ihr am Ende des Kapitels.

In Zahlen schaut das dann so aus: (Man beachte: Je weiter unten und rechts auf der Tabelle, desto aussagekräftiger werden die ersten Ziffern.)

	R_5	R_6	R_7	R_8	R_9	R_{10}	ΔR_{10}
n=1	2.184032	2.181974	2.182765	2.182853	2.183275	2.183139	
2	2.125196	2.123790	2.124330	2.124391	2.124679	2.124586	5.8553e-002
3	2.118126	2.117081	2.117483	2.117527	2.117742	2.117673	6.9135e-003
4	2.119544	2.118735	2.119046	2.119080	2.119246	2.119193	1.5200e-003
5	2.119411	2.118768	2.119015	2.119043	2.119175	2.119132	6.0634e-005
6	2.119218	2.118698	2.118898	2.118920	2.119027	2.118992	1.3986e-004
7	2.119142	2.118717	2.118880	2.118898	2.118986	2.118958	3.4703e-005
8	2.119096	2.118744	2.118879	2.118894	2.118967	2.118943	1.4270e-005
9	2.119074	2.118781	2.118893	2.118906	2.118966	2.118947	3.4175e-006
10	2.119054	2.118807	2.118902	2.118913	2.118963	2.118947	2.4586e-008

Unter dem Eindruck dieser Tabelle werden mir wohl wenige Menschen widersprechen wollen, dass die ersten 5 Ziffern der Summe gefunden sind. Oder?

$$S \doteq 2.1189$$

Warum ich nicht weiss und dennoch zu glauben wage

Gehen wir nochmal einen Schritt zurück. Wir haben für unsere Formel zwei asymptotische Näherungen verwendet.

Die erste ist die Darstellung der Primzahlsumme. Hier ist die Konvergenz gesichert, es gibt jedoch nur schlechte Fehlerabschätzungen. Die beste derzeit bekannte ist $O(e^{-C\sqrt{\log N}})$ mit $C > 0$. Das hilft vorerst wenig weiter, wenn es darum geht, die Korrektheit unserer Ziffern zu verifizieren, da wir nichts über die Konstanten wissen. Ich habe es trotzdem einmal probiert, habe C und die O-Konstante gleich 1 gesetzt, und S mit der Näherung aus (0.2) verglichen, welche wir mit $\tilde{S}_n$ bezeichnen, wobei ich die "besten" verfügbaren Werte $n = 10$ und $R := R_{10}$ gewählt habe. Unter der zusätzlichen Annahme $|R - R_k| \leq 0.01$ für alle $k \geq 10$ liefert eine Rechnung

$$|S - \tilde{S}_{10}| \leq \sum_{k \geq 10} \left(\frac{9}{10}\right)^k \left(R \left|\Sigma_{k,p} - \log \frac{k+1}{k+\beta}\right| + |R - R_k| \log \frac{k+1}{k+\beta}\right)$$

$$\left|\Sigma_{k,p} - \log \frac{k+1}{k+\beta}\right| \leq e^{-\sqrt{k \log 10}} + e^{-\sqrt{(k+1)\log 10}} \leq$$

$$\leq \frac{e^{-\sqrt{10 \log 10}} + e^{-\sqrt{11 \log 10}}}{\log \frac{11}{10+\beta}} \log \frac{k+1}{k+\beta} \doteq 0.22 \log \frac{k+1}{k+\beta}$$

$$|S - \tilde{S}_{10}| \leq \sum_{k \geq 10} \left(\frac{9}{10}\right)^k \log \frac{k+1}{k+\beta} (0.22R + 0.01) = (0.22 + \frac{0.01}{R}) |\tilde{S}_{10} - S_{10^{10}}| \doteq 0.03$$

Die Zahl selber hat keinerlei Aussagekraft, aber die Art wie sie zustande gekommen ist. Von der 2. auf die 3. Zeile habe ich ausgenutzt, dass die Funkion $(e^{-C\sqrt{x}} / \log \frac{x+1}{x+\beta})$ monoton fallend ist. Dies gilt bei $C < 1$ unter Umständen erst für sehr große x, aber die uns zur Verfügung stehenden Daten legen andererseits nahe, dass $\tau(N) := |\sum_{p \leq N} \frac{1}{p} - \log\log N - M|$ *wesentlich schneller* fällt als $e^{-\sqrt{\log N}}$.

	$e^{-\sqrt{\log N}}$	$\tau(N)$
$N = 10$	0.21927	8.0660e-002
10^2	0.11695	1.4140e-002
10^3	0.07220	3.9381e-003
10^4	0.04808	1.2359e-003
10^5	0.03360	3.0460e-004
10^6	0.02430	3.8972e-005
10^7	0.01804	9.5728e-006

Wenn wir also für die Fehlerfunktion τ die gleiche Monotonieeigenschaft annehmen und in der dritten Zeile der Rechnung statt $e^{-\sqrt{10\log 10}}+e^{-\sqrt{11\log 10}}$ den Ausdruck $2\tau(10)\doteq 1.1\cdot 10^{-6}$verwenden, kommen wir auf ein gänzlich anderes Ergebnis:

$$|S-\tilde{S}_{10}|\leq\left(\frac{1.1\cdot 10^{-6}}{0.066}+\frac{|R-R_k|}{R}\right)|\tilde{S}_{10}-S_{10^{10}}|\leq\left(2\cdot 10^{-5}+|1-\frac{R_k}{R}|\right)0.15$$

Der Fehler wird nun vom Ausdruck $|1-\frac{R_k}{R_{10}}|$ dominiert. Was können wir darüber sagen? Die Daten in Tabelle 2 suggerieren, dass er auch für $k\geq 10$ die 10^{-3} nicht mehr übersteigen wird. Das würde uns schon eine schöne Fehlerschranke geben, und implizieren, dass $\tilde{S}_{10}\doteq 2.118946$ um weniger als $1.5\cdot 10^{-4}$ von S abweicht, also $S\in 2.11^{909}_{879}$. (Obwohl unsere Abschätzungen bei weitem nicht alle scharf waren.)

Doch vielleicht lässt sich sogar beweisen, dass R_k konvergiert? Im Rahmen dieses Textes kann ich nicht mehr anbieten als eine heuristische Begründung, warum so ein Beweis existieren könnte: Nehmen wir an es gibt ein c, sodass asymptotisch gilt

$$r_k=\frac{\Sigma^1_{k,p}}{\Sigma_{k,p}}\sim c\frac{\Sigma^1_{k,n}}{\Sigma_{k,n}}.$$

(Das n in den Abkürzungen $\Sigma_{k,n}$ und $\Sigma^1_{k,n}$ soll anzeigen, dass nicht nur Primzahlen, sondern alle natürlichen Zahlen durchlaufen werden.) Das klingt plausibel und könnte vielleicht mit Hilfe von Resultaten bewiesen werden, welche die Primzahlen mit Benford's Law in Zusammenhang bringen. Dann folgt weiter

$$\Sigma_{k,n}\sim\log 10^{n+1}-\log 10^n=\log 10\sim\Sigma_{k-1,n}$$

$$\Sigma^1_{k,n}=\sum_{j=2.10^{k-1}}^{10^k}\left(\frac{1}{10j}+\sum_{d=2}^{9}\frac{1}{10j+d}\right)=\sum_{j=2.10^{k-1}}^{10^k}\frac{1}{10j}+\sum_{d=2}^{9}\frac{1}{10j}-\frac{d}{10j(10j+d)}=$$

$$\frac{9}{10}\sum_{j=2.10^{k-1}}^{10^k}\frac{1}{j}+O(\frac{1}{j^2})=\frac{9}{10}\Sigma^1_{k-1,n}+O(\frac{10^k}{10^{2(k-1)}})\sim\frac{9}{10}\Sigma^1_{k-1,n}$$

Die Summe ist so zu verstehen, dass j nur Zahlen ohne 1 durchläuft. Insgesamt gilt also unter obiger Annahme

$$r_k\sim c\frac{\Sigma^1_{k,n}}{\Sigma_{k,n}}\sim c\frac{9}{10}\frac{\Sigma^1_{k-1,n}}{\Sigma_{k-1,n}}\sim\frac{9}{10}r_{k-1}$$

$$R_k=\left(\frac{10}{9}\right)^k r_k\sim\left(\frac{10}{9}\right)^{k-1}r_{k-1}\sim R_{k-1}$$

Also $\frac{R_k}{R_{k-1}}\to 1$. (Das impliziert zwar noch nicht Konvergenz, legt sie aber nahe.)

Algorithm 1 ssum.m

```
function S = ssum(a,b,k)
% berechnet summe von a bis b (beides natürliche zahlen) indem
% es das intervall in teilintervalle der länge 10^k unterteilt und
% einen vektorisierten siebalgorithmus verwendet.
% (zahlen werden durch wahrheitswerte repräsentiert)

  N = ceil(sqrt(b));
  S = 0;

  p = true(1,N);
  p(1) = false; % 1 ist keine primzahl
  for j = 2:sqrt(N)
    if(p(j))
      p(2*j:j:end) = false;
    end
  end

  N=max(10^k,N);
  for n = a:N:b
    S = S + prime_dec_sieve(n,min(n+N-1,b),p);
  end
end

function S = prime_dec_sieve(a,b,p)
% p ist vektor mit primzahlen die zum sieben verwendet werden

  N = length(p);
  p1 = p(a:end);
  a0 = max(a,N+1);
  p2 = true(1,b-a0+1); %p2 beginnt später als p

  % streichen der nicht-primzahlen
  for j = 1:N
    if(p(j))
      a1 = find(mod(a0:a0+j-1,j)==0);
      p2(a1:j:end) = false;
    end
  end

  p2 = [p1 p2];

  % streichen der 1er
  for k = 10.^(0:floor(log10(b))) % k = 1, 10, 100,...
    tmp = mod(a:b,k*10);
    p2(find((tmp >= k) & (tmp < 2*k))) = 0;
  end

  S = sum(1 ./ (find(p2)+a-1));
end
```

Algorithm 2 ssum2.m

```
function s = ssum2(k,n,m)
% realisiert die Näherungsformel für S

  s = load("ssum_results.m")(k);
  r = load("srel.m")(n);

  a = 9/10;
  b = log10(2);
  R = r*a^(-n+1);

  s1 = sum( a.^(k:m) .* log(1 + (1-b)./((k:m)+b)) );
  s = s + R*s1;
end
```

Beispiel 2

Florian Kain

Aufgabenstellung

Wir sollen das Verhalten der stochastische Folge

$$x_{n+1} = x_n \pm \frac{3}{5} x_{n-1} \qquad n \in \mathbb{N}, \quad x_1 = x_2 = 1 \tag{0.3}$$

untersuchen. Genauer gesagt wollen einen Ausdruck für

$$J := \lim_{n\to\infty} \sqrt[n]{|x_n|} \tag{0.4}$$

finden. Ein sinnvoller Grenzwertbegriff ist zum Beispiel

Definition 1. Eine Folge $(X_n)_{n\in\mathbb{N}}$ von Zufallsvariablen konvergiert genau dann fast überall gegen X, wenn gilt

$$\left(\lim_{n\to\infty} X_n = X\right) = P\left(\{\omega \in \Omega \mid \lim_{n\to\infty} X_n(\omega) = X(\omega)\}\right) = 1 \tag{0.5}$$

Wir schreiben dafür auch$X_n \xrightarrow{f.s.} X$.

Es gibt auch andere, schwächere Grenzwertbegriffe, dennoch sind wir der Meinung, dass dieser ausreichen wird. Diese Vermutung wird unterstützt von den folgenden Beobachtungen:

Ein kleiner Trick und eine erste Abschätzung

Als erste wollen wir uns Gedanken machen ob (0.4) für alle X_n überhaupt existiert.

Lemma 2. *Sei $(a_n)_{n\in\mathbb{N}}$ gegeben durch*

$$a_{n+1} = a_n + \frac{3}{5} a_{n-1} \qquad n \in \mathbb{N}, \quad A_1 = A_2 = 1, \tag{0.6}$$

dann gilt für jede Folge X_n aus (0.3)

$$|X_n| \leq a_n, \forall n \in \mathbb{N} \tag{0.7}$$

Proof. Aus der Dreiecksungleichung und folgt

$$|X_{n+1}| \leq |X_n| + \frac{3}{5}|X_{n-1}| \qquad n \in \mathbb{N}$$

Wir zeigen das Lemma durch Induktion. für den Fall $n = 3$ ist$X_3 \in \{\frac{2}{5}, \frac{8}{5}\}$. Sei nun Die Induktionsvoraussetzung für beliebiges n gegeben:

$$|X_i| \leq a_i, \qquad \forall i \leq n$$

Dann folgt daraus

$$|X_{n+1}| \leq |X_n| + \frac{3}{5}|X_{n-1}| \leq a_n + \frac{3}{5} a_{n-1} = a_n$$

□

die Folge $(a_n)_{n\in\mathbb{N}}$ mag auf den ersten Blick nicht sehr Handlich aussehen. Mit einer geschickten Dartellung der Rekursion hingegen können wir den Grenzwert exakt berechnen.

Definition 3. Sei $(X_n)_{n\in\mathbb{N}}$ wie in (0.3), dann definieren wir

$$y^{(n)} := \begin{pmatrix} X_n \\ X_{n+1} \end{pmatrix}, \quad A := \begin{pmatrix} 0 & 1 \\ \frac{3}{5} & 1 \end{pmatrix}, \quad B := \begin{pmatrix} 0 & 1 \\ -\frac{3}{5} & 1 \end{pmatrix} \tag{0.8}$$

durch die besondere Wahl gilt entweder $y^{(n+1)} = A \cdot y^{(n)}$ oder$y^{(n+1)} = B \cdot y^{(n)}$. Insbesondere gilt

$$\begin{pmatrix} a_{n-1} \\ a_n \end{pmatrix} = A^{n-2} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \tag{0.9}$$

aus den Rechenregeln für $\lim_{n\to\infty}$ und $\|\cdot\|_x$ ergibt sich eine wichtige Eigenschaft der Spektralnorm die auch bei der Vektoriteration verwendet wird

Lemma 4. $\lim_{n\to\infty} \sqrt[n]{|a_n|} = \lim_{n\to\infty} \sqrt[n]{\|A^n a^{(1)}\|} = \|A\|_{spec} = \max_{\lambda\in spec\{A\}} |\lambda| \approx 1.421954$

daraus folgt mit Lemma 0.7

$$0 \leq \lim_{n\to\infty} \sqrt[n]{|X_n|} \leq \frac{1}{2} + \sqrt{\frac{17}{20}} \approx 1.421954 \tag{0.10}$$

Realisierung

Ausfürliches testen für verschiedene Realisierungen von J der Form: "Bilde a Zufallszahlen $\sqrt[n]{X_n}$ und errechne das arithmetische Mittel haben ergeben, dass sich bei $a \approx n$ dieses praktisch immer in der Gegend von 0.97zu finden war. Nehmen wir an, dass $P[X_i \leq x \leq X_{i+1}] = \frac{1}{a}$, wobei $X_{i\leq a}$unsere gewonnen sortierten Zufallszahlen sind, so erweckt die Grafik den Anschein, dass die $\sqrt[n]{|X_n|}$normalverteilt sind. Bei höherem a und n schnürt sich die Verteilung immer weiter zusammen um den vermuteten Erwartungswert bei 0.97. Dies lässt vermuten dass die Varianz gegen 0 konvergiert. Angenommen die Varianz der Verteilung von $\sqrt[n]{X_n} \to 0$, dann haben wir einen Grenzwertbegriff im Sinne von (0.7). Da das arithmetische Mittel gegen den Erwartungswert der Verteilung konvergiert, welcher bei uns die Lösung von J ist,können wir Jdamit aproximieren a oder n groß wählt, da beide Konvergenz garantieren. allerdings erreichen wir die schnellste Berechnung wenn wir a=n wählen. wir erhalten dafür **0.970**165wobei die fettgedruckten Zahlen relativ sicher sind.

