DER ZAPFHAHNALGORITHMUS NACH RABINOWITZ & WAGON

Seminar Computeralgebra: Pi
von zur Gathen & Nüsken

Florian Schoppmann

Wintersemester 2004/05

Zusammenfassung. In *American Mathematical Monthly* 102(3):195–203, 1995, stellen Rabinowitz und Wagon einen Zapfhahnalgorithmus (englisch 'spigot algorithm') vor, der die Dezimalziffern von π berechnet. Ein Zapfhahnalgorithmus liefert seine Ausgabe bereits während des Ablaufs und benutzt einmal ausgegebene Ziffern nicht weiter zur Berechnung der noch folgenden. Die Implementierung gestaltet sich sehr einfach und ist bei ausschließlicher Verwendung von Ganzzahl-Arithmetik möglich.
Stichworte. Pi, Spigot Algorithm, Zapfhahnalgorithmus, Dezimaldarstellung, Variable Basis, Basisumrechnung

Inhaltsverzeichnis

1. Einleitung

Rabinowitz & Wagon (1995) stellen einen bemerkenswerten Algorithmus zur Berechnung der Dezimalstellen von π vor, der auf der Reihendarstellung

$$\pi = \sum_{i=0}^{\infty} \frac{(i!)^2 2^{i+1}}{(2i+1)!} = \sum_{i=0}^{\infty} \frac{2 \cdot i!}{(2i+1)!!} \tag{1.1}$$

beruht – hierbei bezeichnet $(2i+1)!!$ das Produkt der ungeraden Zahlen bis $2i+1$. Der Algorithmus verwendet ausschließlich Ganzzahl-Arithmetik und erlaubt extrem kurze Implementierungen.

Arndt & Haenel (2000), Seite 37, geben das folgende C-Programm von Dik T. Winter, Achim Flammenkamp *et. al.* an, das π auf (genau) 15000 Stellen berechnet. Es verwendet dabei eine Modifikation, die in Abschnitt 4.1 vorgestellt wird.

```
a[52514],b,c=52514,d,e,f=1e4,g,h;
main(){for(;b=c-=14;h=printf("%04d",e+d/f))
for(e=d%=f;g=--b*2;d/=g)d=d*b+f*(h?a[b]:f/5),a[b]=d%--g;}
```

Rabinowitz und Wagon nennen den Algorithmus „spigot algorithm“ – übersetzbar beispielsweise mit Zapfhahnalgorithmus –, da er seine Lösung „tröpfchenweise“ und bereits während seines Ablaufes ausgibt. Einmal ausgegebene Dezimalstellen von π werden dabei nicht weiter vom Algorithmus verwendet. (Diese Formulierung ist nicht ganz exakt: Eigentlich wird nur eine Partialsumme der obigen Reihendarstellung als Näherung für π berechnet, wie später gezeigt wird.)

Zur Erklärung des Algorithmus werden zuerst in Abschnitt 2 die mathematischen Grundlagen und später benötigten Sätze abgehandelt und in Abschnitt 3 dann die Funktionsweise erklärt. Dieser Aufbau soll erzielen, dass die Erläuterung des Algorithmus ohne störende Unterbrechungen erfolgen kann, sondern ausschließlich unter Verwendung bereits zuvor gewonnener Erkenntnisse.

Im Abschnitt 4 schließlich werden einige mögliche Erweiterungen und Verbesserungen des Algorithmus kurz skizziert.

2. Mathematische Grundlagen

2.1. Reihendarstellung von Pi. Zum Korrektheitsbeweis der in (1.1) gegebenen Reihendarstellung von π wird vorab folgendes technische Lemma benötigt:

LEMMA 2.1. *Seien $i, j \in \mathbb{N}_0$. Dann gilt:*

$$\int_0^1 x^i(1-x)^j dx = \frac{i! \cdot j!}{(i+j+1)!}$$

BEWEIS. Nachrechnen ergibt:

$$\begin{aligned}\int_0^1 x^i(1-x)^j dx &= \frac{1}{i+1}x^{i+1}(1-x)^j\Big|_0^1 - \int_0^1 \frac{-j}{i+1}x^{i+1}(1-x)^{j-1}dx \\ &= \frac{j}{i+1}\int_0^1 x^{i+1}(1-x)^{j-1}dx \qquad (2.2)\end{aligned}$$

Induktiv folgt daraus:

$$\begin{aligned}\int_0^1 x^i(1-x)^j dx &\overset{(2.2)}{=} \prod_{k=1}^{j}\frac{j-k+1}{i+k}\underbrace{\int_0^1 x^{i+j}dx}_{\frac{1}{i+j+1}} \\ &= \frac{j!}{\prod_{k=1}^{j+1} i+k} = \frac{i!\cdot j!}{(i+j+1)!}\end{aligned}$$

□

SATZ 2.3. *Die in (1.1) angegebene Reihendarstellung von π ist korrekt.*

BEWEIS. Der Beweis wird geführt wie von Li *et al.* (1949), Seite 633, jedoch mit einigen weiteren Erläuterungen und ohne Verwendung von Sätzen über Beta- und Gamma-Funktion, sondern mit Lemma 2.1. Ausgehend von der rechten Seite der zu zeigenden Gleichung ergibt sich:

$$\begin{aligned}\sum_{i=0}^{\infty}\frac{(i!)^2 2^{i+1}}{(2i+1)!} &= \sum_{i=0}^{\infty} 2^{i+1}\int_0^1 x^i(1-x)^i dx \quad \text{(nach Lemma 2.1)} \\ &= 2\int_0^1 \sum_{i=0}^{\infty}(2x)^i(1-x)^i dx = 2\int_0^1 \sum_{i=0}^{\infty}(\underbrace{2x(1-x)}_{\leq 1/2})^i dx \qquad (2.4) \\ &= 2\int_0^1 \frac{1}{1-2x(1-x)}dx \\ &= \int_0^1 \frac{1}{(x-1/2)^2+1/4}dx \\ &= 2\arctan 2(x-1/2)\Big|_0^1 = 2\arctan 1 - 2\arctan(-1) = \pi\end{aligned}$$

