12. Musterlösung zu Mathematik für Informatiker II, SS 2004

MARTIN LOTZ & MICHAEL NÜSKEN

Aufgabe 12.1 (Reihen). (4+2 Punkte)

Zeige, dass die folgenden Reihen absolut konvergieren:

(i) $\displaystyle\sum_{k=1}^{\infty} \frac{1}{k^4 + z^4}$ für $z \in \mathbb{R}$.

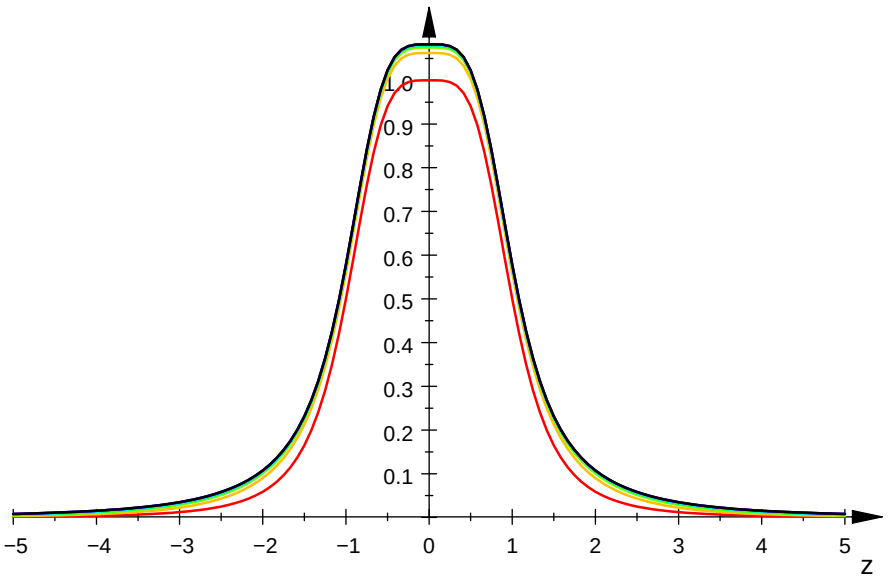
Lösung. Die Reihe konvergiert für jedes $z \in \mathbb{R}$ absolut, sie wird durch die Reihe $\sum_{k=1}^{\infty} \frac{1}{k^4}$ majorisiert. ◯

(ii) $\displaystyle\sum_{k=1}^{\infty} k^{-2} \cos(kz)$ für $z \in \mathbb{R}$.

Lösung. Die Reihe konvergiert für jedes $z \in \mathbb{R}$ absolut. Da immer $|\cos(kz)| \leq 1$ gilt, wird diese Reihe der Beträge durch die Reihe $\sum_{k=1}^{\infty} \frac{1}{k^2}$ majorisiert. ◯

Zusatz: Zeichne (mit Hilfe von MuPAD oder Maple) für beide Reihen den Graphen der Partialsummen bis $k = 1, 2, 3, 4, 5, 10, 20, 50$ im Intervall $[-5, 5]$.

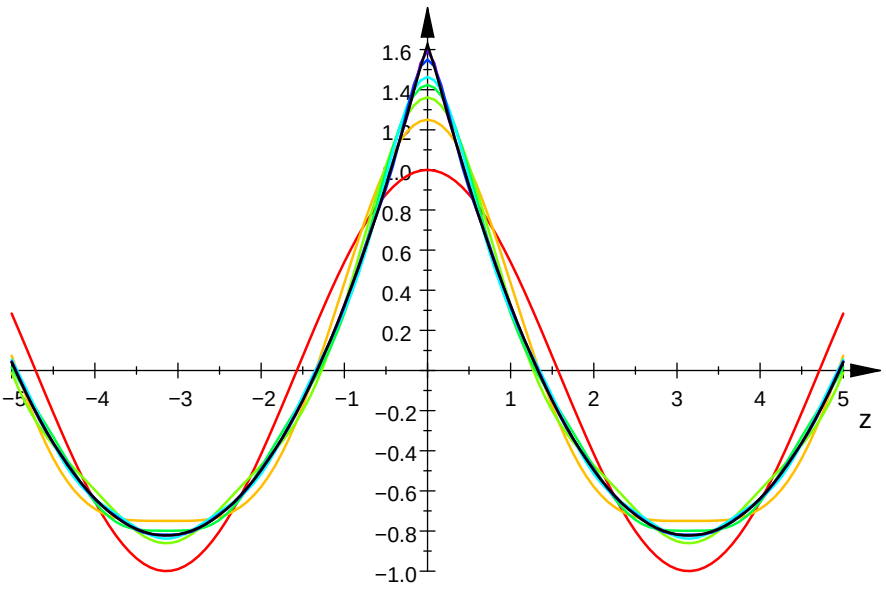
Lösung.