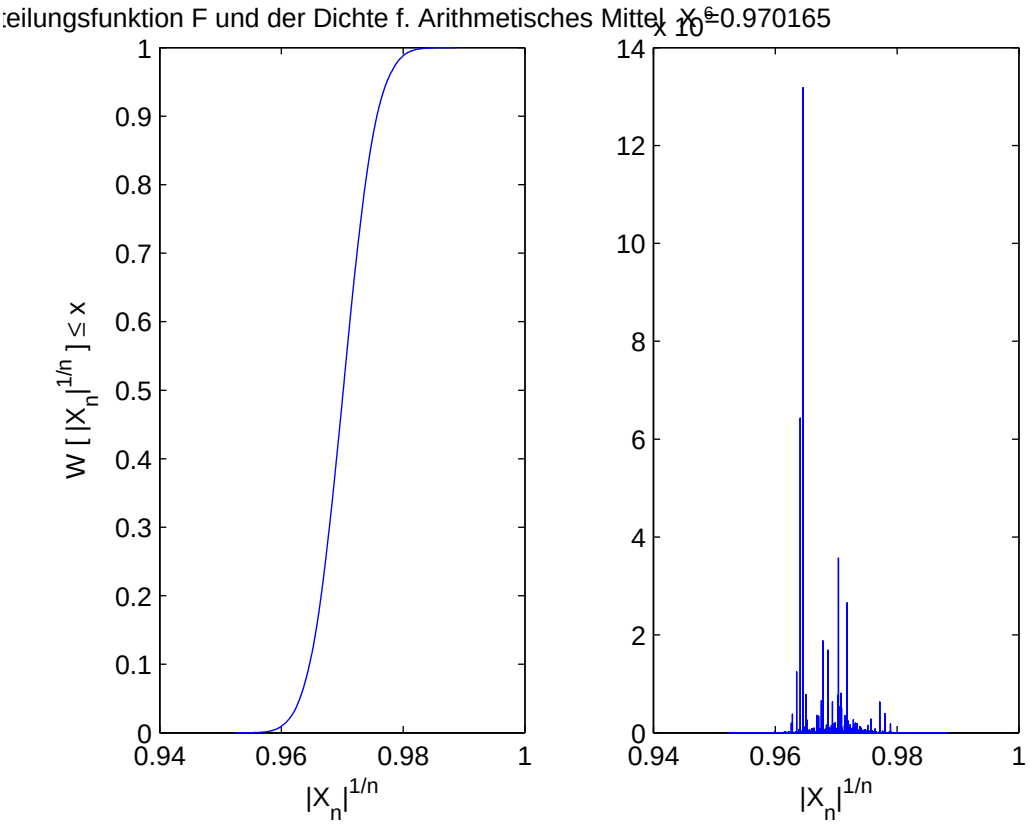


FIGURE 0.1

Beispiel 3

Gregor Mitscha-Eibl

AUFGABENSTELLUNG UND LÖSUNG DURCH "NAIVE METHODE"

Berechne

$$I = \int_0^1 \sin(e^{\frac{1}{x}} \cos(x))dx.$$

(Anmerkung zur Schreibweise: Da der Integrand zwar nicht stetig, aber meßbar und beschränkt ist, existiert das zugehörige Lebesgue-Integral und ist gleich dem Limes in der Angabe.)

Wenn man sich den Graph von $\sin(e^{\frac{1}{x}} \cos(x))$ erst einmal bildlich veranschaulicht hat, werden zwei Tatsachen sehr schnell offenbar:

1. Standard-Quadraturverfahren, insbesondere auch adaptive Verfahren, welche die spezielle Verteilung der Nullstellen der Funktion nicht berücksichtigen, werden an der exponentiell schneller werdenden Oszillation am linken Intervallrand scheitern, und

```
% generiert über Pseudo-Zufallszahlen:
%  a verschiedene stochastische folgen der form (x(n)^1/n
% x(n)=x(n-1) +(2*Y-1)* x(n-2)  , Y~ Bernoulliverteilt mit p=q=1/2
% (2*Y-1) liefert +1 und -1 mit W[2*Y-1=1]=W[2*Y-1=-1]=1/2

function re = test(n,a)
x= ones(2,1);
tmp= 1;
re=zeros(a,1);
test=zeros(n-1,1);
for j=1:a
        for i=1:n-1
                if (floor(0 + (2).*rand(1)))
         k=1;
                else
                        k=-1;
                end
        tmp=x(2);
                x(2)=x(2)+ k*3/5*x(1);
                x(1)=tmp;
        test(i)=abs(x(2))^(1/i);
    end
        plot(test);
        re(j)= abs(x(2))^(1/n);
        x=ones(2,1);
        end
F= (0:1/a:1-1/a)'; % Verteilungsfunktions-werte der Stützstellen
S=sort(re); %Stützstellen
f=zeros(1,a)'; %durch Fs einfachen Diffquotien approximierte Dichte
for t=1:a-1
        f(t)=1/(a*(S(t+1)-S(t)));
end
re=sum(S*1/a);
y1=sprintf('%d',a);
y2=sprintf('%d',n);
y3=sprintf('%f',re);
xlabel('|X_n|^{1/n}')
subplot(1,2,1) plot(S, F);
xlabel('|X_n|^{1/n}')
ylabel('W [ |X_n|^{1/n} ] \leq x')
%Erwartungswert von F unter der Annahme dass dieser gut
%durch das arithmetische Mittel aproximiert wird

end
```

FIGURE 0.2. Code der Grafik generiert und J aproximiert

2. es bietet sich an, die Quadratur jeweils von Nullstelle zu Nullstelle durchzuführen. Zwischen diesen verhält sich der Integrand nämlich wie eine gewöhnliche Sinusfunktion. Man wird daher mit kostengünstigen, auf das jeweilige Intervall transformierten Quadraturformeln bereits eine hohe Genauigkeit erreichen.

Die große Überraschung für mich bei dieser Aufgabe war, dass diese Methode - nur leicht modifiziert -, die mir von Anfang an als naiv und der Herausforderung nicht gerecht werdend erschien, sich als jene herausgestellt hat, mit der ich das Problem letztlich zweifelsfrei auf 15 Nachkommastellen exakt berechnen konnte. Präzisieren wir die Formulierung:

Es sei $x_0, x_1, x_2, ...$ eine monoton fallende Folge von Knoten mit $x_0 = 1$ und $x_n \to 0$, und h der Integrand, dann gilt bzw. gelte

$$I = \sum_{n=0}^{\infty} \int_{x_n}^{x_{n+1}} h(x) =: \sum_{n=0}^{\infty} I_n \approx \sum_{n \leq N} I_n =: S_N.$$

Die wesentliche Abänderung der ursprünglichen Idee war, dass ich für x_n statt der Nullstellen die Extremstellen verwendet habe. Dies wurde im Seminar als Verbesserung vorgeschlagen, da es zu kleineren Summanden und somit schnellerer Konvergenz der Summe führt.

Implementierung, Ergebnisse

Für den Algorithmus in der ersten Fassung habe ich I_n mit der Octave-Function quad berechnet. Abgesehen davon war die einzige Herausforderung, die Extremstellen, also Lösungen der Gleichung $e^{\frac{1}{x}} \cos x = \pi(k + \frac{1}{2})$, zu bestimmen. Ich hatte allerdings davor für das Bsp. bereits andere Ansätze verfolgt, die einen durch Substitution auf $\mathbb{R}$ transformierten Integranden betrachteten, in welchem die Umkehrfunktion $f(e^{\frac{1}{x}} \cos x) = x$ direkt vorkam. Die Routine zur Auswertung von f habe ich daher gleich weiterverwendet. Aus einigen Stunden Rechenaufwand ist alsdann die nachfolgende Tabelle hervorgegangen, die man bereits als akzeptable Lösung der Aufgabenstellung betrachten kann. In der linken Spalte finden sich Werte von S_N für $N = 2, 4, 8, ..., 2^{20}$ Teilintervalle. Die mittlere Spalte ist das Ergebnis von Konvergenzbeschleunigung mit dem Aitken-Verfahren und die rechte das einer Richardson-Extrapolation mit "Pseudo-Stützstellen" $2^{-2}, 2^{-4}, ..., 2^{-2n}$.

	S_N	aitken(S_N)	neville(S_N)
2^1	0.1421967381203933		0.1421967381203933
2^2	0.1281746315224595		0.1235005959898149
2^3	0.1264798246770226	0.1262468158023290	0.1260758419333478
2^4	0.1262570775785977	0.1262233722415980	0.1262026729120931
2^5	0.1262237481475801	0.1262178835891830	0.1262148945797607
2^6	0.1262182267585808	0.1262171304664916	0.1262166769549055
	...	...	...
2^{13}	0.1262170158593892	0.1262170158422539	0.1262170158393353
2^{14}	0.1262170158456987	0.1262170158419731	0.1262170158413869
2^{15}	0.1262170158427338	0.1262170158419142	0.1262170158417955
2^{16}	0.1262170158420911	0.1262170158419132	0.1262170158418877
2^{17}	0.1262170158419489	0.1262170158419085	0.1262170158419035
2^{18}	0.1262170158419218	0.1262170158419154	0.1262170158419137
2^{19}	0.1262170158419188	0.1262170158419184	0.1262170158419182
2^{20}	0.1262170158419344	0.1262170158419213	0.1262170158419415

Einige Beobachtungen anhand dieser Tabelle:

- Wir haben augenscheinlich bereits 13 Ziffern in der Hand, an denen nicht mehr zu rütteln ist: $I \doteq 0.1262170158419$.
- Das Aitken-Verfahren liefert alle 13 Ziffern bereits bei $N = 2^{14}$, die anderen beiden bei 2^{17}.
- Die Konvergenz wird gegen Ende durch Rundungsfehler stark abgeschwächt. Insbesondere die letzte Zeile ist ein Ausreißer, die letzten 3 Ziffern stimmen nicht mehr.
- Ignoriert man die erste und die letzte Zeile, so ist S_N monoton fallend und Neville monoton steigend. Aitken findet sich fast immer dazwischen. In der vorletzten Zeile haben sich die 3 Folgen bis auf die letzte Ziffer getroffen.

> Sehr interpretierfreudige Menschen wie ich gehen jetzt mit Sicherheit davon aus, dass die 14. und 15. Ziffer "18" lauten.

Wir können unsere Ergebnisse also offenbar nur mehr verbessern, indem wir mit höherer Genauigkeit rechnen. Dies war komplettes Neuland für mich. Glücklicherweise habe ich nach einiger Zeit des Herumtüftelns herausgefunden, das Octave dieses Feature in dem (noch nicht sehr gut eingebundenen) symbolics-Package zur Verfügung stellt. Ich musste nur alle Programme geringfügig umschreiben, da der Datentyp für hochgenaue Arithmetik ein anderer ist. Nur die quad-Funktion konnte ich nicht mehr nützen, und habe als Ersatz das Programm *gaussquadMP.m* geschrieben. Es führt eine Gaussquadratur der Ordnung 15 aus, indem auf das Referenzintervall [-1,1] transformiert wird. Knoten und Gewichte mit einer Genauigkeit von 25 Ziffern habe ich im Internet gefunden. In allen Programmen wurde ebenfalls mit 25 oder etwas mehr Ziffern gerechnet.

Um uns zu vergewissern, dass die Ordnung für glatte Funktionen ausreicht, um diese Genauigkeit auch tatsächlich weiterzuvererben, eine kleine Sitzung mit Octave: Wir berechnen $\int_0^\pi \sin(x)dx = 2$ mit $d = 50$ Ziffern.

```
octave-3.2.4.exe:302> symbols;
octave-3.2.4.exe:303> digits(50);
octave-3.2.4.exe:304> f = @(x) Sin(x);
octave-3.2.4.exe:305> q = gaussquadMP(f,0,Pi,50);
octave-3.2.4.exe:306> q-2
ans =
7.5071706699835746306709297932428108364057501823702001811 48E-26
```

Wie erwartet, liegt der Fehler im Bereich 10^{-25}. Wir dürfen also davon ausgehen, dass die Genauigkeit, mit der wir I berechnen, von nun an ausschließlich durch den Fehler des Verfahrens beschränkt ist.

In ungefähr 24 Stunden Rechenzeit haben wir es leider wieder nur geschafft, bis 2^{20}, also ca. einer Million Teilintervalle zu kommen. Um dem Leser eine weitere, der ersten sehr ähnliche, Liste langer Zahlen zu ersparen, nur die letzten 3 Resultate:

	S_N	aitken(S_N)	Δaitken(S_N)
2^{18}	0.12621701584191560207	0.12621701584190659389	3.85E-16
2^{19}	0.12621701584190854590	0.12621701584190651689	7.69E-17
2^{20}	0.12621701584190696066	0.12621701584190650132	1.55E-17

Es ist schön zu sehen, wie die Resultate trotz eines anderen Quadraturalgorithmus mit den vorherigen übereinstimmen. Bei der 17. Ziffer haben wir etwas Pech, da sie genau in die Nähe der 0 fällt und uns somit auch Ziffer 16 vermasselt. Andererseits sollte man nicht geizig sein. Dieses Beispiel ist immerhin das einzige, wo wir auf oder auch nur in die Nähe der Maschinengenauigkeit gekommen sind. Also verkünden wir stolz die 15 gesicherten Ziffern:

$$S \doteq 0.126217015841906$$

Da die verwendeten Algorithmen entweder trivial oder aber allgemein bekannt sind, unterlasse ich es, den Code hier zu replizieren. Nur *aitkenMP.m* soll hier als Beispiel dafür dienen, wie man das Problem löst, dass Octave nicht mit Vektoren hochgenauer Zahlen arbeiten kann.

Algorithm 3 Arbeiten mit vpas und strings

```
function y = aitkenMP(x)
% x wird als spaltenvektor von strings übergeben (= matrix vom typ char)

        [n m] =size(x);
        y= blanks(m);
        y(1,:)="0";
        y(2,:)="0";
        symbols;
        digits(m);

        for k=1:n-2
                xk = vpa(x(k,:));
                xk1 = vpa(x(k+1,:));
                xk2 = vpa(x(k+2,:));
                if (xk2-2*xk1+xk == 0)
                        y2 = xk;
                else
                        y2 = xk - (xk1-xk)*(xk1-xk)/(xk2-2*xk1+xk);
                end
                s = to_char(y2);
                y(k+2,1:length(s)) = s;
        end
end
```

Beispiel 4

Florian Kain

Aufgabenstellung

gesucht ist folgendes Integral:

(0.11)
$$J := \int_{\mathbb{R}^8} s()e(x)dx := \int_{\mathbb{R}^8} sin\left(\prod_{n=1}^{8} \mid x_n^n \mid\right) exp\left(\frac{-1}{10}\sum_{n=1}^{8} x_n^2\right) dx, \qquad x := (x_1, ...x_8)^T$$

Überlegungen

wir vereinfachen das Integral so gut wie möglich: da es in allen Komponenten von x symetrisch ist gilt

(0.12)
$$J = 2^8 \int_{\mathbb{R}_+^8} s(x)e(x)dx$$

nachdem außerdem bemerken wir noch dass $e(x)$ radial symetrisch ist, d.h. nur von $\parallel x \parallel$abhängt.

Nach dem wir das Integral vereinfacht haben überlegen wir uns, mit welcher Methode wir es aproximieren sollen. In der Literatur wir darauf hingewiesen, dass ab knapp 6 Dimensionen, auf jeden Fall aber bei 8, Interploationsverfahren wie z.B. die summierte Simpsonregel der Monte Carlo Methode unterlegen sind. Dies kann man sich durch folgende Grafik verdeutlichen: .