Schritt (2.4) ist erlaubt, da die Funktionenfolge $(f_n : \mathbb{R} \to \mathbb{R})_{n\in\mathbb{N}}$ mit $f_n(x) := \sum_{i=0}^{n}(2x)^i(1-x)^i$ auf $[0,1]$ gleichmäßig konvergent ist. Dies ist leicht mit dem Cauchy'schen Konvergenzkriterium einzusehen:

Sei $\epsilon > 0$. Wähle N mit $2^{-N} < \epsilon$. Dann gilt für $m \geq n \geq N$ und jedes $x \in [0,1]$:

$$
\begin{aligned}
|f_m(x) - f_n(x)| &= \sum_{i=n+1}^{m} (2x)^i(1-x)^i \\
&\leq (2x)^n(1-x)^n \qquad (2.5)\\
&\leq \frac{1}{2^n} \leq \frac{1}{2^N} < \epsilon \qquad (2.6)
\end{aligned}
$$

Begründung für (2.5) und (2.6): $(2x)^i(1-x)^i \geq 2(2x)^{i+1}(1-x)^{i+1}$ für alle $i \in \mathbb{N}$, da $0 \leq 2x(1-x) \leq 1/2$ für $x \in [0,1]$. □

2.2. Darstellungen bezüglich variabler Basis. Prinzipiell lässt sich die Funktionsweise des Algorithmus bereits anhand der in (1.1) gegebenen Reihendarstellung zusammenfassen: Es muss von der Darstellung

$$
\pi = \sum_{i=0}^{\infty} \frac{2 \cdot i!}{(2i+1)!!} = 2 + \frac{1}{3}\Big(2 + \frac{2}{5}\Big(2 + \frac{3}{7}\big(2 + \dots\big)\Big)\Big) \qquad (2.7)
$$

umgerechnet werden in die Dezimaldarstellung

$$
\pi = 3{,}141\cdots = 3 + \frac{1}{10}\Big(1 + \frac{1}{10}\Big(4 + \frac{1}{10}\big(1 + \dots\big)\Big)\Big).
$$

Zur weiteren Erläuterung wird folgende Definition eingeführt:

DEFINITION 2.8. *Sei $(a_n)_{n\in\mathbb{N}_0}$ Folge natürlicher Zahlen und $b := (b_n)_{n\in\mathbb{N}}$ Folge positiver rationaler Zahlen.*

(i) Dann heißt

$$
a_0 + b_1(a_1 + b_2(a_2 + b_3(a_3 + \dots)))
$$

Darstellung bezüglich der variablen Basis b.

(ii) Für die Darstellung in (i) ist $(a_0; a_1, a_2, \dots)_b$ eine Kurzschreibweise mit Semikolon als Trenner zwischen Ganzzahl- und Fraktionalteil.

(iii) Analog zur vorangehenden Definition heißt die Basis b fix, *wenn $b_1 = b_2 = \dots$.*

BEMERKUNG 2.9. *Diese Definition ließe sich natürlich auch entsprechend auf den Ganzzahlteil erweitern – beispielsweise, indem man die ganzen Zahlen zur Indizierung der Basiselemente verwendet. In dieser Arbeit wird eine solche Erweiterung jedoch nicht benötigt, so dass hier nur ein Beispiel zur Motivation angeben wird: Bei geeigneter Definition und Wahl der Basis* $t = (\dots, 24, 60, 60, \dots)$ *wäre etwa* $(3, 4, 0, 15)_t$ *Sekunden eine Darstellung für 3 Tage, 4 Stunden und 15 Sekunden.*

Im Folgenden soll c immer die Basis $(\frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \dots)$ bezeichnen. Die in (2.7) gezeigte Darstellung lässt sich so auffassen als Darstellung bezüglich der variablen Basis c, in der Kurzschreibweise: $(2; 2, 2, 2, \dots)_c$. Insbesondere heißt dies, dass die Wertigkeit jeder Stelle ein variabler Bruchteil der vorangegangenen ist. Genauer gilt: In (2.7) ist die Wertigkeit der Stelle i genau $\frac{i}{2i+1}$ der Stelle $i-1$ (für $i = 1, 2, 3, \dots$). Im Dezimalsystem hingegen ist die Wertigkeit jeder Stelle genau $\frac{1}{10}$ der vorangehenden Stelle.

DEFINITION 2.10. *Sei* $(a_0; a_1, a_2, a_3, \dots)_c$ *Darstellung bezüglich der Basis* c.

(i) Dann heißt die Darstellung regulär, *wenn* $a_0 \in \mathbb{N}_0$ *und für alle* $i = 1, 2, 3, \dots$ *gilt:* $0 \leq a_i < 2i + 1$. *Ein* a_i *heißt dabei* maximale Ziffer, *wenn* $a_i = 2i$.

(ii) Die Darstellung heißt abbrechend, *wenn es ein* $N \in \mathbb{N}$ *gibt, so dass* $a_i = 0$ *für alle* $i \geq N$.