Zur Information: Nach einigem Überreden liefert Maple

$$\sum_{k=1}^{\infty} \frac{1}{k^4+z^4} = \frac{\pi\sqrt{2}\left(\sinh\left(\pi z\sqrt{2}\right)+\sin\left(\pi z\sqrt{2}\right)\right)}{4\left(\cosh\left(\pi z\sqrt{2}\right)-\cos\left(\pi z\sqrt{2}\right)\right)z^3} - \frac{1}{2z^4}.$$

Das ist so natürlich bestenfalls für $z \neq 0$ gültig. Das sagt einem Maple allerdings nicht, darauf muss man selbst achten… Für $z = 0$ ergibt sich der Wert $\frac{\pi^4}{90} \approx 1.0823$, was auch Maple problemlos antwortet, sobald man danach fragt. Dieser Wert stellt auch das Maximum der Funktion dar.



Zur Information: Für $z \in [0, 2\pi]$ gilt $\sum_{k=1}^{\infty} k^{-2}\cos(kz) = \frac{1}{4}(z-\pi)^2 - \frac{1}{12}\pi^2$. (Eine Gleichung, die Maple gar nicht und MuPAD nur näherungsweise rausgerückt hat.) Es ist also kein Zufall, dass es so aussieht, als würde sich das Ganze einer Aneinanderreihen von Parabelstücken annähern. ◯

Aufgabe 12.2 (Konvergenzradius). (4 Punkte)

Bestimme den Konvergenzradius folgender Potenzreihen.

(i) $\displaystyle\sum_{n=1}^{\infty} \left(\frac{z}{n}\right)^n$.

Lösung. Wir benutzen den Satz aus der Vorlesung, wonach der Konvergenzradius einer Potenzreihe $\sum_{n=0}^{\infty} a_n z^n$ durch

$$R = \frac{1}{\lim\limits_{n\to\infty} \sqrt[n]{a_n}}$$

gegeben ist (was für Grenzwert 0 bzw. ∞ geeignet zu lesen ist). In unserem Fall ist

$$\lim_{n\to\infty} \sqrt[n]{\left(\frac{1}{n}\right)^n} = \lim_{n\to\infty} \frac{1}{n} = 0.$$

Also ist der Konvergenzradius ∞. ◯

Lösung. Der Konvergenzbereich für die Reihe $\sum_{n=1}^{\infty} \left(\frac{z}{n}\right)^n$ ist ganz $\mathbb{C}$. Für jedes feste z wird die Reihe, deren Summanden die Absolutbeträge sind, durch die Reihe $\sum_{n=1}^{\infty} \frac{|z|^n}{n!} = e^{|z|}$ majorisiert. ◯

(ii) $\sum_{n=3}^{\infty} \frac{z^n}{n(\log n)^n}$.

Lösung. Der Konvergenzbereich ist ganz $\mathbb{C}$. Die Folge der Wurzeln $\frac{1}{\sqrt[n]{n}\log n}$ konvergiert nämlich gegen 0 (siehe Lexikon-Hinweis!). ◯

(iii) $\sum_{n=0}^{\infty} (n+1)z^n$.