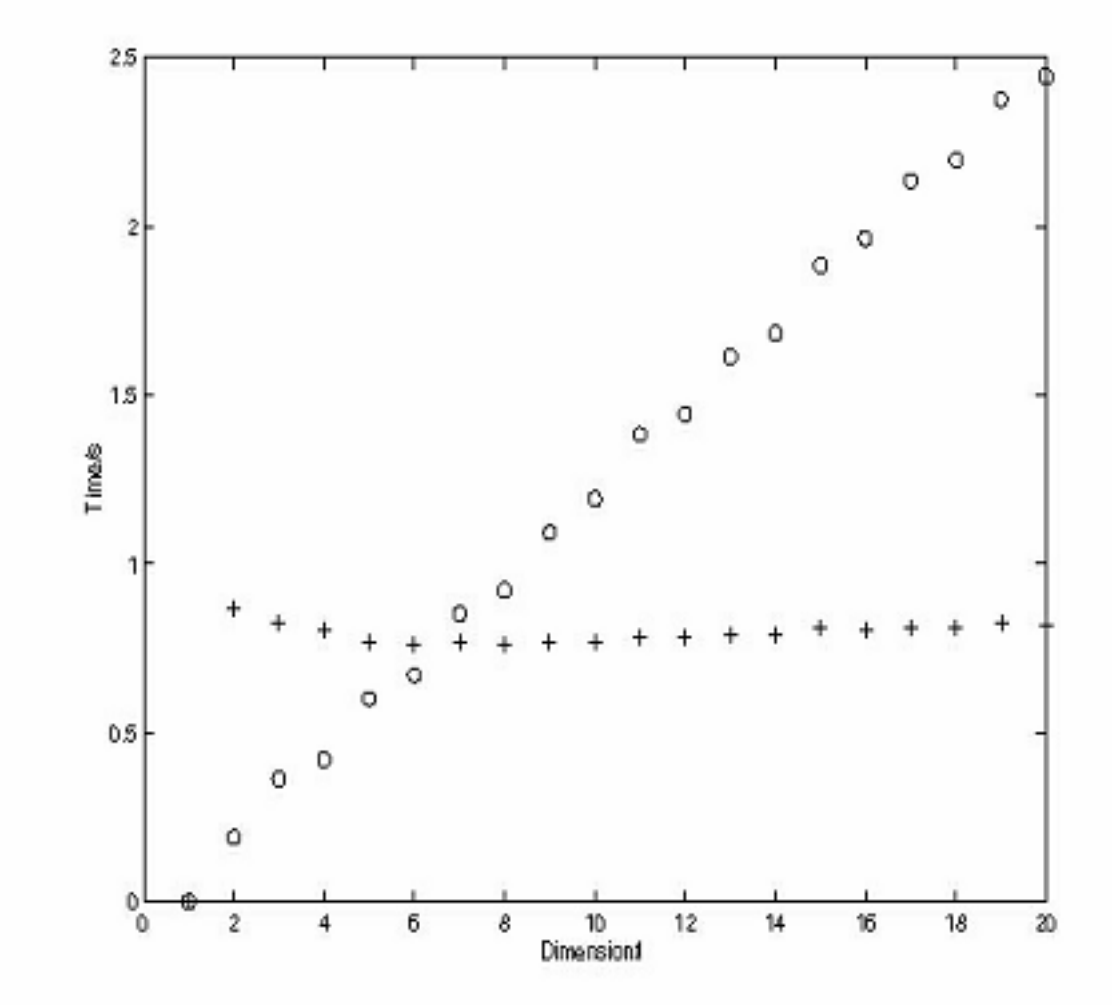


FIGURE 0.3. Vergleich der Laufzeit zur Volumsberechnung einer n-dimensionalen Einheitskugel. (+ Monte Carlo, o Trapezmethode)

MONTE CARLO METHODE

Lemma 5. *Der Zentrale Satz der Monte Carlo Methode*

Sei X eine Zufallsvariable in Ω, $\phi(x)$ eine Wahrscheinlichkeitsdichte und $x_{i\leq N}$ erzeugte Zufallszahlen von x, dann gilt

$$\mathbb{E}\left(f\left(x\right)\right) = \int_{\Omega} f(x)\phi(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \pm \sqrt{\frac{\sigma^2\left(f(x)\right)}{N}} \tag{0.13}$$

0.1. **Monte Carlo mit Gleichverteilung.** Das bedeutet wir können jedes Integral als Erwartungswert einer Funktion $f(x)$ interpretieren, wenn wir einen Faktor

finden der eine Wahrscheinlichkeitsdichte auf Ωbildet. Auf messbarem Ω kann man immer eine Gleichverteilung wählen, da

$$\int_a^b f(x)dx = (b-a)\int_a^b f(x)\frac{1}{b-a}dx = (b-a)\mathbb{E}\left(f\left(x\right)\right)$$

ist dieses Prinzip lässt sich auf mehrdimensionale Mengen erweitern. Wenn wir Ω zb. einen hyperwürfel abstecken, das Integral außerhalb des Gebiets abschätzen und dann MonteCarlo mit Gleichverteilung auf den Integranten $s(x)e(x)$ anwenden. Multiplizieren wir mit dem Maß des Hyperwürfels so erhalten wir eine Näherung für J.

Lemma 6.
(0.14)

$$|\int_{(y,\infty)^8} sin\left(\prod_{n=1}^{8} x_n^n\right) exp\left(\frac{-1}{10}\sum_{n=1}^{8} x_n^2\right) dx\ | \leq 2 \cdot exp\left(\frac{-7y^2}{10}\right)\frac{1}{7!}y^{-28} \qquad y \geq 1$$

Proof. betrachten wir das Integral in x_1

$$exp\left(\frac{-1}{10}\sum_{n=2}^{8} x_n^2\right)\int_{(y,\infty)} sin\left(x_1 z\right) exp\left(\frac{-x_1^2}{10}\right) dx_1\ z = \prod_{n=2}^{8} x_n^n$$

wir substituieren und $t = x_1 z$ und definieren

$$I_1(y,z) := \int_{(yz,\infty)} sin\left(t\right) exp\left(\frac{-t^2}{10z^2}\right) dt$$

betrachten wir dieses Integral nun genauer für $yz = 0$

$$I_1(j\pi,z) = \int_{(j\pi,\infty)} sin\left(t\right) exp\left(\frac{-t^2}{10z^2}\right) dt = \sum_{i=j}^{\infty}\left(-1\right)^i \int_{(0,\pi)} sin\left(t\right) exp\left(\frac{-\left(t+n\pi\right)^2}{10z^2}\right) dt =$$

$$= \int_{(0,\pi)} sin\left(t\right)\sum_{i=j}^{\infty}\left(-1\right)^i exp\left(\frac{-\left(t+n\pi\right)^2}{10z^2}\right) dt$$

fassen wir die Folge $exp\left(\frac{-(t+n\pi)^2}{10z^2}\right) =: (a_i)$ auf, kann man mittels Abschätzungen über geometrischer Reihen zeigen, dass $\sum_{i=j}^{\infty}\left(-1\right)^i a_n \geq 0$ gilt. Außerdem folgt aus der Monotonie von exp, dass$\sum_{i=j}^{\infty} a_{2i} - a_{2i-1} \leq 0$ für j gerade . das bedeutet $0 \leq I_1(j\pi,z) \leq a_j$. analog gilt für j ungerade$0 \geq I_1(j\pi,z) \geq -a_j$ Insbesondere gilt für alle $y,z \geq 0$:

$$0 \leq |\ I_1(y,z)\ | = |\ I_1(\lfloor\frac{y}{\pi}\rfloor\pi,z)\ | \leq a_{\lfloor\frac{y}{\pi}\rfloor} \leq a_0 = \int_{(0,\pi)} sin\left(t\right) exp\left(\frac{-\left(t\right)^2}{10z^2}\right) dt \leq$$

$$\leq \int_{(0,\pi)} sin\left(t\right)\ dt = 2$$

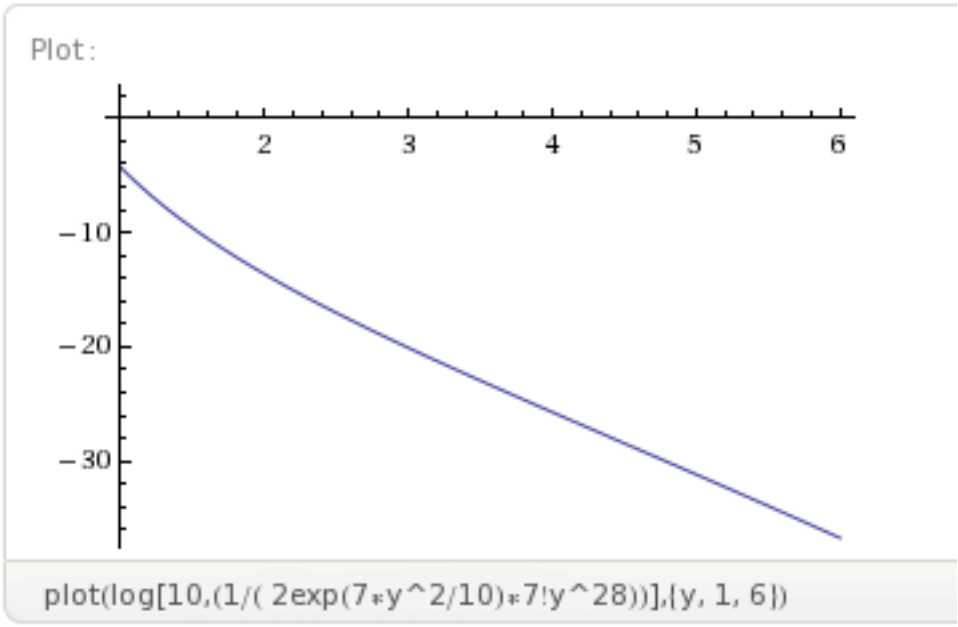


FIGURE 0.4. Konvergenz des Fehler auf der logarithmischen Skala

daraus folgt

$$\left| \int\limits_{(y,\infty)^8} sin\left(\prod_{n=2}^{8} x_n^n\right) exp\left(\frac{-1}{10}\sum_{n=1}^{8} x_n^2\right) dx \right| \leq \int\limits_{(y,\infty)^7} \left(exp\left(\frac{1}{10}\sum_{n=2}^{8} x_n^2\right) z\right)^{-1} \cdot \mid I_1(y,z) \mid dx \leq$$

$$\leq 2 \int\limits_{(y,\infty)^7} exp\left(\frac{-1}{10}\sum_{n=2}^{8} x_n^2\right) \prod_{n=2}^{8} x_n^{-n} \leq 2 \cdot exp\left(\frac{-7y^2}{10}\right) \int\limits_{(y,\infty)^7} \prod_{n=2}^{8} x_n^{-n} =$$

$$= 2 \cdot exp\left(\frac{-7y^2}{10}\right) \prod_{n=1}^{8} \frac{1}{n} y^{-n} \leq 2 \cdot exp\left(\frac{-7y^2}{10}\right) \frac{1}{7!} y^{-28}$$

□

Dadurch können wir den Fehler abschätzen der entsteht wenn wir über das Gebiet $(y,\infty)^8$nicht integrieren. Grafik 0.4 veranschaulicht diesen Fehler. Für y =2 erhalten wir zb 13-fache Genauigkeit. Allerdings gehen und davon noch 3 Stellen verloren da wir noch mit 2^8 multiplizieren müssen. Nun können wir J berechnen mit Lemma (5)

$$J = 2^8 \left(\int\limits_{(0,2)^8} s(x)e(x)dx \pm 10^{-13} \right) \approx 2^{16} \frac{1}{N} \sum_{i=1}^{N} s\left(x_i\right) e\left(x_i\right) \pm 2^8 \sqrt{\frac{\sigma^2\left(s\left(x_i\right) e\left(x_i\right)\right)}{N}} \pm 10^{-10}$$

da

$$0 \leq s\left(x_i\right) e\left(x_i\right) \leq 1$$

gilt auch

$$\sigma^2\left(s\left(x_i\right)e\left(x_i\right)\right)=\mathbb{E}\left(s\left(x_i\right)e\left(x_i\right)\right)^2-\mathbb{E}\left(\left(s\left(x_i\right)e\left(x_i\right)\right)^2\right)\leq 1$$

0.2. **Monte Carlo mit mehrdimensionaler Exponentialverteilung.** Wir können auch aus der besonderen Form von $e(x)$ eine Wahrscheinlichkeitsdichte der 8-dimensionalen Exopnentialverteilung herauslesen. Für diese gilt allgemein:

$$\phi_X(x)=\frac{1}{(2\pi)^{\frac{n}{2}}\mid\Sigma\mid^{\frac{1}{2}}}\cdot exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

wobei n die Diemnsion, Σdie Kovarianzmatrix und$\mid\Sigma\mid$deren Determinante ist. wählen wir $\mu=0$ und$\Sigma=diag(5,...,5)$so ergibt sich

$$J=\int\limits_{\mathbb{R}^8}s()e(x)dx=(2\pi)^4 5^4\int\limits_{\mathbb{R}^8}s(x)exp\left(\frac{-1}{2}x^T\Sigma^{-1}x\right)\frac{1}{(2\pi)^{\frac{n}{2}}\mid\Sigma\mid^{\frac{1}{2}}}dx=(2\pi)^4 5^4\int\limits_{\mathbb{R}^8}s(x)\phi_X(x)dx$$

wobei $X\sim\mathcal{N}_{(0,..0)^T,\Sigma}$8-dimensional normalverteilt ist. Nach dem Lemma 5 folgt

$$(2\pi)^4 5^4\int\limits_{\mathbb{R}^8}s(x)\phi_X(x)dx\approx\frac{(2\pi)^4 5^4}{N}\sum_{i=1}^{N}s(x_i)\pm(2\pi)^4 5^4\sqrt{\frac{\sigma^2\left(f(x)\right)}{N}}\approx\frac{(2\pi)^4 5^4}{N}\sum_{i=1}^{N}s(x_i)\pm(10\pi)^4\sqrt{\frac{2}{N}}$$

man beachte hierbei dass $\sigma^2\left(s(x)\right)=\mathbb{E}\left(s(x)\right)^2-\mathbb{E}\left(s\left(x\right)\right)\leq 2$, da $\mid s(x)\mid\leq 1$.

Realisierung

Wir haben jetzt zwei Verfahren erarbeitet wobei für eine Genauigkeit von 10^k die Anzahl N der Versuche im ersten Fall $N\geq 6,5536\cdot 10^{-2k+4}$ im zweiten Fall $N\geq 1,8977\cdot 10^{-2k+12}$. Allerdings gilt es nun zu entscheiden ob wir eine Gleichverteilung auf $s\left(x_i\right)e\left(x_i\right)$ oder eine Normalverteilung auf $s\left(x_i\right)$. Es gilt also abzuwägen ob die Erzeugung normalverteilter Zufallszahlen weniger kostet als die Berechnung der Exponentialfunktion und einer Multiplikation. Ich denke, dass beide Funktionen in den meisten Umgebungen gleichwertig implementiert sind. Somit zeigt sich, dass für die erst Variante 10^8mal so wenig Zahlen benötigt werden. Allerdings hat sich irgendwo ein fehler eingeschlichen oder die Implementierung ist falsch, da diese nur willkürliche Werte liefert (siehe Anhang). deshalb verwenden wir hier die zweite Version. Hier hatte ich das Problem, dass bei meiner Implementierung schnell die Zählschleife erschöpft war deren Index nur ein Integer sein durfte, ich habe es leider nicht zusammengebracht dieses Problem zubeseitigen in dem ich auf höhere Genauigkeit umsteige. Leider wurden mir meine Programmierkentnisse und die Tatsache, dass ich nur Matlab verwendet habe zum Verhängnis. so konnte ich mit meinem Programm nur eine Genauigkeit von 10^3erreichen. Ich habe dies 1000 mal rechnen lassen und nocheinmal das arithmetische Mittel genommen um vielleicht mehr Zahlen zu erreichen die Ich aber nicht garantieren kann. Eine Näherung für das Integral lautet: $J\approx$ **31**173.88053957139. wobei die fett gedruckten Zahlen fix sind.

```
function res = montecarlo( prec, length )
%precision is 10^prec I=0
for i=1:7*10^(-2*prec+4)
        tmp= length*rand(8,1);
        l= norm(tmp);
        Itmp=1;
        for j=1:8
                Itmp= abs(Itmp*tmp(j)^j);
        end
        Itmp=sin(Itmp)*exp(-sqrt(l)/10);
        I=I+Itmp;
end
res=I/10^(-2*prec+4)*2^8*length^8;
end
```

FIGURE 0.5. Monte Carlo mit Gleichverteilung

```
function res = montecarlo2( prec)
%precision is 10^prec I=0;
for i=1:2*10^(-2*prec+12)
    tmp=  sqrt(5).*randn(8,1);
        Itmp=1;
        for j=1:8
                Itmp= Itmp*tmp(j)^j;
        end
        Itmp=sin(abs(Itmp));
        I=I+Itmp;
end
res=I/10^(-2*prec+12)*(pi*10)^4;
end
```

FIGURE 0.6. Monte Carlo mit Multidimensionaler Normalverteilung

SE Numerical Digit Challenge

Florian Scheweder (e0525265)
Peter Regner (e0525869)

Wintersemester 2010/11

Inhaltsverzeichnis

TU Wien, LVA-Nr 101.413, betreut von Univ.Prof. Dr.rer.nat. A. Jüngel
Die Aufgaben finden sich hier:
http://asc.tuwien.ac.at/~juengel/teaching/2010w/Aufgaben.pdf

> „In diesem Seminar sollen Sie fünf mathematische Probleme lösen, deren Ergebnis jeweils eine reelle Zahl ist. Der Auftrag lautet, diese Zahlen auf möglichst viele Ziffern (ohne Rundung) genau zu berechnen."