BEMERKUNG 2.11. *Die Bezeichnung* normalisiert *wird bewusst nicht verwendet, denn Darstellungen bezüglich der Basis* c *sind* nicht *eindeutig. Ein Beispiel ist:*

$$(0; 2, 0, 0, 0, \dots)_c = \frac{2}{3} = (0; 0, 2, 3, 4, \dots)_c$$

Auch im Dezimalsystem sind nicht alle Darstellungen eindeutig, denn $1 = 0, \overline{9}$. *Eindeutigkeit ist hier aber gegeben, falls man Darstellungen ausschließt, die ab einer Stelle nur noch maximale Ziffern haben (im Dezimalsystem ist die 9 an jeder Fraktional-Stelle maximale Ziffer). Ebenso ist natürlich Eindeutigkeit gegeben, wenn man nur abbrechende Darstellungen betrachtet.*

SATZ 2.12. $(0; 2, 4, 6, \dots)_c$, *also die Darstellung bezüglich der Basis* c *mit maximalen Ziffern, repräsentiert 2. Demnach liegen reguläre Darstellungen der Form* $(0; a_1, a_2, a_3, \dots)_c$ *im Interval* $[0, 2]$.

BEWEIS. Die Herleitung und Beweisführung folgt dem Vorgehen von Rabinowitz & Wagon (1995), Seite 201 f.

Setze

$$S_n := \sum_{i=0}^{n} \frac{(2i)i!}{(2i+1)!!}$$

Anhand von (1.1) sieht man leicht, dass zu zeigen ist: $\lim_{n\to\infty} S_n = 2$. Nachrechnen führt zu der Vermutung, dass

$$2 - S_n = \frac{2^{n+1}}{\binom{2n+1}{n}} = \frac{2(n+1)!}{(2n+1)!!} \tag{2.13}$$

Der Beweis dafür wird per Induktion über $n = 0, 1, 2, \ldots$ geführt. Offensichtlich ist der Induktionsanfang für $n = 0$ erfüllt:

$$2 - S_n = \frac{2(0+1)!}{(2 \cdot 0 + 1)!!} = 2$$

Es bleibt der Induktionsschritt zu zeigen:

$$\begin{aligned} 2 - S_{n+1} &= \frac{2(n+1)!}{(2n+1)!!} - \frac{(2(n+1))(n+1)!}{(2(n+1)+1)!!} \quad \text{(nach IV: (2.13))} \\ &= \frac{2(2n+3)(n+1)!}{(2n+3)!!} - \frac{(2n+2)(n+1)!}{(2n+3)!!} \\ &= \frac{(2n+4)(n+1)!}{(2n+3)!!} = \frac{2(n+2)!}{(2n+3)!!} \end{aligned}$$

Da der Ausdruck (2.13) offensichtlich für $n \to \infty$ gegen 0 strebt, folgt die Behauptung. □

KOROLLAR 2.14. *Ist $(0; a_1, a_2, a_3, \ldots)_c$ eine reguläre und abbrechende Darstellung einer Zahl x, so liegt x im Intervall $[0, 2)$.*

BEWEIS. Dies ist nach dem letzten Satz klar. □

KOROLLAR 2.15. *Ist $(a_0; a_1, a_2, a_3, \ldots)_c$ eine reguläre und abbrechende Darstellung einer Zahl x, so gilt: $x \in [a_0, a_0 + 2)$. Der Ganzzahlteil von x ist also entweder a_0 oder $a_0 + 1$.*

BEWEIS. Dies folgt sofort aus dem vorangegangenen Korollar. □

SATZ 2.16. *Sei* $\pi_m := (\underbrace{2;2,2,2,\ldots,2}_{m \text{ Ziffern}})_c$ *für ein* $m \in \mathbb{N}$. *Dann gilt:*

$$0 < \pi - \pi_m < \frac{8}{3} \cdot 10^{-3m/10}$$

BEWEIS. Man verwendet die Reihendarstellung (1.1), um zu sehen, dass $\pi - \pi_m$ genau der (stets positive) Rest einer Partialsumme ist.

$$\pi - \pi_m = \sum_{i=m}^{\infty} \frac{(i!)^2 2^{i+1}}{(2i+1)!} \tag{2.17}$$

$$< \frac{(m!)^2 2^{m+2}}{(2m+1)!} = \frac{4m!}{(2m+1)!!} \tag{2.18}$$

$$\leq \frac{2}{3} \cdot \frac{4m!}{2 \cdot 4 \cdots (2m)} = \frac{2}{3} \cdot \frac{4m!}{2^m m!} = \frac{8}{3} \cdot 2^{-m} \tag{2.19}$$

$$< \frac{8}{3} \cdot 10^{-3m/10} \tag{2.20}$$

Die Abschätzung in (2.18) gilt, da jedes Glied der Reihe in (2.17) weniger als halb so groß wie sein Vorgänger ist. Folglich lässt sich die Reihe (2.17) durch das Doppelte ihres ersten Gliedes nach oben abschätzen. In (2.19) gilt Gleichheit genau dann, wenn $m = 1$. Die Abschätzung ist daher leicht einzusehen. Ungleichung (2.20) folgt schließlich, da $2^x = (2^{10/3})^{3x/10} > 10^{3x/10}$ für $x \in \mathbb{Q}_{>0}$. □

KOROLLAR 2.21. *Sei* $n \in \mathbb{N}$. *Mit* $m := \lceil \frac{10n}{3} \rceil$ *gilt* $\pi - \pi_m < \frac{8}{3} \cdot 10^{-n}$.