Lösung. Der Konvergenzradius ist $R = 1$ (siehe hierzu auch 11.9.viii). Das folgt auch wieder aus

$$\lim_{n\to\infty} \sqrt[n]{n+1} = \lim_{n\to\infty} \sqrt[n]{n} = 1.$$ ◯

Lösung. Für festes z benutzen wir das Quotientenkriterium, wie es in Aufgabe 12.4 dargestellt wird. Damit haben wir

$$\left|\frac{(n+1)z^n}{nz^{n-1}}\right| = \frac{n+1}{n}|z| \leq q$$

für ein $q < 1$ und genügend großes n genau dann, wenn z vom Absolutbetrag kleiner als 1 ist. ◯

(iv) $\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} z^n$.

Lösung. Der Konvergenzradius ist unendlich, denn $-\frac{-1}{\sqrt[n]{(2n+1)!}}$ ist eine Nullfolge. Die Reihe hängt mit dem Sinus zusammen. ◯

Lexikon: $\lim_{n\to\infty} \sqrt[n]{n} = 1$.

Aufgabe 12.3 (Ω). (5 Punkte)

Wir definieren $\Omega(1)$ als die Menge aller reellen Folgen $(a_n)_{n\in\mathbb{N}}$ für die es eine Konstante $C \in \mathbb{R}_{>0}$ gibt mit $|a_n| \geq C$ für alle n;

$$\Omega(1) = \left\{ a \in \mathbb{R}^{\mathbb{N}} \,\middle|\, \exists C > 0 \colon \forall n \in \mathbb{N} \colon\ |a_n| \geq C \right\}.$$

Damit bedeutet $a \in \Omega(b) := \Omega(1) \cdot b$, dass $\left(\frac{a_n}{b_n}\right)_{n\in\mathbb{N}} \in \Omega(1)$ liegt, also von der Null weg beschränkt ist. Prüfe (und begründe):

(i) $n^4 \in \mathcal{O}(n^5)$.

Lösung. Richtig. Der Quotient $n^4/n^5 = 1/n$ ist eine Nullfolge. ◯

(ii) $\frac{1}{42}n^3 - n \in \mathcal{O}(n^3)$.

Lösung. Richtig. Der Quotient $\frac{1}{42}(n^3 - n)/n^3$ ist durch $1/42$ nach oben beschränkt. ◯

(iii) $\frac{3^n}{n^2} - n^{17} \in \mathcal{O}(2^n)$.

Lösung. Falsch. Der Quotient

$$\frac{\frac{3^n}{n^2} - n^{17}}{2^n} = \left(\frac{3}{2}\right)^n \frac{1}{n^2} - \frac{n^{17}}{2^n}$$

ist unbeschränkt. ◯

(iv) $n \log n \operatorname{loglog} n \in \mathcal{O}(n^2)$.

Lösung. Richtig. Die Folge $\log n \log\log n / n$ ist eine Nullfolge. ◯

(v) $\frac{n}{\log n} \in \mathcal{O}(\sqrt{n})$.

Lösung. Falsch. Die Folge $\sqrt{n}/\log n$ divergiert. ◯

(vi) $n^4 \in \Omega(n^5)$.

Lösung. Falsch. Siehe (i). ◯

(vii) $\frac{1}{42}n^3 - n \in \Omega(n^3)$.