Ausgearbeitet wurden die einzelnen Aufgaben von:

1: Peter Regner

2: Florian Scheweder (2.3), Peter Regner (2.1, 2.2)

3: Florian Scheweder, Peter Regner

4: Florian Scheweder, Peter Regner

5: Florian Scheweder

1 Langsam konvergente Reihe

1.1 Problemstellung und die Konvergenz der Reihe

Die gegebene Reihe $\sigma := \sum_{S \cap \mathbb{P}} \frac{1}{n}$ entspricht der harmonischen Reihe $\sum_{n>0} \frac{1}{n}$, bei der einige Summanden weggelassen werden (S bezeichne die Menge aller natürlichen Zahlen ohne der Ziffer 1 in Dezimaldarstellung). Die Divergenz der harmonischen Reihe ist bekannt, insofern ist es als erste Fragestellung naheliegend, ob unsere Reihe σ überhaupt konvergiert – anderfalls wäre eine numerische Berechnung schließlich nicht sinnvoll.

Nun könnte man vermuten, dass es so wenige Primzahlen gibt, dass dies die wesentliche Eigenschaft für die Konvergenz von σ ist, also dass schon $\sum_{p \in \mathbb{P}} \frac{1}{p}$ konvergiert, aber auch diese Reihe divergiert. Weil diese Tatsache doch überraschend und auf alle Fälle irgendwie spannend ist, wollen wir hier einen Beweis von Erdős in einer leicht abgewandelten Form aus [HS07, S. 53ff] wiedergeben.

Theorem 1. $\sum_{p \in \mathbb{P}} \frac{1}{p} = \infty$

Beweis. Wir zeigen indirekt, dass eine Konvergenz der Reihe zu einem Widerspruch führen würde. Wenn der Limes der Teilsummen existiert (und endlich ist), gibt es einen Index i, sodass der Reihenrest

$$\sum_{l>i} \frac{1}{p_l} < \frac{1}{2}, \tag{1.1}$$

wobei p_l die l-te Primzahl bezeichne.

Für ein beliebiges $x \in \mathbb{N}$ betrachte nun die Anzahl der natürlichen Zahlen[1], die kleiner als x sind und nur Primfaktoren aus den ersten i Primzahlen enthalten:

$$N_i(x) = \#\left\{y \in \mathbb{N} \mid y < x, y = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_i^{\alpha_i}, \alpha_j \geq 0\right\}$$

Für ein solches y betrachte die Aufteilung $y = k \cdot m^2$ mit einem quadratfreien $k \in \mathbb{N}$, das heißt $k = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_i^{\beta_i}$ mit $\beta_j \in \{0, 1\}$. Für k gibt es also 2^i verschiedene Möglichkeiten. Für m gibt es wegen $m^2 \leq y < x$ weniger als $\sqrt{x}$ Möglichkeiten. Insgesamt ergibt sich daher $N_i(x) < 2^i \cdot \sqrt{x}$.

Betrachte nun umgekehrt jene $y < x$, die keine Primfaktoren p_l mit $l \leq i$ enthalten:

$$\#\left\{y \in \mathbb{N} \mid 1 < y < x, y = p_{i+1}^{\alpha_{i+1}} \cdot p_{i+2}^{\alpha_{i+2}} \cdot \ldots \cdot p_{i+l}^{\alpha_{i+l}}, \alpha_j \geq 0\right\} = x - N_i(x) \tag{1.2}$$

Für eine fixe Primzahl p gibt es höchstens $\frac{x}{p}$ natürliche Zahlen, die den Primfaktor p enthalten und kleiner als x sind (nämlich $p, 2 \cdot p, 3 \cdot p, \ldots$), also

$$\#\left\{y \in \mathbb{N} \mid y < x, y = a \cdot p\right\} \leq \frac{x}{p}.$$

[1] Hier ist $0 \in \mathbb{N}$.

Für natürliche Zahlen y, die zumindest einen der Primfaktor $p_{i+1}, p_{i+2}, p_{i+3}, \ldots$ enthalten, gilt

$$\#\{y \in \mathbb{N} \mid 1 < y < x, y = a \cdot p_l, l > i\} \leq \frac{x}{p_{i+1}} + \frac{x}{p_{i+2}} + \frac{x}{p_{i+3}} + \cdots$$

Auf der linken Seite steht aber eine Obermenge von der Menge aus (1.2) und mit (1.1) folgt daher

$$x - N_i(x) \leq \frac{x}{p_{i+1}} + \frac{x}{p_{i+2}} + \frac{x}{p_{i+3}} + \cdots < \frac{x}{2}.$$

Wir erhalten also $\frac{x}{2} < N_i(x)$ und haben vorher gezeigt, dass $N_i(x) < 2^i \cdot \sqrt{x}$. x war beliebig, aber für $x > 2^{2i+2}$ erhalten wir einen Widerspruch. □

Die Entscheidende Eigenschaft für die Konvergenz unserer Reihe ist also die andere – dass nur Zahlen ohne Ziffer 1 in der Dezimaldarstellung des Nenners aufsummiert werden. Nach A. J. Kempner, der 1914 derartige Reihe studierte, werden sie Kempner-Reihen genannt – siehe hierzu [Kem14].

Um für unseren speziellen Fall die Konvergenz einzusehen betrachte:

$$\begin{aligned}
\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{9} &< 8 \cdot \frac{1}{1} \\
\frac{1}{20} + \frac{1}{22} + \cdots + \frac{1}{29} + \frac{1}{30} + \cdots + \frac{1}{99} &< 8 \cdot 9 \cdot \frac{1}{10} \\
\frac{1}{200} + \cdots + \frac{1}{209} + \frac{1}{220} + \cdots + \frac{1}{999} &< 8 \cdot 9 \cdot 9 \cdot \frac{1}{10^2} \\
&\vdots
\end{aligned}$$

Somit ergibt sich eine obere Schranke:

$$\sigma := \sum_{S \cap \mathbb{P}} \frac{1}{n} < \sum_{S} \frac{1}{n} < \sum_{n \geq 0} 8 \cdot \left(\frac{9}{10}\right)^n < \frac{8}{1 - \frac{9}{10}} = 80 < \infty$$

Und damit ist die Konvergenz auch für unsere Reihe σ bewiesen.

Allerdings stellt sich die numerische Ermittung des Grenzwerts, als ziemlich schwierig heraus, weil die Reihe sehr langsam konvergiert. Für die Kempnerreihe, bei der Summanden mit Ziffer 9 im Nenner weggelassen werden, ergibt sich ungefähr $22.92067661926415034816\ldots$, aber die Summe der ersten $\approx 10^{28}$ Brüche mit Nenner kleiner als 10^{30} und ohne Ziffer 9 ist noch immer kleiner als 22 (siehe [Bai79]).

1.2 Schranken

Als ersten Schritt bietet sich an, mit dem trivialen Algorithmus der Summation der Kehrwerte der entsprechenden Zahlen eine möglichst gute (aber trotzdem sehr sehr schlechte) untere Schranke zu bestimmen. Wie sich herausstellen wird, ist das aber

auch schon das Beste, was im Rahmen dieses Seminars mit mathematisch exakten Methoden möglich ist.

Nachdem das Ergebnis für die Kempnerreihe bereits aus [Bai79] bekannt ist (siehe (1.3)), lässt sich auch eine obere Schranke bestimmen, indem die Kehrwerte von Nichtprimzahlen ohne Ziffer 1 bis zu einem möglichst großen N aufsummiert werden. Zum Primzahltesten werden wir das Miller-Rabin-Verfahren einsetzen – ein sehr schnelles Verfahren, das mit der Monte-Carlo-Methode arbeitet, also keine sicheren Ergebnisse liefert. Konkret ist das Ergebnis "n ist eine Primzahl" nur sehr wahrscheinlich richtig, aber das Ergebnis "n ist keine Primzahl" ist immer richtig. Die obere Schranke können wir daher ganz ohne Unsicherheit angeben. Die untere Schranke ist nicht ganz sicher, weil vielleicht auch die ein oder andere Nichtprimzahl in den Nenner eines Summanden der Reihe gerutscht ist, aber nachdem die Schranken ohnehin noch sehr weit außeinander liegen, gibt es noch keinen Anlass sich darüber den Kopf zu zerbrechen. Mit dem dem Algorithmus 1 ergibt sich mit Nennern die kleiner als 10^8 sind

$$\begin{aligned} 1.946759 &\overset{?}{<} \sum_{S\cap\mathbb{P}} \frac{1}{n} = \sigma \\ 7.377722 &< \sum_{S\cap\mathbb{P}^c} \frac{1}{n} \\ 16.17696 &\approx \sum_{S} \frac{1}{n} \end{aligned} \tag{1.3}$$

und somit

$$1.946759 \overset{?}{<} \sigma < 8.799239. \tag{1.4}$$

Interessant ist, dass hier viele naheliegende Ideen keinen oder keinen wesentlichen Geschwindigkeitsschub bringen. Mathematische Methoden (zB. mit Modolorechnung) zur Überprüfung, ob die Ziffer 1 im Nenner vorkommt, sind der der Stringsuche von Python ungefähr um einen Faktor 2 in der Laufzeit unterlegen. Ein C-Programm mit der Library GMP hat eine sehr ähnliche Laufzeit und ist deshalb die Mühe nicht wert. Erstaunlicherweise hat sogar dasselbe Programm ohne Primzahltest immer noch ungefähr die Hälfte der Laufzeit. Für die direkte Summation mit der Größordnung an Summanden, für die dies überhaupt in absehbarer Zeit möglich ist, kann man also festhalten, dass das Primzahltesten gar nicht das Hauptproblem ist.

Um 1.4 noch etwas zu verbessern haben wir in 2 das Pythonmodul decimal verwendet, um die Zahlen genauer zu darstellen zu können, und mit das Modul multiprocessing , um die Rechenlast auf alle verfügbaren Prozessorkerne zu verteilen und so die Laufzeit ein wenig zu senken. Das Pythonmodul BigFloat hat sich hier als langsamer erwiesen, weshalb decimal zum Einsatz gekommen ist. Mit ROUND_FLOOR kann sichergestellt werden, dass $\frac{1}{n}$ durch das nächstkleinere decimal-Objekt dargestellt wird. So kann es nicht vorkommen, dass durch Rundungsfehler in der Darstellung der Summanden die Schranke falsch wird.

So erhalten wir nun

Algorithm 1 Pythoncode mit trivialem Algorithmus zur Summation über den Miller-Rabin-Primzahltest

```
#!/usr/bin/env python
import sys
import math
import MillerRabin

# Konstante
N = 10**8     # Nenner sind kleiner als N

summe = long(0)
s = 1
while s <= N:
      if ('1' not in str(s)) and (MillerRabin.miller_rabin(s)):
            summe += 1. / long(s)
      s += 1

      if s % 100000 == 0:
            print 'Summe = ', summe, 'N = ', s

print 'Final result:'
print summe
```

$$2.02645697 \overset{?}{<} \sum_{S\cap\mathbb{P}} \frac{1}{n} = \sigma \tag{1.5}$$

$$8.06878906 < \sum_{S\cap\mathbb{P}^c} \frac{1}{n} \tag{1.6}$$

$$16.17696 \approx \sum_{S} \frac{1}{n}$$

also ergibt sich

$$2.02645697 \overset{?}{<} \sigma < 8.10817094.$$

Dabei wurden für (1.5) alle Summanden mit Nenner kleiner als $5,0664 \cdot 10^{11}$ aufsummiert (das hat auf einem Quadcore Intel Xeon CPU X3430 @ 2.40GHz ungefähr 424 Stunden gedauert) und für i(1.6) alle Summanden mit Nenner kleiner als $2.28 \cdot 10^9$ (in etwas mehr als 11 Stunden auf einem Intel Core 2 Duo CPU E8400 @ 3.00GHz).

1.3 Weitere Ideen

Während es für die Kempnerreihe viel Literatur und einen einen schnellen Algorithmus zur numerischen Bestimmung des Grenzwertes gibt, sind die Methoden nicht

Algorithm 2 Pythoncode zur Summation über den Miller-Rabin-Primzahltest und Unterstützung für Mutlicore

```
#!/usr/bin/env python
import sys
import math
import MillerRabin
import time
import multiprocessing
from decimal import *

# Configuration...
getcontext().prec = 200
getcontext().rounding = ROUND_FLOOR

def n_is_summand(n):
      if ('1' not in str(n)) and (not MillerRabin.miller_rabin(n)):
            return True
      else:
            return False

def partial_sum(list_of_n):
      #return sum( filter(n_is_summand, list_of_n) )
      partialsum = Decimal(0)
      for n in filter(n_is_summand, list_of_n):
            partialsum += 1 / Decimal(n)
      return partialsum

def summitup():
      # Konstante
      n           = 1            # kleinster Nenner (potentiell, dh. nur
          wenn ohne 1 und prim)
      N           = 10**20       # groesster Nenner (potentiell, dh. nur
          wenn ohne 1 und prim)
      per_step  = 10**5     # so viele Summanden - 1 werden immer eine
          Prozessorkern zugeteilt

      pool = multiprocessing.Pool()
      summe = Decimal(0)
      while n <= N:
            number_blocks = []
            for i in range(1,9):
                  number_blocks += [range(n, n + per_step)]
                  n += per_step
            summe += sum( pool.map(partial_sum, number_blocks) )

            print 'Summe = ', summe, ' = 1/2 + ... + 1/n, mit n = ', n, '
                =10**', math.log10(n)
      print 'Final result:'
      print summe
      return 0

def main():
      start = time.time()
      summitup()
      t = time.time() - start
      print 'Runtime:', t, 'sec =', (t / 3600.), 'h'

if __name__ == '__main__':
    main()
```

Algorithm 3 Pythoncode mit rekursivem Algorithmus zur Summation mit dem Miller-Rabin-Primzahltest

```
1 sigma = 0
2 S = range(2,10)
3 while S[-1] <= N:
4       for n in S:
5             if MillerRabin.miller_rabin(n):
6                   sigma += 1. / n
7       S = [10*n + k for n in S for k in ([0] + range(2,10))]
```

einfach für das vorliegende Problem zu verwenden. Es wird dabei ausgenützt, dass sich die Menge S_i der natürlichen Zahlen ohne Ziffer 1 mit i Dezimalstellen rekursiv aus S_{i-1} bilden lässt

$$S_i = \{10 \cdot s + d | d = 0, 2, 3, \ldots, 9; s \in S_{i-1}\}.$$

Unter Verwendung dieses Gesetzes lassen sich Reihen herleiten, die wesentlich schneller konvergieren und so kann der Grenzwert der Kempnerreihe schnell numerisch ermittelt werden.

Es scheint uns aber nicht möglich diese Verfahren für die Primzahleinschränkung zu adaptieren. Bestenfalls kann die Idee wie in Algorithmus (3) verwendet werden. In der Praxis ist dieser Algorithmus aber nicht schneller und stößt schon ab $N = 10^8$ an die Grenzen des Hauptspeichers (2 GB).