BEWEIS. Einsetzen in den letzten Satz liefert das gewünschte Ergebnis:

$$\pi - \pi_m \overset{Satz\ 2.16}{<} \frac{8}{3} \cdot 10^{-3\lceil \frac{10n}{3} \rceil/10} \leq \frac{8}{3} \cdot 10^{-3\frac{10n}{3}/10} = \frac{8}{3} \cdot 10^{-n}$$ □

3. Funktionsweise des Algorithmus

Nimmt man für einen Augenblick an, dass nur Zahlen kleiner als 1 von einer regulären und abbrechenden Darstellung der Form $(0; a_1, a_2, a_3, \ldots)_c$ repräsentiert werden, so ließe sich – wie in Algorithmus 3.1 zu sehen – ein einfaches iteratives Verfahren zur Bestimmung der Dezimaldarstellung einer Zahl $(a_0; a_1, a_2, a_3, \ldots, a_n, 0, \ldots)_c$ angeben:

ALGORITHMUS 3.1. Hypothetischer Algorithmus zur Umrechung in die Dezimaldarstellung.

Eingabe: Die umzurechnende Darstellung $(a_0; a_1, a_2, a_3, \dots, a_n, 0, \dots)_c$, die entsprechende natürliche Zahl n.

Ausgabe: Gewünschte Stellen der Dezimaldarstellung.

---

1. Wiederhole 2–4
2. Gebe den Ganzzahlteil aus (also a_0) – nach Annahme kann a_0 nur beim ersten Durchlauf größer als 9 sein. Das heißt, in jedem weiteren Schritt wird immer genau eine Dezimalstelle ausgegeben. Subtrahiere den Ganzzahlteil. (Setze: $a_0 \leftarrow 0$).
3. Multipliziere mit 10. Dies kann wie folgt geschehen: Multipliziere jede Ziffer $a_0, \dots, a_n$ mit 10 und regularisiere die Darstellung anschließend wieder von rechts nach links, indem jede Ziffer reduziert und ein eventuell vorhandener Übertrag nach links weitergereicht wird.
4. Speichere diese regularisierte Darstellung von $10 \cdot (0; a_1, a_2, a_3, \dots)_c$ wieder in $(a_0; a_1, a_2, a_3, \dots)_c$.
5. Bis die gewünschte Anzahl Dezimalstellen ausgegeben wurde (die gewünschte Genauigkeit wurde also erreicht).

BEMERKUNG 3.2. *Dieser hypothetische Algorithmus erfüllt bereits die in Abschnitt 1 genannten Eigenschaften eines Zapfhahnalgorithmus.*

3.1. Umrechnung in Dezimaldarstellung. Es soll nun überlegt werden, wie der gerade vorgestellte Algorithmus erweitert werden kann, so dass er auch mit den tatsächlichen Eigenschaften einer Darstellung bezüglich der Basis c funktioniert.

Offensichtlich dürfen Ziffern erst ausgegeben werden, wenn ihr Wert bereits fest steht. Dies ist für alle Ziffern ab der Zehner-Stelle der Fall, wenn $(a_0 \mod 10)$ ungleich 9 ist. Denn dann kann sich an den Ziffern von $\lfloor a_0/10 \rfloor$ nach Korollar 2.15 nichts mehr ändern.

BEMERKUNG 3.3. *Dies deckt noch nicht alle Fälle ab, in denen Ziffern ausgegeben werden können. Ein Beispiel: Ist der aktuelle Übertrag auf a_0 gleich 19, so können sogar alle Ziffern von a_0 ausgegeben werden. Denn nach Korollar 2.14 kann es in den nächsten Schleifen-Durchläufen keinen Übertrag mehr geben, der sich auf die aktuellen Ziffern von a_0 noch auswirkt.*

Da nach der beschriebenen Idee nur Ziffern links der Einer-Stelle ausgegeben werden, darf jeweils nach der Ausgabe a_0 nur modulo 10 reduziert und nicht mehr grundsätzlich auf 0 gesetzt werden. Bei der Implementierung

in Programmiersprachen mit einem begrenzten Wertebereich für Ganzzahlvariablen (wie beispielsweise C oder Pascal) ist allerdings zu beachten, dass es zu einem Überlauf kommen könnte, wenn in hinreichend vielen Schleifen-Durchläufen hintereinander $(a_0 \mod 10) = 9$ ist. Hier kann man sich wie folgt behelfen:

Entscheidet man anhand der Zehner-, und nicht mehr anhand der Einer-Ziffer von a_0, ob die Ziffern davor (in letzterem Fall also $\lfloor a_0/100 \rfloor$) ausgegeben werden, hat man folgenden Vorteil. Nach Subtraktion der ausgegeben Ziffern (multipliziert mit ihrer Wertigkeit), lässt sich die Dezimaldarstellung von $\lfloor a_0/10 \rfloor$ – ihre Ziffern seien als *Vorziffern* bezeichnet – sehr kompakt repräsentieren: Durch die führende Ziffer sowie die Anzahl folgender Neunen. (Beim zuvor beschriebenen Vorgehen ließe sich $\lfloor a_0/10 \rfloor$ nicht so kompakt darstellen.)

Es ergibt sich schließlich folgende Modifikation von Schritt 2 in Algorithmus 3.1:

- Reduziere a_0 modulo 10 (das heißt, setzte: $a_0 \leftarrow a_0 \mod 10$). a_0 kann maximal $109 = 9 \cdot 10 + 19$ sein. Der Quotient sei mit q bezeichnet.
- Wenn $q = 10$, addiere 1 auf jede Vorziffer (9 wird zu 0). Gebe alle Vorziffern aus. Neue (einzige) Vorziffer wird 0.
- Wenn $q = 9$, füge q zu den Vorziffern hinzu.
- Andernfalls gebe alle Vorziffern aus. Neue (einzige) Vorziffer wird q.