Lösung. Richtig, denn die Folge der Quotienten $1/42 - 1/n^2$ konvergiert monoton steigend gegen $1/42$. Also wird die Folge für *genügend große* n etwa die Zahl $1/100$ überschreiben. Dies geht als C aus der Definition von Ω durch, weil auch am Anfang, wo die Folge noch negativ

ist, keine betraglich kleineren Werte stehen, insbesondere keine 0. Diese hier notwendige Betrachtung der ersten Werte mit ihren möglichen Fallstricken zeigt aber eigentlich nur, dass unsere obige Definition noch nicht praxistauglich ist, weil uns letzlich gar nicht interessiert, was für kleine n passiert! Wir wollen ja zum Beispiel auch, dass $\frac{1}{25}n^3 - n \in \Omega(n^3)$ gilt, was aber mit der obigen Definition nicht der Fall ist, weil die Folge bei $n = 5$ einen Fehltritt macht. Besser ist daher folgende Definition:

$$\Omega(1) = \left\{ a \in \mathbb{R}^{\mathbb{N}} \;\middle|\; \exists C > 0 \colon \exists N \colon \forall n \in \mathbb{N}_{\geq N} \colon\ |a_n| \geq C \right\}.$$

Entsprechend sollte auch die Definition von $\mathcal{O}$ abgewandelt werden. [Was wir hier erleben, ist wie sich mathematische Definition nach und nach dem annähern, was wir von ihnen wollen.]

Dieser Teil zeigt, zusammen mit Teil (ii), dass $\mathcal{O}$ und Ω sich nicht ausschließen! ◯

(viii) $\frac{3^n}{n^2} - n^{17} \in \Omega(2^n)$.

Lösung. Richtig. Siehe Teil (iii). ◯

(ix) $n \log n \operatorname{loglog} n \in \Omega(n^2)$.

Lösung. Falsch. Siehe Teil (iv). ◯

(x) $\frac{n}{\log n} \in \Omega(\sqrt{n})$.

Lösung. Richtig. Siehe Teil (v). ◯

Aufgabe 12.4 (Quotientenkriterium). (4 Punkte)

Beweise das Quotientenkriterium in der folgenden „Vollversion“:

Satz (Quotientenkriterium). *Sei $\sum_{n=0}^{\infty} a_n$ eine Reihe mit $a_n \neq 0$ für alle $n \geq 0$. Es gebe eine reelle Zahl q mit $0 < q < 1$ so, dass*

$$\left| \frac{a_{n+1}}{a_n} \right| \leq q$$

für alle $n \geq N$. Dann konvergiert die Reihe $\sum_{n=0}^{\infty} a_n$ absolut (und damit auch gewöhnlich).

Lösung. Sei $b_n = |a_n|$. Nach Definition konvergiert die Reihe $\sum_{n=0}^{\infty} a_n$ genau dann absolut, wenn die Reihe $\sum_{n=0}^{\infty} b_n$ konvergiert. Angenommen, es gibt ein $0 < q < 1$ mit

$$\left| \frac{a_{n+1}}{a_n} \right| = \frac{b_{n+1}}{b_n} \leq q.$$

Induktiv zeigt man, dass $b_k \leq q^k b_0$ gilt. Wir haben also

$$\sum_{n=0}^{\infty} b_n \leq b_0 \sum_{n=0}^{\infty} q^n = b_0 \frac{1}{1-q},$$

woraus folgt dass die Folge der Teilsummen $\sum_{n=0}^{N} b_n$ beschränkt ist. Wegen $b_n > 0$ für $n > 0$ ist diese Folge monoton steigend, also konvergent. Daraus folgt wiederum, dass die Folge $\sum_{n=0}^{\infty} a_n$ absolut, und somit auch gewöhnlich, konvergiert. ◯

Lösung. Kombiniere Satz 9.13 über das Quotientenkriterium und Satz 9.14 über die absolute Konvergenz aus Brill (2001). ◯