Auch wenn sich hier verteilte System gut einsetzen lassen, weil die Aufgabe sich gut aufteilen lässt, wird man auch da mit Brute-force nicht sehr weit kommen. Zielführender dürfte es sein Aussagen über das assymptotische Verhalten von Primzahlen wie den Primzahlsatz oder Satz von Mertens (siehe zB. [Rib04, S. 171])

$$\sum_{p \leq x} \frac{1}{p} = \log\log x + C + O\left(\frac{1}{\log x}\right),$$

wobei $C = 0.2615\ldots$ zu verwenden. Auch Aussagen über die statistische Verteilungen der Ziffern in Primzahlen könnte hier einfließen. Im Rahmen des Seminars sind uns aber leider keine brauchbaren Ergebnisse in der Hinsicht gelungen.

2 Zufällige Fibonaccifolge

Bei diesem Beispiel ist die Erwartungswert $\sigma = \lim_{n\to\infty} |x_n|^{\frac{1}{n}}$ für eine zufällige Fibonaccifolge, die wie folgt definiert ist

$$x_0 = x_1 = 1$$

$$x_{n+1} = x_n \pm \frac{3}{5} \cdot x_{n-1} \qquad n\epsilon\mathbb{N},$$

zu berechnen (dabei wird das Vorzeichen jeweils mit der Wahrscheinlichkeit $\frac{1}{2}$ zufällig gewählt). Für ein beliebiges β wird $\sigma(\beta) = \lim_{n\to\infty} |\tilde{x}_n|^{\frac{1}{n}}$ als Lyapunov-Konstante bezeichnet, wobei

$$\tilde{x}_{n+1} = \tilde{x}_n \pm \beta \cdot \tilde{x}_{n-1} \qquad n\epsilon\mathbb{N}$$

Für unser konkretes $\beta = \frac{3}{5} = 0.6$ wird in [ET98, S. 4] als Ergebnis 0.97017 angebeben. Wir wollen für unseren konkreten Fall nach einigen theoretischen Überlegungen zuerst mit Monte-Carlo-Experimenten die Folge untersuchen und dann das Ergebnis mit der Methode über Zufallsmatrizen herleiten.

2.1 Theoretische Überlegungen

Eine einfache Abschätzung für die betragsmäßig am schnellsten wachsende Folge $\breve{x}_n$ und allen möglichen x_n – und zwar die bei der das Vorzeichen immer positiv oder immer negativ ist – liefert bereits eine obere Schranke:

$$\breve{x}_n = \breve{x}_{n-1} + \frac{3}{5} \cdot \breve{x}_{n-2} < \breve{x}_{n-1} + \breve{x}_{n-2} = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right] < \Phi^{n+1},$$

wobei$\Phi = \frac{1}{2}(1+\sqrt{5}) \approx 1,6180$ den goldenen Schnitt bezeichnet. Weil $\breve{x}_n$ die betragsmäßig am schnellsten wachsende Folge war und aus der Monotie der n-ten Wurzel folgt $\sigma < \lim_{n\to\infty} \Phi^{\frac{1}{n}} \Phi = \Phi$ und es gilt natürlich auch $0 \leq \sigma$.

Dass der Limes nicht für jede Folge gleich ist, sieht man zB. wenn man die beiden Folgen mit abwechselnden Vorzeichen und immer mit positivem Vorzeichen betrachtet. Für erstere ergibt sich ein Grenzwert von ungefähr 0.78 und für zweitere 1.13. In [ET98, S. 3] wird aber angegeben, dass der Grenzwert σ zumindest fast sicher existiert.

2.2 Monte-Carlo-Experimente

In 25833 Sekunden (ungefähr 7 Stunden) war es mit sehr einfachen Mitteln möglich, bereits die ersten 4 Ziffern zu ermitteln. Dazu haben wir mit Octave für $3 \cdot 10^7$ zufällige Folgen x_n den Wert für $x_{5\cdot 10^3}$ bestimmt und dann das arithmetische Mittel gebildet – siehe Algorithmus 4.

Weil in [ET98] angegeben ist, dass die in 2.3 beschriebene Methode den Monte-Carlo-Experimenten überlegen ist, verzichten wir hier auf weitere Überlegungen zu den Monte-Carlo-Experimenten.

Algorithm 4 Annäherung von σ durch das arithmetische Mitteln von zufällig generierten x_N für großes N

```
tic;
N = 3*10^7;
% gemittelt ueber N Werte
Iterations = 5 * 10^3;    % es gibt die Folge bis x_Iterations
sigma = 0.97017;                        % ungefaehres Endergebnis

x1 = ones(N,1);
x2 = ones(N,1);
xn = zeros(Iterations, 1);

for i = 1:Iterations
        sgm = sign(rand(N,1) - 0.5);
        tmp = x2;
        x2 = x2 + (0.6 * sgm) .* x1;
        x1 = tmp;

        if (x2 < 10^250)
                disp('Achtung! Wert zu klein!');
                breakErzeugung von Zufallsfolgen
        end

        xn(i) = sum (abs(x2) .^ (1/i)) / N;
end

plot(1:Iterations, xn, 1:Iterations,
sigma * ones(Iterations,1) )

disp('Ergebnis')
xn(end)
toc
```

2.3 Methode über Zufallsmatrizen

Da das Berechenen vieler und Langer zufälliger Folgen und das anschließende Mitteln über die so erhaltenen "Grenzwerte" sehr Speicheraufwändig sowie sehr ungenau ist schreiben wir die Folge in Vektorform wie in einem Artikel von Viswanath [Vis98] beschrieben :

$$\begin{pmatrix} x_n \\ x_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ \pm\beta & 1 \end{pmatrix} \begin{pmatrix} x_{n-1} \\ x_n \end{pmatrix}$$

Dadurch läßt sich die ganze Rekkursion schreiben als

$$\begin{pmatrix} x_n \\ x_{n+1} \end{pmatrix} = M_n M_{n-1} \cdots M_1 \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

wobei die M_i mti Wahrscheinlichkeit $\frac{1}{2}$ entweder $\begin{pmatrix} 0 & 1 \\ \beta & 1 \end{pmatrix}$ oder $\begin{pmatrix} 0 & 1 \\ -\beta & 1 \end{pmatrix}$ sind. Also ist unser Problem nicht länger das der zufälligen Fibonaccifolge sonder das eines zufälligen Matrizenprodukts. Um unser Problem auch als Markov-Prozess aufzufassen betrachten wir nicht länger einen zweidimensionalen Zustandsraum sondern reduzieren die Dimension um eins indem wir den Winkel, $x = \binom{\cos(\theta)}{\sin(\theta)}$ mit $\theta \in (-\pi/2, \pi/2]$, oder die Steigung, $x = \binom{1}{m}$ mit $m = \tan(\theta) \in (-\infty, \infty]$ zwischen je zwei Folgegleidern betrachten. Es stellt sich heraus, dass die Wachstumsrate nicht von den Eigenwerten der Zufallsmatrizen abhängt sondern vielmehr vom asymptotischen Verhalten des Produkts $M_n M_{n-1} \cdots M_1$.

Ergebnissen aus der Theorie der Zufallsmatrizen zufolge gilt für eine konstante γ mit Wahrscheinlichkeit 1:

$$\frac{log \, \|M_n \cdots M_1\|}{n} \longrightarrow \gamma \text{ für } n \to \infty \tag{2.1}$$

$$\sqrt[n]{|x_n|} \longrightarrow e^{\gamma} \text{ für } n \to \infty \tag{2.2}$$

Um nun γ berechnen zu können bedienen wir uns einer Formel von Furstenberg:

$$\gamma = \int amp(x) \, d\nu(x) \tag{2.3}$$

Ziel ist es nun jenes Maß ν zu finden welches uns angibt wie Lange der Markov-Prozess in einem bestimmten Intervall $I = [\theta_1, \theta_2]$oder $[m_1, m_2]$ verbringt. Hierfür gelten nun folgende Überlegungen:

Ist der Markov-Prozess in einem Zustand x angelangt so war der vorhergehende Zustand mit Wahrscheinlichkeit $\frac{1}{2}$ entweder $A^{-1}x$ oder $B^{-1}x$. Daher sollte für alle Borel-messbaren Mengen S folgendes gelten:

$$\nu(S) = \frac{1}{2}\nu(A^{-1}S) + \frac{1}{2}\nu(B^{-1}S)$$

Die beiden inversen Abbildungen der Steigung lauten:

$$A^{-1}m = \frac{\beta}{m-1}, \qquad B^{-1}m = \frac{\beta}{1-m}$$

Und daher lautet die Bedingung für unser Maß:

$$\nu([m_1, m_2]) = \frac{1}{2}\nu\left(\frac{\beta}{[m_1, m_2] - 1}\right) + \frac{1}{2}\nu\left(\frac{\beta}{1 - [m_1, m_2]}\right) \tag{2.4}$$

Um nun $\nu([m_1, m_2])$ zu approximieren teilen wir zunächst $[-\pi/2, \pi/2]$ in N gleichgroße Teilintervalle $I_1, \ldots, I_N$. und erhalten für 2.4:

$$\nu(I_j) = \frac{1}{2}\nu\left(\arctan\left(\frac{\beta}{\tan(I_j) - 1}\right)\right) + \frac{1}{2}\nu\left(\arctan\left(\frac{\beta}{1 - \tan(I_j)}\right)\right) \tag{2.5}$$

Um nun diese Gleichung anzunähern setzen wir für unsere diskrete Menge von Intervallen $I_1, \ldots, I_N$ folgendes Gleichungssystem

$$\nu_j = \frac{1}{2}\sum_{k=1}^{N} c_k \nu_k$$

wobei ν_j die diskrete Approximation an $\nu(I_j)$ ist und die Koeffizienten $c_k \in [0, 2]$ angeben wieweit das Intervall I_k mit den beiden Intervall auf der rechten Seite von 2.5 überlappt. Diese N Gleichungen bilden ein lineares System vom Rang $N - 1$welches durch ersetzen der N-ten Gleichung mit $\sum_{j=1}^{N} \nu_j = 1$ zu einem nichtsymmetrisches und dünnbesetztes System vom Rang N wird.

Weiters sei die Funktion $amp(x)$ in 2.3 gegeben durch:

$$amp(x) = \frac{1}{2}log\frac{\|Ax\|_2}{\|x\|_2} + \frac{1}{2}log\frac{\|Bx\|_2}{\|x\|_2} = \frac{1}{4}\log\left(\frac{\beta^4 + 4m^4}{(1 + m^2)^2}\right)$$

Um nun eine Genauigkeit von sechs Stellen zu erreichen müsste man $N = 2^{21}$ wählen. Leider war es uns jedoch nicht möglich einen Solver für dünnbesetzte Gleichungssysteme in Matlab zu finden der unser Gleichungssystem in annehmbarer Zeit löst. Auch der uns empfolene Solver Bi-CGSTAB brachte im Vergleich zum \-Operator keine nenneswerte Beschleunigung. Wir haben jedoch mit $N = 256$ in wenigen Sekunden die ersten zwei Stellen berechnen können.

3 Oszillierendes Integral

Zu berechnen ist das uneigentliche Integral

$$I = \lim_{\epsilon \to 0} \int_{\epsilon}^{1} \sin\left(\cos(x) \cdot e^{\frac{1}{x}}\right). \tag{3.1}$$

Die Schwierigkeit hierbei ist bei Betrachtung des Graphen des Integranden (Abbildung auf dieser Seite) ersichtlich – die Funktion osziliert in der Nähe von 0 so stark, dass für eine Quadratur eine immer kleinere Schrittweite nötig wäre, um das Intergral gut zu approxomieren.

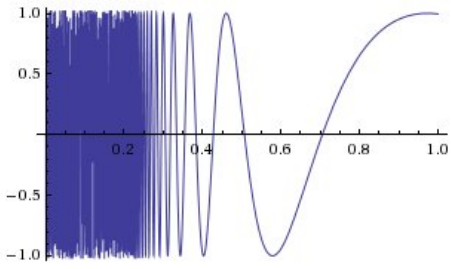


Figure 3.1: Graph des Integranden aus 3.1

3.1 Quadratur mit Standardmethoden

Für stark osszilierende Funktionen, empfiehlt die Matlab-Dokumentation die Funktion quadgk und beschreibt sie als "adaptive quadrature based on a Gauss-Kronrod pair (15th and 7th order formulas)"[2]. Weiters wird angebeben, dass die Funktion auch mit singulären Integrationsgrenzen umgehen kann, wenn die Singularität nicht zu stark ist, wie zB. bei einem Endpunkt c bei dem sich der Integrand wie $\log|x-c|$ oder $|x-c|^p$ für $p \leq -1/2$ verhält. Bei (3.1) ist die Singularität bei 0 zwar stärker, aber trotzdem gibt die Matlab-Dokumentation Anlass auf Hoffnung für gute Ergebnisse mit verhältnismäßig einfachen Methoden[3].

Jedoch spukt quadgk bei den Grenzen 0 und 1 nur aus *"warning: quadgk: non finite integrand encountered X"*. Aber für das Integral von

Fehler durchs Abschneiden des Limes $2 \cdot \epsilon$

.

Warning: quadgk: maximum interval count (1000000) met
warning: quadgk: Error tolerance not met. Estimated error 0.0110442
ans = 0.12623

[2] http://www.mathworks.com/help/techdoc/ref/quadgk.html

[3] Aufgrund der abzulehnenden Lizenzpolitik von *The MathWorks, Inc.*, die es TU-Studierenden verunmöglicht Matlab zu leistbaren Preisen auf mehr als einem Computer oder User-Account zu installieren – und selbst dafür viel Bürokratie von den diesbezöglich ohnehin schon stark belasteteten Studierenden abverlangt, ist jedoch die Opensource-Software *Octave* zum Einsatz gekommen.

Algorithm 5

```
N=10^6;Epsilon=0.005;
integrand = @(x) sin(cos(x) .* e .^(1 ./ x));
quadgk(integrand,Epsilon,1,'MaxIntervalCount',N)
```

3.2 Quadratur von Nullstelle zu Nullstelle

Um die Oszilation in den Griff zu bekommen legt Abbildung (3.2) nahe, das Intervall von 1 bis 0 in die Intervalle zwischen den Nullstellen des Integranden zu unterteilen und dann die Reihe der Integrale über diese Intervalle zu betrachten.

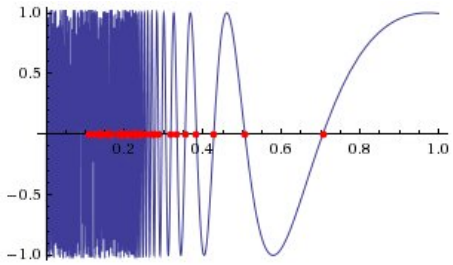


Figure 3.2: Graph des Integranden aus 3.1 mit Nullstellen

Die Dichte der Nullstellen jedoch nimmt für $x \to 0$ exponentiell zu und machen es daher unmöglich diese weit genug zu berechnen.

3.3 Eine untere Schranke

Um eine untere Schranke möglichst gut zu berechnen teilen wir unser Integral in drei Abschnitte auf:

1. Abschnitt: von der 2.Nullstelle, sprich bei $x_0 \approx 0.50702\ldots$ bis 1 wobei wir diesen Bereich so genau wie möglich approximieren und konnten so einen Wert von $A_1 = 0.0935682320981838$ berechnen.

2. Abschnitt: Diesen Abschnitt berechnen wir von der Nullstelle x_o bis zur Nullstelle x_{2N} so genau wie möglich und versuchen aus ihm den 3. Abchnitt zu berechnen. Hier erhalten wir für $N = 5 \cdot 10^4 - 1$ den Wert $A_2 = 0.0326487638958058$.

3. Abschnitt: Für diesen Abschnitt berechnen wir nun eine untere Schranke wie nachfolgend beschrieben.

Wenn man das Ergebnis aus dem 2.Abschnitt ein wenig genauer betrachtet erkennt man eine Tendenz wie sich die Flächeninhalte zwischen je drei Nullstellen verhalten,

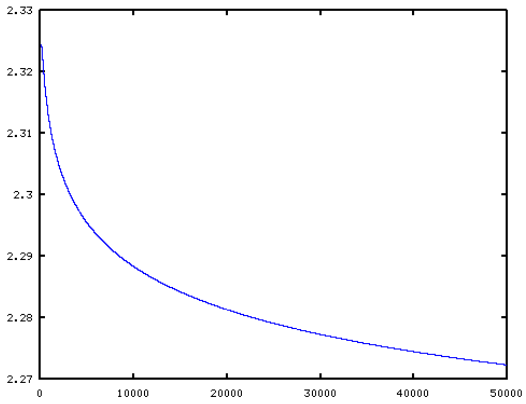


Figure 3.3: x-Achse: n y-Achse:β_n

also jene N Flächeninhalte I_D welche eine Summe zischen dem Flächeninhalt unter einem Wellenberg und einem Wellental sind. Unsere Vermutung lautet nun:

$$I_D(n) = \alpha \cdot \frac{1}{n^{\beta_n}} \tag{3.2}$$

für $\alpha, \beta_n \in \mathbb{R}$.