Bemerkung 3.4. *Es kann wünschenswert sein, nicht die tatsächlichen ersten n Dezimalziffern der im Allgemeinen nicht-abbrechenden Dezimaldarstellung auszugeben, sondern n* gültige Stellen*, also eine auf n Stellen gerundete Ausgabe. Hier würde der Algorithmus die n-te Stelle zurückhalten und erst (gegebenenfalls gerundet) ausgeben, wenn auch die $(n+1)$-te Stelle feststeht.*

3.2. Der Zapfhahnalgorithmus für Pi. Ausgerüstet mit den bisherigen Erkenntnissen kann jetzt ein Zapfhahnalgorithmus zur Bestimmung der Dezimalstellen von π angegeben werden. Der folgende ist dabei angelehnt an den Algorithmus von Rabinowitz & Wagon (1995), Seite 198. Er gibt jedoch die letzte Dezimalstelle stets aufgerundet aus und passt daher zum späteren Korrektheitsbeweis.

ALGORITHMUS 3.5. Zapfhahnalgorithmus für π.

Eingabe: Natürliche Zahl n, die angibt, wie viele Stellen ausgegeben werden sollen.

Ausgabe: Die ersten n Stellen der Dezimaldarstellung von π_m, wobei $m = \lceil 10n/3 \rceil$, mit einem Fehler $< 12\frac{2}{3} \cdot 10^{-n}$.

1. Initialisierung: Setze: $A := (a_0; a_1, a_2, \ldots, a_{m-1})_c \leftarrow (2; 2, 2, \ldots, 2)_c$ mit $m := \lceil 10n/3 \rceil$. Setze (einzige) Vorziffer auf 0.
2. Für $i \leftarrow 1, 2, \ldots, n$ erledige 3–11
   { i bezeichnet die Anzahl (vorläufig) errechneter Dezimalstellen am Ende der Schleife }
3. Multipliziere jedes Element von A mit 10.
4. Regularisiere A: Von rechts beginnend, reduziere für $k = m-1, \ldots, 1$ jedes Element a_k modulo $2k+1$. (Basiselement c_k ist $k/(2k+1)$). Bezeichne q den Quotienten der Reduktion, so erhalte als Übertrag für die nächste Stelle $q \cdot k$.
   { Der letzte Übertrag auf den Ganzzahlteil kann maximal 19 betragen. }
5. Reduziere a_0 modulo 10. q bezeichne im Folgenden den Quotienten.
6. Falls $q = 9$ dann
7. Füge q zu den Vorziffern hinzu.
8. Sonst wenn $q = 10$ dann
9. Addiere 1 auf jede Vorziffer (9 wird zu 0). Gebe alle Vorziffern aus. Neue (einzige) Vorziffer wird 0.
10. Sonst
11. Gebe alle Vorziffern aus. Neue (einzige) Vorziffer wird q.
12. Wenn letzte Vorziffer = 9, addiere 1 auf jede Vorziffer. Andernfalls addiere 1 nur auf die letzte Vorziffer.
13. Gebe alle Vorziffern aus.

BEMERKUNG 3.6. *Bei der Implementierung ist natürlich sicher zu stellen, dass es nicht aufgrund beschränkter Wertebereiche der Ganzzahlvariablen zu Überläufen kommt. Durch Ausprobieren stellt man aber fest, dass der Algorithmus mit „relativ" kleinen Zahlen rechnet. Die von Rabinowitz & Wagon (1995), Seite 202 f., angegebene Implementierung (die Algorithmus 3.5 bis auf Schritt 12 entspricht) verwendet zur Bestimmung der ersten 5.000 Dezimalziffern von π ausschließlich Ganzzahlarithmetik im Bereich bis* 631.520.

LEMMA 3.7. *(i) Algorithmus 3.5 ist korrekt. Insbesondere liefert er n Stellen von π mit einem Fehler $< 12\frac{2}{3} \cdot 10^{-n}$.*

(ii) Der Zapfhahnalgorithmus hat ferner lineare Konvergenz sowie bezüglich der Eingabe n quadratische Laufzeit und linearen Speicherplatz-Verbrauch.

BEWEIS. Dass der Algorithmus immer terminiert, ist offensichtlich.

Nach Korollar 2.21 gilt: $\pi - \pi_m < \frac{8}{3} \cdot 10^{-n}$. Schritt 12 in Algorithmus 3.5 führt dazu, dass die ausgegebene Näherung $\widetilde{\pi}_m$ von π_m einen Fehler von maximal 10^{-n+1} aufweisen kann. Addition der beiden maximalen Fehler liefert: $|\pi - \widetilde{\pi}_m| < 12\frac{2}{3} \cdot 10^{-n}$, wie behauptet.

Die lineare Konvergenz des Algorithmus ist offensichtlich, da in jedem Schritt durchschnittlich eine Dezimalstelle ausgegeben wird.

Die quadratische Laufzeit lässt sich intuitiv durch das Füllen einer Tabelle mit $\mathcal{O}(n) \cdot \mathcal{O}(n)$ Einträgen erklären (siehe auch Tafel 3.1) – im Algorithmus befindet sich eine Schleife in einer Schleife, wobei die Schleifenzähler jeweils die Zahlen $1, \ldots, n$ durchlaufen.