***Aufgabe 12.5** (Crossover). (4 Punkte)

Es gibt Algorithmen zur Faktorisierung von natürlichen Zahlen mit ganz unterschiedlichen Laufzeiten. MuPad 3.0 hat die Funktionen `numlib::pollard`, `numlib::ecm` mit den Laufzeiten $\mathcal{O}^{\sim}(2^{n/4})$, $\mathcal{O}^{\sim}(2^{\sqrt{n}})$. Wir haben noch die Prozedur `triv_ifactor` beigesteuert, die mit der trivialen Methode einen Faktor findet. Ihre Laufzeit beträgt $\mathcal{O}(2^{n/2})$. [MuPADs Funktion `ifactor` verwendet eine Kombination verschiedener Techniken, ua. eine Tabelle der Primzahlen bis 300 000 und `numlib::ecm`. Sie ist damit asymptotisch gleich schnell wie `numlib::ecm`.] Wir haben Messung vorgenommen auf einem 700MHz-Pentium-III-Rechner, unter anderem mit einer 48-Bitzahl, die ein Produkt zweier 24-Bitzahlen war und folgende Laufzeiten gemessen:

Algorithmus Schranke	`triv_ifactor` $\mathcal{O}(2^{n/2})$	`numlib::pollard` $\mathcal{O}^{\sim}(2^{n/4})$	`numlib::ecm` $\mathcal{O}^{\sim}(2^{\sqrt{n}})$	`ifactor` $\mathcal{O}^{\sim}(2^{\sqrt{n}})$
Messung 40-Bit	4,226 sec	0,050 sec	0,231 sec	0,040 sec
Messung 48-Bit	46,717 sec	0,100 sec	0,150 sec	0,050 sec

Nimm an, dass die Schranken „exakt“ sind. [Also etwa, dass die Laufzeit des trivialen Algorithmus *gleich* $c \cdot 2^{n/2}$ ist mit einer gewissen Konstante c.] Untersuche die drei Algorithmen `triv_ifactor`, `numlib::pollard` und `numlib::ecm`.

(i*) Bestimme anhand der Messung mit 48-Bit-Eingabe jeweils die Konstante.

Lösung. Die Konstante c ergibt sich im ersten Fall durch $c = 46.717\,\text{sec}\,/2^{48/2}$, dabei wurde für n (Bitlänge) die Zahl 48 eingesetzt. Die anderen Konstanten folgen analog. Die Rechnung ergibt in den drei Fällen:

$$\begin{aligned} c_1 &= 0.000002784550190\,\text{sec}, \\ c_2 &= 0.00002441406250\,\text{sec}, \\ c_3 &= 0.001231669737\,\text{sec}\,. \end{aligned}$$

Dabei bezeichne c_i die Konstante für die i-te Spalte in der Tabelle. ◯

(ii*) Bestimme die Bitlänge n der Zahlen, die mit demselben Rechner innerhalb eines Tages faktorisierbar wären.

Lösung. Ein Tag besteht aus $24 \cdot 60 \cdot 60 = 86400$ Sekunden. Daraus ergibt sich, z.B. für den trivialen Algorithmus, $86400 = 1949 \cdot 2^{n/2}$, woraus folgt

$$n = 2 \log \left(\frac{86400 \sec}{c_1} \right) = 69,7.$$

Für die anderen Algorithmen folgt ähnlich: $n = 126,9$ (Pollard) und $n = 679$ (ecm). ◯

(iii*) Bestimme die Bitlänge n der Zahlen, die mit 1000 10-GHz-Rechnern innerhalb eines Tages faktorisierbar wären.