Daraus folgt sofort für $n = 1$, dass $\alpha = I_D(1) = 0.0218812765213859$ sein muss und für $\beta_n$erhählt man durch umformen:

$$\beta_n = \frac{log\left(\frac{\alpha}{I_D(n)}\right)}{log(n)} \tag{3.3}$$

Da unsere β sehr langsam aber doch fallen gilt weiters: $I_D(n) \geq I_D(1) \cdot \frac{1}{n^{\beta_N}}$ für $\forall n \geq N$ und $\beta_N = 2.27235171257442$. Um nun den 3. Abschnitt nach unten abzuschätzen und ohne eine weiter Nullstelle zu berechnen, approximieren wir ihn mittels

$$\sum_{n=N+1}^{\infty} \alpha \cdot \frac{1}{n^{\beta_N}} \tag{3.4}$$

und bekommen $A_3 = \sum_{n=N+1}^{5*10^7} \alpha \cdot \frac{1}{n^{\beta_N}} = 1.80576938049079e-08$

Nach Addition aller drei Abschnitte erhalten wir die untere Schranke und leider auch unsere beste Näherung

$$U = A_1 + A_2 + A_3 = 0.126217014051683$$

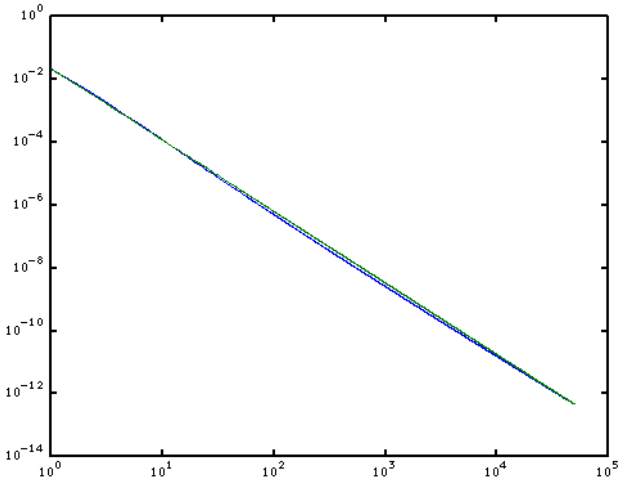


Figure 3.4: blau: $I_D(n)$ grün: $\alpha \cdot \frac{1}{n^{\beta_N}}$

4 Integral über $\mathbb{R}^8$

Ziel dieser Aufgabe ist es, das Integral

$$J = \int\limits_{\mathbb{R}^8} \sin\left(\prod_{n=1}^{8} |x_n^n|\right) \exp\left(-\frac{1}{10}\sum_{n=1}^{8} x_n^2\right) dx$$

zu berechnen.

Da die Dimension des Integrals ziehmlich hoch ist, macht eine herkömmliche Quadratur wenig Sinn, da der Aufwand zu groß wäre.

Eine Befragung des modernen Orakels *wolframalpha.com* lässt Hoffnung aufkeimen, wenn zuerst die vereinfachte Frage nach dem entsprechenden Integral mit nur vier Variablen $x_1, \ldots, x_4$ auf der kompakten Menge $[-1,1]^4$ stellen, also

$$\tilde{J} = \int\limits_{[-1,1]^4} \sin\left(\prod_{n=1}^{4} |x_n^n|\right) \exp\left(-\frac{1}{10}\sum_{n=1}^{4} x_n^2\right) dx$$

oder in der Sprache von *WolframAlpha*:

```
NIntegrate[Sin[Abs[Prod[x[i]^i,{i,1,4}]]]*exp(-1/10*Sum[x
    [i]^2,{i,1,4}]), {x[1], -1, 1}, {x[2], -1, 1}, {x[3],
    -1, 1}, {x[4], -1, 1}]
```

Hierbei ergibt sich $\tilde{J} \approx 0.10363901499310284$. Aber schon bei fünf Variablen oder einem geringfügig größerem Intervall Antwortet das Orakel nur mehr

> "**Computation timed out.** No more results available."

Ein Zusammenhang zwischen den berechenbaren Problemen und dem gesuchten Wert J dürfte sich leider nicht einfach finden lassen, insofern müssen andere Lösungswege eingeschlagen werden.

4.1 Abschätzung

Wenn man den Sinus mit 1 abschätzt, erhält man eine obere Schranke für das Integral:

$$\begin{aligned}
J &\leq \int\limits_{\mathbb{R}^8} \left|\sin\left(\prod_{n=1}^{8} |x_n^n|\right) \exp\left(-\frac{1}{10}\sum_{n=1}^{8} x_n^2\right)\right| d\vec{x} \\
&\leq \int\limits_{\mathbb{R}^8} \exp\left(-\frac{1}{10}\sum_{n=1}^{8} x_n^2\right) d\vec{x} \\
&\leq \int\limits_{\mathbb{R}^7} \exp\left(-\frac{1}{10}\sum_{n=1}^{7} x_n^2\right) \cdot \underbrace{\int\limits_{\mathbb{R}} \exp\left(-\frac{x_8^2}{10}\right) dx_8}_{\approx 5.604991} d(x_i)_{i=1,\ldots,7} \qquad (4.1) \\
&\approx 5.604991^8 = 974090.910340025
\end{aligned}$$

Das stellt auch die Existenz des Integrals sicher. Der numerische Wert aus (4.1) lässt sich mit WolframAlpha mit N[Integrate[E^(−x^2/10), {x, −Infinity, Infinity}]] ermitteln.

4.2 Monte-Carlo-Integration

Auch wenn wir damit die Garantie auf ein richtiges Ergebnis aufgeben, ist ein randomisierter Algorithmus eine einfache Methode, um schnell zu brauchbaren Ergebnissen zu kommen.

Wir wollen ein Verfahren verwenden wie die in [Kol08, Kapitel 18.1] als Mittelwertsmethode beschriebene Variante der Monte-Carlo-Integration. Die Idee ist recht simpel: Wir interpretieren das Integral J als Erwartungswert einer geeigneten Zufallsvariable Y (die erst zu finden ist), erzeugen durch Zufallsexperiemente oder -generatoren möglichst viele Realisierungen y_i von Y und können so den dann den Erwartungswert über das arithmetische Mittel $\overline{y_i}$ schätzen:

$$J = \mathbb{E}Y \approx \overline{y_i} := \frac{1}{n}\sum_{i=1}^{n} y_i$$

Als nächstes stellt sich natürlich die Frage nach der Verteilung von Y und wie wir viele Realisierungen y_i erhalten können. Hier erweist sich die gewählte Methode aus zweierlei Gründen als besonders geeignet für das vorliegende Problem. Der zweite Faktor $\exp\left(-\frac{1}{10}\sum_{n=1}^{8} x_n^2\right)$ im Integranden von J erinnert schon an die Dichtefunktion einer multivariaten Normalverteilung – das werden wir ausnützen können. Außerdem haben normalverteilte Zufallszahlen die angenehme Eigenschaft, sehr selten Werte sehr große oder sehr kleine Werte anzunehmen. So wird es möglich sein den ganzen $\mathbb{R}^8$ betrachten zu können anstatt sich mit einer Annährung über eine möglichst große kompakte Teilmenge des $\mathbb{R}^8$ zufrieden geben zu müssen.

Wie zb. in [CK08, Satz C5.6, S. 209f] nachzulesen ist, gilt für eine stetige Funktion $t : \mathbb{R}^8 \to \mathbb{R}$ und einen Zufallsvektor $\overrightarrow{X} \in \mathbb{R}^8$

$$\mathbb{E}t(\overrightarrow{X}) = \int_{\mathbb{R}^8} t(x) \cdot f_{\overrightarrow{X}}(x)\, dx$$

falls $\mathbb{E}t(\overrightarrow{X})$ existiert. Die Dichtefunktion einer p-dimensionalen reellen normalverteilten Zufallsvariabeln $\overrightarrow{X}$ mit Erwartungswert μ und Kovarianzmatrix Σ (mit $|\Sigma|$ bezeichnen wir in diesem Zusammenhang die Determinate von Σ) lautet

$$f_{\overrightarrow{X}}(\overrightarrow{x}) = \frac{1}{(2\pi)^{\frac{p}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\overrightarrow{x}-\mu)^{\intercal}\Sigma^{-1}(\overrightarrow{x}-\mu)\right).$$

Im Vergleich mit dem bereits erwähnten zweiten Faktor des Integranden von J erhält man nun mit

$$\begin{aligned} \mu &:= \overrightarrow{0} \\ \Sigma &:= 5 * E_8 \end{aligned}$$

$$t(\overrightarrow{x}) = (2\pi)^4 5^4 \cdot \sin\left(\prod_{n=1}^{8} |x_n^n|\right)$$

das gewünschte Ergebnis

$$\int_{\mathbb{R}^8} t(x) \cdot f_{\overrightarrow{X}}(x)\, dx = J.$$

Somit ist $Y = t(\overrightarrow{X})$ wobei $\overrightarrow{X}$ mit ermitteltem μ und Σ multivariat normalverteilt ist und es gilt wie eingangs erwähnt

$$J = \mathbb{E}Y = \mathbb{E}t(\overrightarrow{X}) \approx \frac{1}{n} \sum_{i=1}^{n} t(\overrightarrow{x_i}).$$

4.2.1 Varianz und Konfidenzintervall

Um die Genauigkeit des Ergebnisses zu schätzen, wird in Algorithmus 6 auf Seite 22 nicht nur der Erwartungswert sondern auch die Varianz geschätzt. Hierzu findet der Schätzer

$$\begin{aligned} &\tfrac{1}{N-1} \cdot \textstyle\sum_{i=1}^{N} (x_i - \overline{x})^2 \qquad\qquad = \\ &\tfrac{1}{N-1} \cdot \left(\textstyle\sum_{i=1}^{N} {x_i}^2 - 2 \cdot \overline{x} \cdot \sum_{i=1}^{N} x_i + N \cdot \overline{x}\right) \end{aligned}$$

Anwendung.

Daraus ergibt sich, dass ein absoluter Fehler von höchstens

$$u_{\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{N}} \tag{4.2}$$

mit der Wahrscheinlichkeit $1 - \alpha$ vorliegt, wie zB. in [Kol08, Satz 16.1] beschrieben wird (u_α bezeichnet hier das α-Fraktil von $N(0,1)$ und σ die Wurzel der Varianz).

4.2.2 Varianzreduktion

Mit der beschriebenen Methode lassen sich zwar schon einige Stellen in absehbarer Zeit mit normalen Desktoprechnern ermitteln (siehe 4.2.4), aber wegen der recht großen Varianz der verwendeteten Zufallsvariablen, ist das Konfidenzintervall auch bei sehr vielen Zufallsvektoren sehr groß. Wie aus (4.2) hervorgeht, sind 10^2 mehr Zufallsvektoren nötig, um das Ergebnis um eine Stelle genauer zu erhalten. Da stößt man natürlich sehr schnell an Grenzen, die sich durch die Vorgabe ergeben, innerhalb weniger Monate Ergebnisse präsentieren zu können.

Um die Ergebnisse mit der beschriebenen Methode zu verbessern, ist es zielführender die Varianz zu reduzieren. Hierfür bietet sich als sehr elementares Verfahren die Methode der Abtrennung des Hauptteils an, wie sie in [Erm75, §2.1] beschrieben ist. Dazu wird eine Funktion G benötigt, die sich leicht integrieren lässt (symbolisch oder mit herkömlicher Quadratur) und genügend nahe bei der zu integrierenden Funktion F liegt. Anstatt $\int F(x)\, dx$ zu berechnen, wird $\int G(x)\, dx + \int F(x) - G(x)\, dx$

berechnet, wobei nur der zweite Summand mit der Monte-Carlo-Integration berechnet werden muss und eine niedrigere Varianz erhofft kann, wenn die Funktion G gut gewählt wurde.

In unserem konkreten Fall, haben wir die Funktion F über die Reihentwicklung des Sinus innerhalb eines Intervalls $[-c, c]$ approximiert

$$G(\vec{x}) = T \circ \mathbb{I}_{[-c,c]} \left(\prod_{i=1}^{8} x_i^i \right) \cdot \exp\left(-\frac{1}{10} \sum_{n=1}^{4} x_n^2 \right),$$

wobei

$$T(x) = \sum_{k=0}^{\tilde{N}} (-1)^k \cdot \frac{x^{2k+1}}{(2k+1)!}.$$

Die Monte-Carlo-Integration für $F - G$ kann dann genauso durchgeführt werden.

Die Ergebnisse sind allerdings ernüchternd: während das Integral $\int F(x) - G(x)\,dx$ tatsächlich verhältnismäßig klein ist (ungefähr ein siebzigstel des Integrals $\int F(x)\,dx$), reduziert sich die Varianz nur sehr wenig. Mit einer Reihenentwicklung der Ordnung 171 und $c = 33$ ergibt sich eine Varianz von ungefähr $2.88 \cdot 10^{11}$ statt bisher $3.39 \cdot 10^{11}$ und somit eine Verkleinerung des Fehlerintervalls um nicht ganz 8% während sich die Rechenzeit mehr als verdreifacht hat (von 149 auf 517 Sekunden bei einer Stichprobengröße von 10^8).

Vermutlich lassen sich mit anderen Methoden zur Varianzreduzierung bessere Ergebnisse erzielen, aber zugunsten der anderen vier Aufgaben, begnügen wir uns mit den gefundenen Stellen.

4.2.3 Mögliche Implementierungen – R vs. Python und NumPy vs. Octave

Um möglichst viele normalverteilte Zufallsvektoren zu erzeugen haben *R*, *Python* mit dem Modul *NumPy* und Octave getestet. Wir jeweils 30 mal 10^6 Zuvallsvektoren mit gegebenem μ und Σ erzeugt und sie in einer Variable abgespeichert. Die gemessenen Laufzeiten sind in Tabelle1 angegeben.

	R	python	Octave
real	0m45.972s	0m23.864s	0m14.883s
user	0m38.580s	0m20.740s	0m10.020s
sys	0m6.310s	0m2.670s	0m3.640s

Tabelle 1: Laufzeiten zur Erzeugung von 30 mal 10^6 normalverteilten Zufallsvektoren

Deshalb fällt unsere Wahl auf Ocatve. Andere Möglichkeiten haben wir nicht untersucht – sie mögen vielleicht noch ein wenig Rechenzeit sparen, aber dafür umso mehr Lebenszeit kosten. Matlab ist noch um ein paar Prozent schneller, wurde aber hier nicht berücksichtigt (vergleiche hierzu die Fußnote 3 auf Seite 13).

4.2.4 Ergebnisse

Mit dem Octave-Code aus Algorithmus 6 auf der nächsten Seite ergibt sich

$$J \approx 31108.278034997\ldots =: J_{10^{10}}$$

wobei $|J - J_{10^{10}}| < 4.7925265$ mit einer Wahrscheinlichkeit von 0.9 gilt und $|J - J_{10^{10}}| < 7.50654$ mit einer Wahrscheinlichkeit von 0.99. Von der Richtigkeit der ersten 4 Stellen kann also ausgegangen werden. Dazu waren allerdings insgesamt etwas mehr als 23 Stunden Rechenzeit notwendig (verteilt auf 4 Etappen, jeweils zwei auf einem Kern von einem Intel Core2 Duo E8400 @ 3.00Ghz und einem Intel Pentium 4 CPU @ 3.06GHz). Für eine weitere Stelle bräuchten wir also noch etwas mehr als 4 Tage Rechenzeit (mit einem vergleichbarem CPU-Kern).