Der Speicherplatzverbrauch beträgt $\mathcal{O}(n)$, wie leicht im Algorithmus abzulesen. Intuitiv: Die Zeilen der Tabelle werden nur für den jeweils nächsten Schritt verwendet und gespeichert. □

BEMERKUNG 3.8. *Anhand der oberen Schranke des Fehlers lässt sich hinterher natürlich eine untere Schranke für die Anzahl korrekt ausgegebener Ziffern finden: Finde von rechts ausgehend, die erste Stelle, die auch bei Addition oder Subtraktion von $12\frac{2}{3} \cdot 10^{-n}$ (auf beziehungsweise von $\widetilde{\pi}_m$) unverändert bleibt.*

Arndt & Haenel (2000), Seite 79 ff., behaupten, dass für $m = \lfloor \frac{10n}{3} + 1 \rfloor$ (sic!) Algorithmus 3.5 tatsächlich n korrekte Stellen ausgibt, sofern Schritt 12 weggelassen und der Zähler n nicht in Schritt 2 inkrementiert wird, sondern immer erst bei der Ausgabe von Ziffern. Sie geben jedoch weder einen mathematischen Beweis, noch zeigen sie, dass der Algorithmus so weiterhin terminiert. Allerdings schreiben sie, diese Vermutung getestet zu haben – erklären lässt sich dies damit, dass einige der gezeigten Abschätzungen zwar korrekt, für viele Eingaben jedoch „konservativer“ als eigentlich nötig sind.

3.3. Ein Beispieldurchlauf. Im Folgenden soll der Algorithmus an einem einfachen Beispiel illustiert werden. Aufgrund der Tatsache, dass nur die ersten drei Stellen von π berechnet werden sollen, kann in jedem Iterationsschritt eine Dezimalziffer von π ausgegeben werden. (Das heißt, die Idee der Vorziffern wie im Algorithmus 3.5 ist hier noch nicht nötig.)

Die ersten beiden Zeilen der Tabelle dienen der Initialisierung. In den folgenden 5er Zeilenblöcken wird die Tabelle jeweils von rechts oben nach links

Stelle i:	0	1	2	3	4	5	6	7	8	9
Alte Basis:	1	$\frac{1}{3}$	$\frac{2}{5}$	$\frac{3}{7}$	$\frac{4}{9}$	$\frac{5}{11}$	$\frac{6}{13}$	$\frac{7}{15}$	$\frac{8}{17}$	$\frac{9}{19}$
Ziffern von π:	2	2	2	2	2	2	2	2	2	2
·10:	20	20	20	20	20	20	20	20	20	20
Übertrag:	10	12	12	12	10	12	7	8	9	0
Summe:	30	32	32	32	30	32	27	28	29	20
Quotient:	3	10	6	4	3	2	2	1	1	1
Rest:	0	2	2	4	3	10	1	13	12	1
·10:	0	20	20	40	30	100	10	130	120	10
Übertrag:	13	20	33	40	60	42	84	56	0	0
Summe:	13	40	53	80	90	142	94	186	120	10
Quotient:	1	13	10	11	10	12	7	12	7	0
Rest:	3	1	3	3	0	10	3	6	1	10
·10:	30	10	30	30	0	100	30	60	10	100
Übertrag:	10	20	21	24	55	30	35	24	45	0
Summe:	40	30	51	54	55	130	65	84	55	100
Quotient:	4	10	10	7	6	11	5	5	3	5
Rest:	0	0	1	5	1	9	0	9	4	5

Tafel 3.1: Tabellarische Illustration des Zapfhahnalgorithmus

unten aufgebaut: Es wird mit der Multiplikation mit 10 rechts begonnen und anschließend nach links regularisiert. Man beachte, dass an Stelle 0 modulo 10 reduziert wird.

Nach Satz 2.16 gilt als Fehlerabschätzung: $|\pi - \pi_{10}| < \frac{8}{3} \cdot 10^{-3} = 0{,}002\overline{6}$. Algorithmus 3.5 hätte 3,15 ausgegeben, da er die letzte Stelle stets aufrundet. Nach Lemma 3.7(i) liegt der Fehler bei weniger als $12\frac{2}{3} \cdot 10^{-3} = 0{,}012\overline{6}$. Nachrechnen bestätigt dies: $|\pi - \widetilde{\pi}_{10}| \approx 0{,}0084$.

4. Ausblick

4.1. Erweiterungen des Zapfhahnalgorithmus. Es lassen sich Laufzeitverbesserungen des Zapfhahnalgorithmus um einen konstanten Faktor k erzielen, indem in jedem Schleifen-Durchlauf nicht mit 10, sondern mit 10^k multipliziert wird (Algorithmus 3.5 Schritt 3). Dadurch werden in jedem Durchgang durchschnittlich k neue Dezimalstellen berechnet. – Das Prinzip hierbei ist, dass der Ganzzahlteil links von der Einer-Stelle im (10^k)er- und nicht mehr im 10er-System berechnet wird. Jede „Ziffer“ im (10^k)er-System liegt dabei zwischen 0 und $10^k - 1$ und repräsentiert somit k Dezimalziffern.

Jede Vorziffer ist also eine Zahl in diesem Bereich. Entsprechend muss in Schritt 6 überprüft werden, ob $q = 10^k - 1$ und in Schritt 8, ob $q \geq 10^k$. Bei Schritt 9 wäre noch die Anpassung zu machen, dass die neue (einzige) Vorziffer $q - 10^k$ wird. Denn: Der maximale Übertrag auf den Ganzzahlteil ist nach Korollar 2.14 offenbar $2 \cdot 10^k - 1$. Damit wird a_0 maximal $9 \cdot 10^k + 2 \cdot 10^k - 1 = 11 \cdot 10^k - 1$ und der durch die anschließende Reduktion von a_0 modulo 10 entstehende Quotient q maximal $11 \cdot 10^{k-1} - 1$.

Das folgende Lemma erlaubt, den Algorithmus zu vereinfachen, wenn nur eine begrenzte Anzahl von Ziffern berechnet werden soll. Das auf Seite 2 vorgestellte Programm macht sich dies zu Nutze.