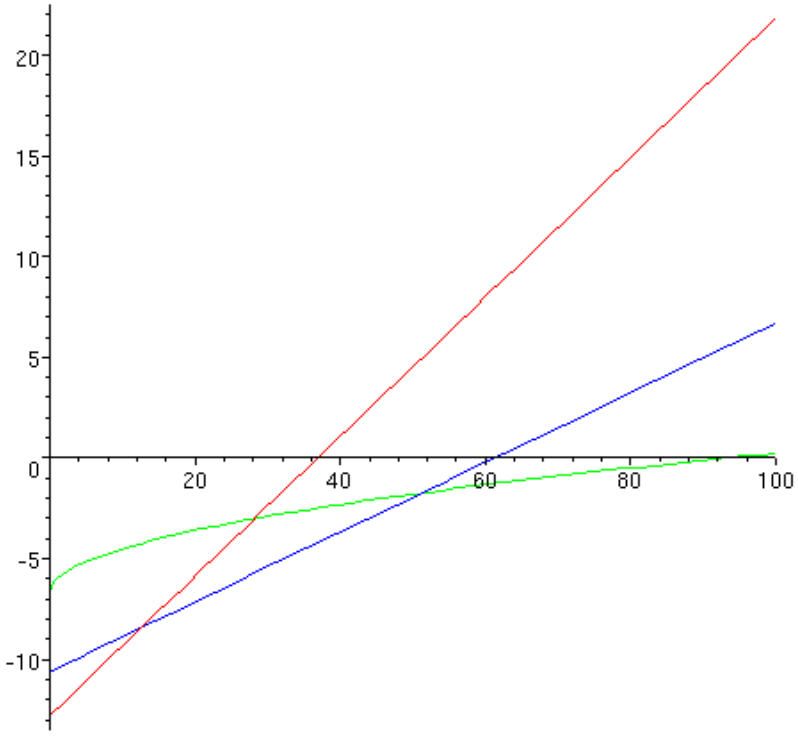
Lösung. Die angegebene Zahl Rechner ist um einen Faktor $1000 \cdot \frac{10\,\mathrm{GHz}}{700\,\mathrm{MHz}} \approx 14285$ schneller als der eine auf dem gemessen wurde. (Jedenfalls, wenn keine Kommunikation die Rechnung bremst.) Ein Tag auf diesen Maschinen entspricht also $\frac{10000}{7}$ Tagen auf dem urpsünglichen Rechner. Beim trivialen Algorithmus erhalten wir

$$n = 2 \log_2 \left(\frac{86400 \cdot \frac{10000}{7}}{c_1} \right) = 97.31030704.$$

Für die anderen Algorithmen gilt: $n = 182.0918294$ (Pollard) bzw. $n = 1589.313749$ (ecm). ◯

(iv*) Für welche Bitlängen ist welcher Algorithmus der schnellste? [Bestimme hierzu die Crossoverpunkte, dh. die Bitlängen, wo die Algorithmen einander in der Geschwindigkeit ablösen.]

Lösung. Aus $c_1 2^{n/2} = c_2 2^{n/4}$ erhalten wir $n = 12.52878471$. Der triviale Algorithmus wird also bereits bei 12.5 Bit durch den Algorithmus von Pollard abgelöst. (Wenn alle Näherungen gut genug sind...) Aus $c_1 2^{n/2} = c_3 2^{\sqrt{n}}$ erhalten wir $n = 28.19833105$. Der triviale Algorithmus würde bei 28 Bit vom ECM-Algorithmus abgelöst, aber da ist ja Pollard schon besser. Und aus $c_2 2^{n/4} = c_3 2^{\sqrt{n}}$ erhalten wir $n = 51.26759408$. Also wird der Algorithmus von Pollard bei 51 Bit durch den ECM-Algorithmus abgelöst.



Hier ist die Laufzeit logarithmisch ($\log_2 \frac{\text{Laufzeit}}{\text{sec}}$) gegen die Bitzahl aufgetragen, rot ist der triviale Algorithmus, blau ist Pollard und grün der ECM-Algorithmus.

Mit all diesen Angaben muss man aber vorsichtig umgehen, denn wir haben ziemlich stark genähert! Etwa für den ECM-Algorithmus sollte man korrekterweise $c_4 \exp(\sqrt{(2+o(1))\ln 2 \cdot n \ln n})\mathsf{M}(n)$ verwenden... Mit dieser und weiteren Anpassungen ergibt sich ein Kreuzungspunkt zwischen Pollard und ECM-Algorithmus erst bei 123 Bit. ◯

```
triv_ifactor:=proc(n)
begin
  if n mod 2=0 then return(2); end_if;
  for k from 3 to floor(sqrt(n)) step 2 do
    if n mod k=0 then return(k); end_if;
  end_for;
  return(n);
end_proc:
```