Algorithm 6 Octave-Code für die Monte-Carlo-Integration

```
% Konstante...
N = 10^6; N2 = 10^4; % Stichprobengroesse = N * N2
result = 0; variance_part1=0;variance_part2=0;
S = eye(8)*5; mu = zeros(1,8);

tic; for i=1:N2
    % Realisierungen der Zufallsvariable X...
    r = mvnrnd(mu, S, N);

    % Wende Funktion t auf Realisierungen an...
    r = sin(prod(abs( r .^ repmat([1:8], N, 1)), 2));

    % Schaetzen vom Erwartungswert
    tmp_result = sum(r) / N;
    result = result + tmp_result;

    % Schaetzen der Varianz
    variance_part1 = variance_part1 + sum ( r.^2 ) / N;
    variance_part2 = variance_part2 + tmp_result;

    printf("%.1e von %.1e Stichproben generiert ... \
        Zwischenergebnis: %f\n", i*N, N*N2, result / i)
end; toc

% Stichprobengroesse
disp("Stichprobengroesse:\n%d", N * N2)

% Schaetzer fuer E(t(X))
result = (10 * pi)^4 * (result / N2);
printf("Geschaetzter Erwartungswert:\n %f", result)

% Schaetzer fuer die Varianz
result_variance = N / (N*N2 - 1) * ((10 * pi)^8 *
    variance_part1 + 2 * result * (10 * pi)^4 * variance_part2
     + N2 * result);
printf("Geschaetzte Varianz:\n%f", result_variance)

% Konfidenzintervall
disp('Konfidenzintervall')
error = sqrt(result_variance / (N*N2));
printf("Mit Wahrscheinlichkeit  %f ist der Fehler kleiner als
    %e\n", 1-2*0.050, 1.644*error)
printf("Mit Wahrscheinlichkeit  %f ist der Fehler kleiner als
    %e\n", 1-2*0.025, 1.959*error)
printf("Mit Wahrscheinlichkeit  %f ist der Fehler kleiner als
    %e\n", 1-2*0.010, 2.326*error)
printf("Mit Wahrscheinlichkeit  %f ist der Fehler kleiner als
    %e\n", 1-2*0.005, 2.575*error)
```

5 Lotka-Volterra-Differentialgleichung

Bei dieser Aufgabe geht es darum, dass man den Parameter $b > 0$ in der Lotka-Volterra-Differentialgleichung

$$x' = (1 - by)x \tag{5.1}$$
$$y' = (-1 + x)y \tag{5.2}$$

so bestimmt, sodass die Periodendauer $T = 10$ ist.
Aufgrund der Ähnlichkeit mit einer lineraren Differentialgleichung ($z' = \epsilon * z$) probieren wir den Ansatz

$$x(t) = e^{u(t)}, y(t) = e^{v(t)}$$

Daraus folgt sofort

$$x' = u' * e^u = u' * x$$
$$y' = v' * e^v = v' * y$$

und durch einsetzen in unsere DGL erhalten wir

$$u' * x = (1 - be^v) * x$$
$$v' * y = (-1 + e^u) * y$$

Nach kürzen von x und y aus der 1. bzw. 2. Gleichung und Bildung des Differntial-Quotienten erhalten wir

$$\frac{v'}{u'} = \frac{-1 + e^u}{1 - be^v}$$

Durch umformen auf

$$(1 - be^v) * v' = (-1 + e^u) * u'$$

ergibt sich eine DGL mit getrennten Veränderlichen wälche wir auf beide beiden Seiten integrieren.

$$v - be^v + c_1 = -u + e^u + c_2$$

Mit $c := c_1 - c_2$ findet man eine Invariante der DGL, nämlich

$$c = e^u + be^v - u - v \tag{5.3}$$

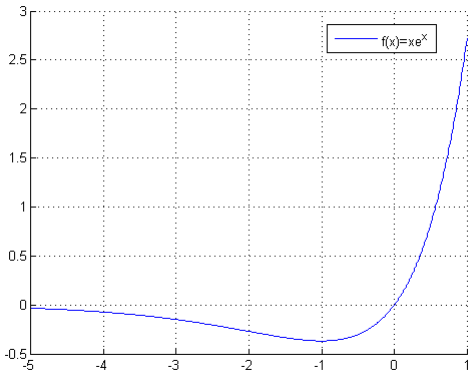
welche man durch einsetzen der Startwerte $x(0) = y(0) = \frac{1}{5}$ berechnen kann.

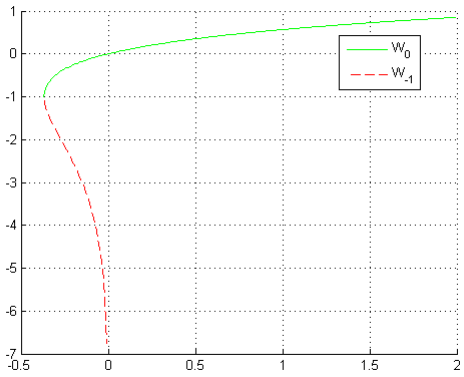
$$c = x(0) + by(0) - \ln(x(0)) - \ln(y(0)) = \frac{1}{5}(1 + b) + 2 * \ln(5)$$

Nach Auflösen von Gleichung 5.3 nach y sowie Anwendung der Exponentialfunktion auf beiden Seiten und der Berücksichtigung das $e^v = y$ gilt sowie Multiplikation mit $(-b)$ erhalten wir

$$-by * e^{-by} = -be^{e^u - u - c} \tag{5.4}$$

Um diese Gleichung nun nach y auflösen zu können bedienen wir und der Lamberteschen W-Funktion welche die Umkehrfunktion von $f(x) = x * e^x$ ist. Es gilt also $x = W(x) * e^{W(x)}$. Da f im Bereich $[-\infty, 0]$ nicht injektiv ist besitz die Lambertsche W-Funktion auf dem Intervall $\left[-\frac{1}{e}, 0\right]$ zwei Lösungen. Mit W_0 bezeichnen wir von nun an den oberen und mit W_{-1} den unteren der beiden Äste. Mehr dazu später.





Aus Gleichung 5.4 erhalten wir nach Anwendung der Lambertschen W-Funktion, Multimplikation mit $-\frac{1}{b}$ sowie der Tatsache dass $e^u = x$ gilt die beiden geschlossenen Lösungen für y:

$$y_i(x) = -\frac{1}{b} W_i\left(-b\frac{e^{x-c}}{x}\right) \quad mit\, i \epsilon \{-1, 0\} \tag{5.5}$$

Da die Lambertsche W-funktion nur für Werte größer gleich $-\frac{1}{e}$ definiert ist untersuchen wir nun wann das Argument der Lambertschen W-Funktion in Gleichung 5.5 den Wert $-\frac{1}{e}$ annimmt

$$-e^{-1} = -b * e^{x - \ln x - c}$$

Daraus folgt nun weiter:

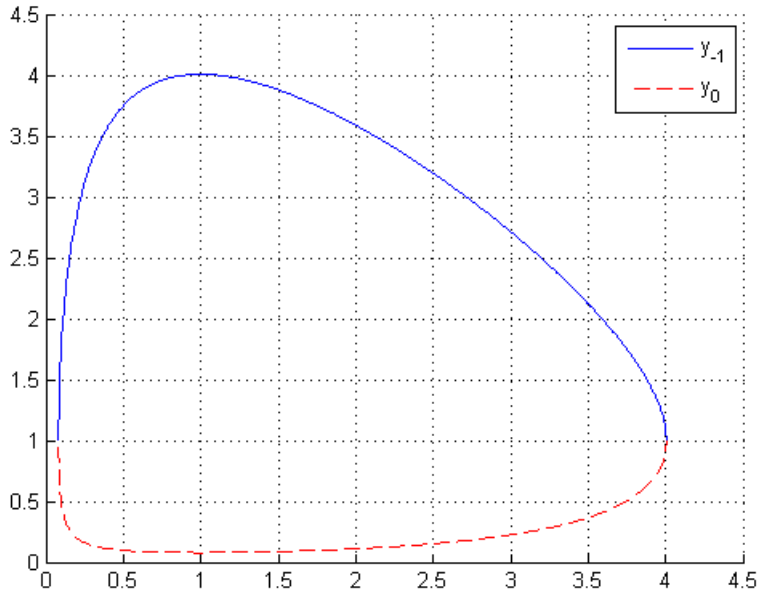
$$\begin{aligned} -1 &= \ln b + x - \ln x - c \\ \ln x - x &= \ln b - c + 1 \\ xe^{-x} &= be^{1-c} \\ -xe^{-x} &= -be^{1-c} \end{aligned}$$

Nun können wir auf beiden Seiten die Lambertsche W-Funktion Anwenden und erhalten den minimalen und den maximalen Wert der für x möglich ist.

$$x_{min} = -W_0\left(-be^{1-c}\right)$$

$$x_{max} = -W_{-1}\left(-be^{1-c}\right)$$

Da $c \geq 3$ und $b \geq 0$ gibt es keine weiteren Probleme. Ein Phasendigramm für $b = 1$ sieht dann wie folgt aus:



Um nun die Periodenlänge T zu berechnen betrachten wir Gleichung 5.1 aus unserer DGL und lösen sie nach t auf und setzten für y die beiden Lösungen aus 5.5ein:

$$x' = \frac{dx}{dt} = (1 - by)x$$

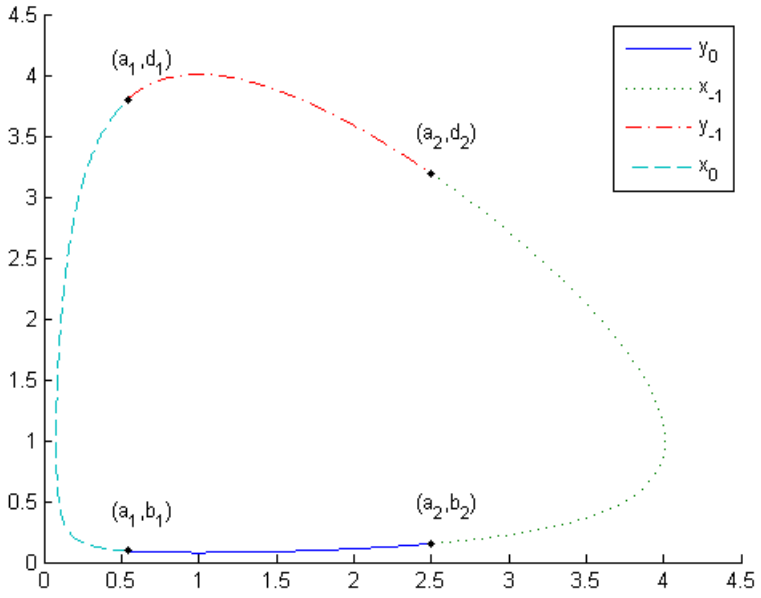
$$dt = \frac{dx}{\left(1 + W_i\left(-b\frac{e^{x-c}}{x}\right)\right) * x}$$

Jetzt können wir leicht die Periodendauer T berechnen. Es gilt nun:

$$T = \int_{x_{min}}^{x_{max}} \frac{dx}{\left(1 + W_0\left(-b\frac{e^{x-c}}{x}\right)\right) * x} + \int_{x_{max}}^{x_{min}} \frac{dx}{\left(1 + W_{-1}\left(-b\frac{e^{x-c}}{x}\right)\right) * x} \tag{5.6}$$

Aufgrund der Tatsache, dass der Nenner in beiden Integranden für x_{min} und x_{max} verschwindet handelt es sich also um zwei uneigentliche Integrale. Um dieses "Problem" zu umgehen spalten wir das gesamte Phasendiagramm nicht nur in zwei sonder in vier Teile auf. Hierfür definieren wir uns:

$$
\begin{aligned}
a_1 &:= 1 - \frac{1}{2}(1 - x_{min}) \\
a_2 &:= 1 + \frac{1}{2}(x_{max} - 1) \\
b_1 &:= y_0(a_1) \\
b_2 &:= y_0(a_2) \\
d_1 &:= y_{-1}(a_1) \\
d_2 &:= y_{-1}(a_2)
\end{aligned}
$$



Im Intervall $[a_1, a_2]$ sind die Nenner in Gleichung 5.6 sicher von Null verschieden und wir können die beiden Integrale berechnen. Für die anderen beiden Teilabschnitte des Phasendiagramms lösen wir Gleichung 5.2 nach t auf und erhalten:

$$dt = \frac{dy}{(-1 + x) * y}$$

Nun kann man analog zur Findung der Lösung für y auch Lösungen für x finden:

$$x_i(y) = -W_i\left(-\frac{e^{by-c}}{y}\right) \; mit \; i\epsilon\{-1, 0\}$$

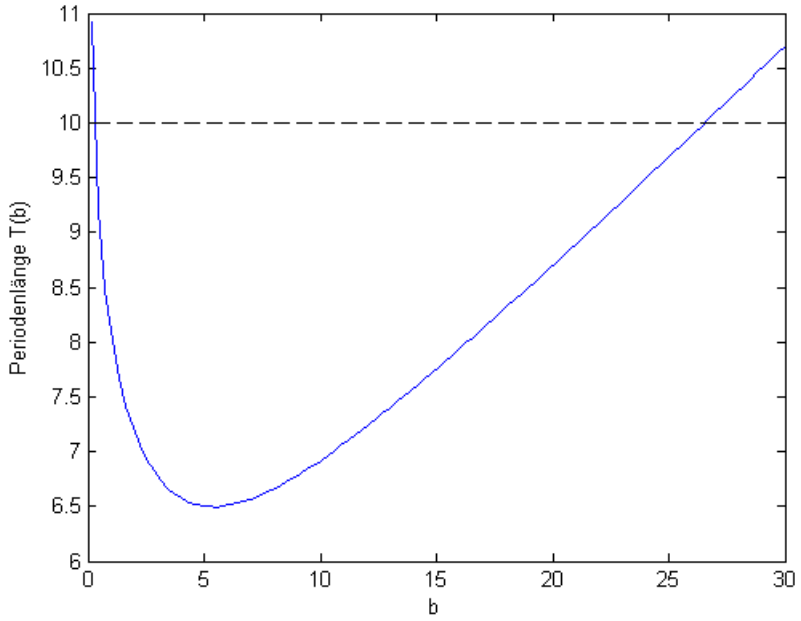
Für dt erhält man:

$$dt = \frac{dy}{-\left(1 + W_i\left(-\frac{e^{by-c}}{y}\right)\right) * y}$$

Das ergibt nun für das gesamte Periodendauer:

$$\begin{aligned} T &= \int_{a_1}^{a_2} \frac{dx}{\left(1 + W_0\left(-b\frac{e^{x-c}}{x}\right)\right) * x} + \\ &+ \int_{b_2}^{d_2} \frac{dy}{-\left(1 + W_{-1}\left(-\frac{e^{by-c}}{y}\right)\right) * y} + \\ &+ \int_{a_2}^{a_1} \frac{dx}{\left(1 + W_{-1}\left(-b\frac{e^{x-c}}{x}\right)\right) * x} + \\ &+ \int_{d_1}^{b_1} \frac{dy}{-\left(1 + W_0\left(-\frac{e^{by-c}}{y}\right)\right) * y} \end{aligned}$$

Warum auch immer gibt es nun zwei Lösungnen für unser Problem wie die nach folgende Graphik zeigt, bei der b gegen die Periodenlänge T geplottet wird.



Also gibt es nicht nur ein sonder zwei b die eine Periodenlänge von $T = 10$ verursachen.
Mit Hilfe von Matlab, in dem die Lambertsche W-Funktion bereichts implementier ist und der Verwendung der simpsonschen Quadratur (auch aus Matlab) konnten wir mit einem einfachen Bisektionsverfahren die beiden Lösungen auf 14 Stellen genau berechnen

$$\begin{aligned} b_1 &= 0.265578070942143 \\ b_2 &= 26.536321705563118 \end{aligned}$$

Literatur

[Bai79] Robert Baillie, *Sums of reciprocals of integers missing a given digit*, The American Mathematical Monthly **86** (1979), no. 5, 372–374 (English).

[BLWW06] F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel, *Vom I
ösen numerischer Probleme ein Streifzug entlang derßiam 10x10-digit Challenge"; mit... 44 TAB.*, Springer-Lehrbuch Masterclass, Springer, 2006.