LEMMA 4.1. *Wird π_m vom (modifizierten) Zapfhahnalgorithmus im (10^k)er-System berechnet (m sei erklärt wie im Algorithmus 3.5), so kann es einen Übertrag, der mehr als eine Vorziffer beeinflusst, nur an einer durch k teilbaren Position der Dezimaldarstellung von π_m geben, an der mindestens $k+1$ Nullen stehen.*

BEWEIS. Im Algorithmus 3.5 wird ein Schleifendurchlauf (Schritte 2–11) betrachet, bei dem ein Übertrag auftritt. Ohne Einschränkung seien am Schleifenbeginn $d_1 = 1$ und $d_2 = 10^k - 1$ die beiden einzigen Vorziffern. Gezeigt wird, dass an der zu d_2 gehörigen Position der Dezimaldarstellung von π_m mindestens $k+1$ Nullen stehen.

Kommt es zu einem Übertrag (Schritt 8), so gilt $q \geq 10^k$. Folglich wird d_1 um 1 erhöht, d_2 wird zu 0. Dann werden alle Vorziffern ausgegeben und neue einzige Vorziffer wird $d_3 := q - 10^k < 10^{k-1}$. Die erste Dezimalziffer dieser Vorziffer d_3 ist aber 0 und wird sich später nicht mehr ändern. Denn würde sie sich ändern, so hätte am Anfang der Schleife gegolten:

$$\begin{aligned}&(\underbrace{d_1 \cdot 10^{k+1} + d_2 \cdot 10^1 + a_0}_{=1\cdot 10^{k+1}+(10^k-1)\cdot 10+a_0}; a_1, a_2, \dots)_c \\ &= (2 \cdot 10^{k+1} - 10 + a_0; a_1, a_2, \dots)_c \geq 2 \cdot 10^{k+1} + 0 \cdot 10^1 + 1\end{aligned}$$

Die letzte 1 ist hier die nachträglich um 1 inkrementierte erste Dezimalstelle der neuen Vorziffer d_3. Dies ist aber ein Widerspruch dazu, dass nach Korollar 2.15 der Ganzzahlteil einer regulären abbrechenden Darstellung der Form $(a_0; a_1, a_2, \dots)_c$ nur a_0 oder $a_0 + 1$ sein kann. Folglich muss es mindestens $k+1$ Nullen in der Dezimaldarstellung von π_m geben, damit sich mehr als eine Vorziffer ändert. In diesem Fall (aufgeschrieben als Dezimalziffernfolge): $1\,9\dots9$ wird zu $2\,0\dots0$.

Der zweite Teil der Behauptung ist klar, da offensichtlich jede Vorziffer an einer durch k teilbaren Position der Dezimaldarstellung von π_m steht. □

Falls also $k = 4$ und nur 50.000 Ziffern ausgegeben werden sollen, genügt bis zur Ausgabe das Abwarten immer nur einer Vorziffer – denn es müssten 5 aufeinander folgende Nullen an einer durch 4 teilbaren Position auftreten. Selbst für 4 aufeinander folgende Nullen ist dies jedoch erst bei Position 54.936 das erste Mal der Fall. (Vergleiche Arndt & Haenel (2000), Seite 83.)

Die asymptotische Laufzeit des Algorithmus im $\mathcal{O}$-Kalkül bleibt von der beschriebenen Erweiterung natürlich unberührt.

4.2. Zapfhahnalgorithmus für die Euler'sche Zahl. Man kann sich fragen, ob der hypothetische Algorithmus 3.1 eventuell gar nicht so hypothetisch ist, es also eine andere Basis gibt, für die er funktionieren würde.

Zumindest für die Euler'sche Zahl ist dies tatsächlich der Fall: Bezüglich der Basis $b := (\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots)$ lässt sich e offenbar darstellen als $(2; 1, 1, 1, \dots)_b$.

Die Definitionen aus Abschnitt 2.2 gelten dabei für die neue Basis b entsprechend: Bezüglich b heißt eine Darstellung $(a_0; a_1, a_2, a_3, \dots)_b$ *regulär*, wenn $a_0 \in \mathbb{N}$ und für alle $i = 1, 2, 3, \dots$ gilt: $0 \leq a_i \leq i$. Analog heißt eine Ziffer a_i *maximal*, wenn $a_i = i$. *Abbrechend* ist für Basis b identisch wie für Basis c erklärt.

LEMMA 4.2. *$(0; 1, 2, 3, \dots)_b$, also die Darstellung bezüglich der Basis b mit maximalen Ziffern, repräsentiert 1. Demnach liegen reguläre und abbrechende Darstellungen der Form $(0; a_1, a_2, a_3, \dots)_b$ im Interval $[0, 1)$.*

BEWEIS.

$$\begin{aligned}(0; 1, 2, 3, \dots)_b &= \sum_{i=2}^{\infty} \frac{i-1}{i!} \\ &= \frac{2-1}{2!} + \frac{3-1}{3!} + \frac{4-1}{4!} + \dots \\ &= \frac{2}{2!} - \frac{1}{2!} + \frac{3}{3!} - \frac{1}{3!} + \frac{4}{4!} - \frac{1}{4!} + \dots \\ &= 1 - \frac{1}{2!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{3!} - \frac{1}{4!} + \dots = 1\end{aligned}$$

Da es ein $N \in \mathbb{N}$ gibt mit $a_n = 0$ für alle $n \geq N$, folgt die Behauptung. □

Algorithmus 3.1 kann also zur Berechnung von e eingesetzt werden. Rabinowitz & Wagon (1995), Seite 201, zeigen, dass ab $n = 28$ sogar nur n Ziffern in der Darstellung bezüglich der Basis b benötigt werden, um mit relativ großer Wahrscheinlichkeit die ersten n richtigen Stellen von e zu ermitteln.