[CK08] E. Cramer and U. Kamps, *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik: Ein Skript f
ür Studierende der Informatik, der Ingenieur- und Wirtschaftswissenschaften*, Springer-Lehrbuch, Springer, 2008.

[Erm75] S.M. Ermakov, *Die Monte-Carlo-Methode und verwandte Fragen*, HochschulbÃŒcher fÃŒr Mathematik, R. Oldenbourg, 1975.

[ET98] M. Embree and L. N. Trefethen, *Growth and Decay of Random Fibonacci Sequences*, Proceedings: Mathematical, Physical and Engineering Sciences **Vol. 455** (1998), no. No. 1987 (Jul. 8, 1999), 2471–2485.

[HS07] J. Havil and M. Stern, *GAMMA: Eulers Konstante, Primzahlstr
ände und die Riemannsche Vermutung*, Springer, 2007.

[Kem14] A. J. Kempner, *A Curious Convergent Series*, Amer. Math. Monthly **21** (1914), pp. 48–50.

[Kol08] M. Kolonko, *Stochastische Simulation : Grundlagen, Algorithmen und Anwendungen*, 1. auflage ed., Wiesbaden : Vieweg + Teubner, 2008.

[Rib04] P. Ribenboim, *The little book of bigger primes*, Springer, 2004.

[Vis98] Divakar Viswanath, *Random fibonacci sequences and the number 1.13198824...*, Math. Comp **69** (1998), 1131–1155.

Ein paar Stellen zu Aufgabe 1

Karl Rupp

1 Vorbemerkungen*

Nachdem ich es nicht lassen kann, mich an numerischen Problemen zu beweisen, habe ich die knackigste Aufgabe der diesjährigen Numerical Challenge für meine Spielereien ausgewählt. Prof. Jüngel hat Anfang Jänner nebenläufig erwähnt, dass es zur ersten Aufgabe erst eine bekannte Stelle gibt. Soetwas lässt man sich nicht zweimal sagen und versucht sich sogleich selbst daran.

2 Erste Annäherung mit Brute Force

Dass die Reihe konvergiert, ist unmittelbar klar, wenn man erkennt, dass die Summanden eine Teilmenge der Summanden der Kempner-Reihe ist. Für erste numerische Untersuchungen werden zuerst Partialsummen der Form

$$s_N := \sum_{i=1}^{N} a(n) \tag{1}$$

betrachtet. Um die Primzahlen bis N zu erzeugen, die nicht die Ziffer 1 enthalten, werden zuerst alle Primzahlen bis $\sqrt{N}$ erzeugt, und dann damit alle Zahlen bis einschließlich N auf Teilbarkeit getestet. Damit erhält man die folgenden Werte:

N	s_N	$\frac{s_N - s_{N-1}}{s_{N-1} - s_{N-2}}$
10^2	1.176	-
10^3	1.436	-
10^4	1.606	0.65
10^5	1.723	0.69
10^6	1.805	0.70
10^7	1.865	0.73
10^8	1.911	0.77
10^9	1.947	0.78
10^{10}	1.975	0.79
10^{11}	1.999	0.81
10^{12}	2.017	0.82

Die Rechenzeit für den letzten Fall in der Tabelle liegt bei etwa 10 Stunden unter Benutzung von acht CPU-Threads auf einer Intel Core i7-960 CPU. Jede zusätzliche Zehnerpotenz erhöht die Rechenzeit um etwa einen Faktor 20, da einerseits zehnmal mehr Primzahlen zu finden sind, und andererseits die Anzahl möglicher Primteiler um etwa $\sqrt{10}$ zunimmt.

3 Approximation durch Geometrische Reihen

Wie man gut in der vorangegangenen Tabelle erkennen kann, beträgt der Beitrag durch Berücksichtigung einer weiteren Ziffer etwa drei Viertel dessen, was bei der vorangegangenen Ziffer dazugekommen ist. Dies läßt es plausibel erscheinen, die gefragte Reihe durch eine geometrische Reihe der Form

$$\tilde{s}_N = \tilde{s}_0 \sum_{k=1}^{\log(N)} \lambda_i \tag{2}$$

mit $\lambda \approx 0.8$ anzunähern bzw. fortzusetzen. Im folgenden sind ein paar Werte für λ mit der zugehoerigen extrapolierten Summe $\tilde{s}_\lambda$ gelistet:

λ	$\tilde{s}_\lambda$
0.81	2.097
0.82	2.103
0.83	2.109
0.84	2.116
0.85	2.124
0.86	2.133
0.87	2.143
0.88	2.156
0.89	2.170
0.90	2.187

Die im vorigen Abschnitt berechneten Werte für λ deuten auf einen Wert im Bereich 0.85, allerdings leidet eine genauere Bestimmung von λ unter der hohen Sensitivität von $\tilde{s}_\lambda$, da kleine Abweichungen in λ bereits Abweichungen in der ersten oder zweiten Nachkommastelle der fortgesetzten Summe haben.

Da allerdings λ nach oben mit 0.9 beschräenkt ist (jede zusaetzliche Ziffer einer Primzahl ist mit einer Wahrscheinlichkeit 0.9 zulässig) und $\lambda < 0.82$ aufgrund der numerischen Ergebnisse ausgeschlossen erscheint, können die ersten beiden Ziffern von s als 2.1 angegeben werden.

4 Diskrete Heuristik liefert Schranken

Im Folgenden sollen heuristische untere und obere Schranken für s hergeleitet werden. Dazu betrachtet man

$$r_k = \sum_{n=10^k}^{10^{k+1}} a(n) \ . \tag{3}$$

Es gilt klarerweise

$$\frac{\#\{n \in \{10^k, \ldots, 10^{k+1}\} : a(n) \neq 0\}}{10^{k+1}} < r_k < \frac{\#\{n \in \{10^k, \ldots, 10^{k+1}\} : a(n) \neq 0\}}{10^k} \ , \tag{4}$$

wobei $\#\{M\}$ die Anzahl der Elemente der Menge M bedeutet. Damit gilt es nun, die Anzahl der Primzahlen ohne Ziffer 1 zwischen 10^k und 10^{k+1} zu bestimmen.

Das Primzahlentheorem besagt, dass

$$\pi(N) := \#\big\{n \in \{1,\ldots,N\} : n \text{ prim}\big\} \to N/\ln N \tag{5}$$

konvergiert. Damit folgt unmittelbar

$$\#\big\{n \in \{10^k,\ldots,10^{k+1}\} : n \text{ prim}\big\} \to \frac{10^{k+1}}{(k+1)\ln 10} - \frac{10^k}{k\ln 10}\ . \tag{6}$$

An dieser Stelle sei nun die Annahme getroffen, dass jede Primzahlenziffer als Zufallsvariable angesehen werden kann. Dies ist stillschweigend bereits im vorigen Abschnitt bei der Begründung für $\lambda \le 0.9$ geschehen. Insbesondere sei also angenommen, dass eine Primzahlen mit k Stellen mit Wahrscheinlichkeit

$$\text{const.} \times 0.9^{k-2} \tag{7}$$

keine Ziffer 1 hat, wobei die Konstante die Wahrscheinlichkeiten für passende erste und letzte Ziffern der Primzahl beinhalten. Damit kann nun die Anzahl der Primzahlen mit $k+1$ Ziffern, aber ohne Ziffer 1, wie folgt abgeschätzt werden:

$$\begin{aligned} \#\big\{n \in \{10^k,\ldots,10^{k+1}\} : a(n) \neq 0\big\} &\approx \\ \approx C \times 0.9^k \times \#\big\{n \in \{10^k,\ldots,10^{k+1}\} : n \text{ prim}\big\} &\\ \to C \times 0.9^k \times \left(\frac{10^{k+1}}{(k+1)\ln 10} - \frac{10^k}{k\ln 10}\right)\ . & \end{aligned} \tag{8}$$

Zusammen mit (4) ergeben sich damit die unteren und oberen Schranken für r_k:

$$\underline{r}_k \lesssim r_k \lesssim \overline{r}_k\ , \tag{9}$$

wobei

$$\underline{r}_k \to C \times 0.9^k \times \left(\frac{1}{(k+1)\ln 10} - \frac{0.1}{k\ln 10}\right)\ , \tag{10}$$

$$\overline{r}_k \to C \times 0.9^k \times \left(\frac{10}{(k+1)\ln 10} - \frac{1}{k\ln 10}\right)\ . \tag{11}$$

Damit gilt, wenn man die Partialsumme s_{10^k} noch exakt berechnen kann,

$$s_{10^k} + \sum_{l=k}^{\infty} \underline{r}_k \lesssim s \lesssim s_{10^k} + \sum_{l=k}^{\infty} \overline{r}_k\ . \tag{12}$$

Eine weitere Steigerung der Genauigkeit kann erreicht werden, wenn man r_k nicht ueber eine ganze Zehnerpotenz laufen lässt (d.h. über die Zahlen 10^k bis 10^{k+1}), sondern von $m\times 10^k$ bis $(m+1)\times 10^k$, wobei $m \in \{1,\ldots,9\}$. Die daraus resultierenden (heuristischen) Schranken für s seien mit $\underline{s}_k$ und $\overline{s}^k$ bezeichnet. Für die praktische Durchführung werden die Reihen für $\underline{r}_k$ und $\overline{r}_k$ nach endlich vielen Gliedern abgebrochen; in vorliegenden Fall ist dies bei $k = 100$ geschehen, da hier die Reihenglieder bereits hinreichend abgeklungen sind.

Mit der erwähnten Zerteilung in Intervalle $m\times 10^k$ bis $(m+1)\times 10^k$ ergeben sich die folgenden Schranken für s:

k	$\underline{s}_k$	$\overline{s}_k$
3	2.037	2.208
4	2.084	2.186
5	2.094	2.175
6	2.100	2.166
7	2.104	2.158
8	2.107	2.152
9	2.110	2.147
10	2.111	2.143
11	2.112	2.139

Um die Güte der Schranken zu untersuchen, wird der nach oben und unten abgeschätzten Beitrag für Primzahlen mit k Ziffern mit dem tatsächlichem Beitrag verglichen:

k	untere	$s_{10^{k+1}} - s_{10^k}$	obere
3	0.143	0.170	0.184
4	0.098	0.117	0.126
5	0.071	0.081	0.092
6	0.054	0.060	0.069
7	0.042	0.046	0.053
8	0.033	0.036	0.042
9	0.026	0.028	0.034
10	0.021	0.023	0.027
11	0.018	0.019	0.022

Es ist dabei überraschend, dass der geschätzte Beitrag anfangs ziemlich nahe an der Intervallmitte liegt. Mit anschließender Konvergenzbeschleunigung auf die erhaltenen unteren und oberen Schranken erhält man

$$2.116 \leq s \leq 2.122 \tag{13}$$

Damit sind mit hoher Wahrscheinlichkeit die ersten drei Ziffern von s gegeben durch 2.11, insbesondere wenn man berücksichtigt, dass der tatsaechliche Beitrag mit wachsendem k der unteren heuristischen Schranke annähert.

5 Analytische Heuristik für bessere Schranken

Bei der Berechnung der diskreten Schranken wird der Beitrag r_k von $a(n)$ zu s durch

$$C \times 0.9^k \frac{\pi(10^{k+1}) - \pi(10^k)}{10^{k+1}} \lesssim r_k \lesssim C \times 0.9^k \frac{\pi(10^{k+1}) - \pi(10^k)}{10^k} \tag{14}$$

mit einer Konstanten C abgeschätzt, wobei das Interval anstelle einer gesamten Zehnerpotenz auf kleinere Teile geteilt worden ist. Lässt man die Länge dieser kleinen Teilintervalle gegen Null gehen, so erhält man analog den Riemannsummen die Integraldarstellung

$$C \times 0.9^k \int_{10^k}^{10^{k+1}} \frac{\pi'(x)}{x} \, \mathrm{d}x \tag{15}$$

für beide Schranken, wobei $\pi'(x)$ eine kontinuierliche Primzahldichte analog einer kontinuierlichen Wahrscheinlichkeitsdichte beschreibt.

Die diskrete Primzahlzählfunktion $\pi(x)$ kann nach Pierre Dusart durch

$$\frac{x}{\ln x}\left(1+\frac{1}{\ln x}\right) < \pi(x) < \frac{x}{\ln x}\left(1+\frac{1}{\ln x}+\frac{2.51}{(\ln x)^2}\right) \tag{16}$$

für $x \geq 355991$ kontinuierlich abgeschätzt werden, womit untere und obere Schranken $\underline{\pi}'(x)$, $\overline{\pi}'(x)$ für die Primzahldichte $\pi'(x)$ erhalten werden:

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{x}{\ln x}\left(1+\frac{1}{\ln x}\right)\right]}_{=:\underline{\pi}'(x)} \leq \pi'(x) \leq \underbrace{\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{x}{\ln x}\left(1+\frac{1}{\ln x}+\frac{2.51}{(\ln x)^2}\right)\right]}_{\overline{\pi}'(x)} \tag{17}$$

Mit (15) ergeben sich die verbesserten unteren und oberen Schranken für s als

$$\underline{s}_N^{\mathrm{ana}} = s_N + C \times \int_{10^{k+1}}^{\infty} 0.9^{\lfloor \log_{10}(x) \rfloor} \frac{\underline{\pi}'(x)}{x}\,\mathrm{d}x\ , \tag{18}$$

$$\overline{s}_N^{\mathrm{ana}} = s_N + C \times \int_{10^{k+1}}^{\infty} 0.9^{\lfloor \log_{10}(x) \rfloor} \frac{\overline{\pi}'(x)}{x}\,\mathrm{d}x\ . \tag{19}$$

Eine genauere Approximation an $\pi(x)$ als die beiden Schranken stellt auch noch der Ausdruck

$$\pi^{\mathrm{approx}} = \frac{x}{\ln x}\left(1+\frac{1}{\ln x}+\frac{2}{(\ln x)^2}\right) \tag{20}$$

dar, der in der folgenden Tabelle als beste Schätzung s_N^{ana} geführt wird und wie in (18) und (19) gebildet wird.

k	$\underline{s}_{10^k}^{\mathrm{ana}}$	$s_{10^k}^{\mathrm{ana}}$	$\overline{s}_{10^k}^{\mathrm{ana}}$
7	**2.11**816	**2.11**926	**2.11**936
8	**2.11**810	**2.11**880	**2.11**888
9	**2.11**811	**2.11**858	**2.11**864
10	**2.11**817	**2.11**849	**2.11**853
11	**2.11**822	**2.11**845	**2.11**848
12	**2.11**829	**2.11**845	**2.11**847

Die Konstante C ist dabei derart bestimmt worden, dass bestmögliche Übereinstimmung bei bereits bekannten Werten von s_N erreicht worden ist.

Um die Qualität der Schranken zu beurteilen, werden die tatächlichen Werte von $s_{10^{k+1}} - s_{10^l}$, d.h. den Beitrag durch Primzahlen mit k Stellen, mit den über die Integrale (18) und (19) berechneten Schranken sowie der dazwischen liegenden Schätzung $s_{10^{k+1}}^{\mathrm{ana}} - s_{10^k}^{\mathrm{ana}}$ verglichen:

k	untere	gesch.	$s_{10^{k+1}} - s_{10^l}$	obere
8	**0.03**59964	**0.036**2313	**0.0360053**	**0.036**2500
9	**0.0286**193	**0.028**7656	**0.0286692**	**0.028**7797
10	**0.023**0651	**0.0231**599	**0.0231233**	**0.0231**704
11	**0.018**7928	**0.0188**563	**0.0188577**	**0.0188**640

In allen Fällen stimmen zumindest zwei Ziffern überein, womit von einer Genauigkeit der Beiträge von etwa einem Prozent auszugehen ist. Da gemäß unseren bisherigen Resultaten $s - s_{10^12}$ etwa 0.1 beträgt, kann von einer Genauigkeit bis mindestens zur dritten Nachkommastelle ausgegangen werden. Dies entspricht auch den Stellen, die durch die beiden Schranken $\underline{s}_{10^k}^{\text{ana}}$ und $\overline{s}_{10^k}^{\text{ana}}$ gefunden worden sind. Somit sind vier Ziffern von s zu

$$\boxed{2.118}$$

bestimmt.