Es ist daher nicht verwunderlich, dass ein Zapfhahnalgorithmus zur Berechnung von e bereits 1968 von Sale angegeben wurde.

4.3. Zapfhahnalgorithmen auf Basis anderer Reihendarstellungen von Pi. Wie gesehen, ist eine aus rationalen Zahlen bestehende Basis d besonders für einen Zapfhahnalgorithmus geeignet, wenn jede Darstellung der Form $(a_0; a_1, a_2, a_3, \dots, 0, \dots)_d$ immer a_0 als Ganzzahlteil hat. Es drängt sich also die Frage auf, ob es auch für π eine solche Basis gibt.

Rabinowitz & Wagon (1995), Seite 199, verweisen auf eine Reihendarstellung, die von Gosper (1974) gefunden wurde:

$$\pi = 3 + \frac{1}{60}\Big(8 + \frac{2 \cdot 3}{3 \cdot 7 \cdot 8}\Big(\dots\Big(5i - 2 + \frac{i(2i-1)}{3(3i+1)(3i+2)}\big(\dots\big)\Big)\Big)\Big)$$

Diese Reihendarstellung führt zu der Basis $d := (\frac{1}{60}, \frac{6}{168}, \frac{15}{330}, \dots)$, bezüglich derer sich π darstellen lässt als: $(3; 8, 13, 18, \dots)_d$. Wird eine reguläre Darstellung wieder analog zu den bisherigen Fällen definiert, so haben Rabinowitz und Wagon nachgerechnet, dass $(0; a_1, a_2, a_3, \dots)_d \leq 1,092\dots$ sein muss.

Dies ist ein großer Vorteil: Mit der Annahme, dass die Ziffern von π gleichverteilt sind, war bislang die Wahrscheinlichkeit für einen Übertrag von mehr als 9 auf den Ganzzahlteil genau $\frac{1}{2}$. Mit Wahrscheinlichkeit $\frac{1}{20}$ ist also der Ganzzahlteil nach Multiplikation mit 10 und Addition des Übertrags ≥ 100 und die Vorziffern ändern sich. Mit der neuen Basis d ist die Wahrscheinlichkeit für einen Übertrag von mehr als 9 weniger als $\frac{1}{11}$. Somit können sich auch die Vorziffern mit Wahrscheinlichkeit weniger als $\frac{1}{110}$ ändern.

Rabinowith und Wagon bezeichnen diese Eigenschaft als „within 1% of spigot-perfection“. Zudem konvergiert die Reihe schneller, so dass man mit weniger Speichplatz auskäme. Zu beachten ist jedoch, dass zwischendurch mit größeren Zahlen gerechnet wird.

4.4. Ein Streaming-Algorithmus zur Berechnung von Pi. Betrachtet man Algorithmus 3.5, so stellt man fest, dass offenbar die Art der Regularisierung eine anfängliche Festlegung der zu berechnenden Stellen verlangt.

Gibbons (2004) erwähnt einen Streaming-Algorithmus zum Regularisieren bezüglich einer unendlichen variablen Basis (ohne ihn direkt anzugeben). Die Streaming-Idee führt er anschließend zu einem weiteren Zapfhahnalgorithmus zur Berechnung von π fort, den er auch explizit angibt.

Offensichtlich können auf diese Art und Weise die bisherigen Schwachstellen behoben werden, die aus der frühen Festlegung auf n zu berechnende Dezimalstellen von π resultierten – insbesondere die potentielle Ungenauigkeit von $12\frac{2}{3} \cdot 10^{-n}$. Allerdings verlangt der Streaming-Algorithmus Ganzzahl-Arithmetik mit unbeschränkter Genauigkeit. Nicht zuletzt aus diesem Grund gibt Gibbons daher sein Programm in Haskell an.

Literatur

Jörg Arndt & Christoph Haenel (2000). *π Algorithmen, Computer, Arithmetik*. Springer-Verlag, Berlin, Heidelberg, New York, 2nd edition. 2, 11, 14

Jeremy Gibbons (2004). Unbounded Spigot Algorithms for the Digits of π. URL http://web.comlab.ox.ac.uk/oucl/work/jeremy.gibbons/publications/spigot.pdf. 15

R. W. Gosper (1974). Acceleration of Series. *MIT Artificial Intelligence Laboratory Publications* **Memo 304**. URL ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-304.pdf. 15

Jerome C. R. Li, Ragnar Dybvik, Paul Carnahan, M. S. Klamkin, N. J. Fine & D. H. Browne (1949). E854. *American Mathematical Monthly* **56**(9), 633–635. URL http://links.jstor.org/sici?sici=0002-9890281949112956 3A93C6333AE3E2.0.CO3B2-7. 3

Stanley Rabinowitz & Stan Wagon (1995). A Spigot Algorithm for the Digits of π. *American Mathematical Monthly* **102**(3), 195–203. URL http://links.jstor.org/sici?sici=0002-98902819950329102 3A33C1953AASAFTD3E2.0.CO3B2-H. 2, 6, 9, 10, 14, 15

A. H. J. Sale (1968). The calculation of e to many significant digits. *The Computer Journal* **11**(2), 229–230. URL http://www3.oup.co.uk/computer_journal/hdb/Volume_11/Issue_02/110229.sgm.abs.html. 14

Florian Schoppmann
Dörener Weg 61
33100 Paderborn
fschopp@upb.de