Tabellenspiele

Ein Programm-Algorithmus zur Berechnung von Primzahlen

Dieser Artikel baut auf meinem ersten Beitrag („Primzahlenverteilung ohne Geheimnisse“) auf und überschneidet sich teilweise mit ihm, damit er auch unabhängig davon gelesen werden kann. Für das Verständnis dieses Beitrags ist jedoch Kapitel 10 (Transformation auf den Zahlenstrahl) des ersten Beitrags hilfreich. Ging es im ersten Beitrag um geometrische Besonderheiten in der „Dreißiger-Primzahlmatrix“, so beschäftigt sich dieser Artikel mit einem darauf basierenden, kleinen Software-Algorithmus zur Berechnung von Primzahlen. Auf Grund der Fülle der Veröffentlichungen zu diesem Themenkreis ist es nahezu unmöglich, den Überblick zu behalten. Die in diesem Beitrag aufgelisteten Erkenntnisse entstammen ausschließlich eigenen Überlegungen. Eventuelle Übereinstimmungen mit anderen Publikationen sind daher absolut zufällig und unbeabsichtigt.

Durch den Einsatz moderner Computer mit einfachen Programmiersprachen ist die (Hobby-) Erforschung von Primzahlen relativ einfach geworden. Seit einigen Jahren wurde Visual Basic offiziell von Visual.net („Visual dot net“) abgelöst. Für die meisten privaten Anwendungen halte ich die Nutzung des alten Visual Basic jedoch immer noch als die einfachste und schnellste Methode.

Zur Erinnerung

Es ist nichts Neues, dass die natürlichen Zahlen von Eins aufwärts, schreibt man sie in eine Tabelle (Matrix) mit einer willkürlich gewählten Breite, bestimmte Muster ergeben, wenn man innerhalb dieser Zahlen die Primzahlen markiert. Dass dies sogar funktioniert, wenn man die Zahlen spiralenförmig anordnet, zeigt die sogenannte „Ulam-Spirale“ (siehe z.B. bei Wikipedia).

Bild 1 zeigt eine Tabelle, in der die (durch Kästchen symbolisierten) Zahlen in der ersten Zeile von 1 bis 30 aufgelistet sind. Die zweite Zeile repräsentiert die Zahlen 31 bis 60 usw. Es fällt auf, dass alle (rot markierten) Primzahlen nur in ganz bestimmten Spalten auftauchen: In allen Spalten unter den (Prim-) Zahlen 1, 7, 11, 13, 17, 19, 23, 29.

Weiterhin fällt auf, dass (leider) nicht alle Zahlen dieser Reihen Primzahlen sind: Es gibt Ausnahmen, die durch Lücken in den betreffenden Spalten erkennbar sind. Diese Lücken erscheinen auf den ersten Blick unregelmäßig angeordnet. Bei diesen Lücken handelt es sich stets um Primfaktoren, also um Produkte aus Primzahlen, was sich durch weitere Berechnungen sehr leicht herleiten lässt. Die erste Lücke erscheint bei der Zahl 49, was dem Produkt „7 mal 7“ entspricht. Die nächste Primzahl-Lücke taucht bei der Zahl 77 auf, dem Produkt aus 7 und 11. Auch bei diesen beiden Zahlen handelt es sich um Primzahlen.

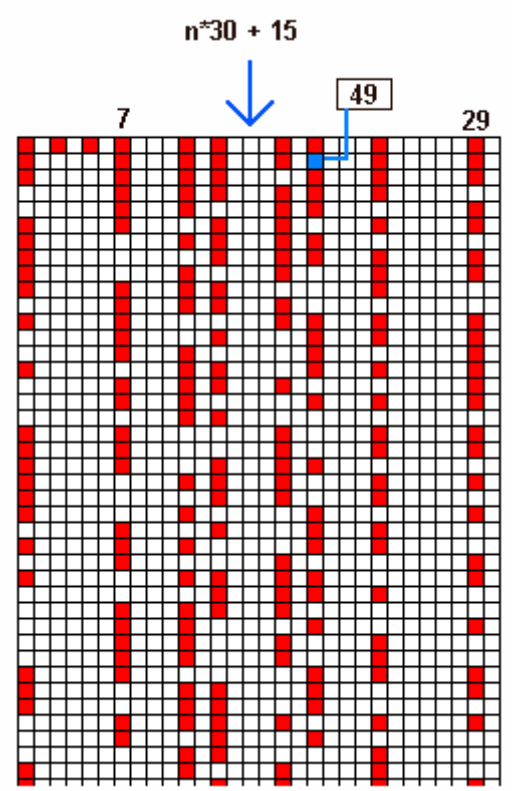


Bild 1: „Dreißiger-Matrix“ bzw. „Dreißiger-Tabelle“

Ganz nebenbei:

Ein weiterer, interessanter Aspekt, der an dieser Stelle jedoch nicht weiter verfolgt wird, besteht in der Tatsache, dass die Spalten mit den rot markierten Primzahlen symmetrisch zur Zahl 15 sind. Die im Bild gezeigten Reihen treten auch bei vielen anderen Tabellenbreiten auf, doch eine Spiegelsymmetrie zur Tabellenmitte, die dazu noch über die unten erwähnten Eigenschaften verfügt, ergibt sich (soweit ich bisher feststellen konnte) nur bei der Breite 30 und natürlich deren Vielfachen:

Die erste Primzahl, die nach 15 erscheint, ist die 17. Man darf also schreiben:
$17 = 15 + 2^1$ (2 „hoch“ 1)

Auch die folgenden Primzahlen lassen sich, im Zusammenhang mit der Zahl 15, durch Zweierpotenzen darstellen:

$19 = 15 + 2^2$
$23 = 15 + 2^3$
$31 = 15 + 2^4$

Doch halt: Was ist mit der Zahl 29? Die wäre doch nach 23 an der Reihe gewesen? Da die oben gezeigte Folge von Zweierpotenzen auch für Zahlen kleiner 15 gilt und sich obigen Aussagen auch auf alle Zahlen n*30 ±15 (für n = 0, 1, 2, 3 usw.) beziehen, wird die Zahl 29 erfasst, wenn man von der Mitte der zweiten Zeile ausgeht (1*30 + 15 =45) und von der Zahl 45 den Wert 2^4 subtrahiert.

Die Spaltenpositionen der Dreißiger-Tabelle lassen sich also durch den Ausdruck:

$n*30 + 15 \pm 2^x$ (für x = 1 bis 4)

darstellen.

Auch bei der Zahl 6 ergeben sich interessante Zusammenhänge in Bezug auf Primzahlen:
Beginnt man mit 1 und addiert immer wieder die Zahl 6 hinzu, so ergeben sich die Zahlen 7, 13, 19, ***25***, 31, 37 usw., wobei die Zahl 25 und alle weiteren sich ergebenden Zahlen mit der Endung 5 die

einzigen Zahlen dieser Reihe sind, bei denen es sich nicht um Primzahlen handelt. Um die restlichen Primzahlen 11, 17, 23, 29 usw. zu erhalten, braucht man eine weitere Zahl, zu der immer wieder der Wert 6 hinzu addiert wird: die Zahl 5.

Die obigen Aussagen lassen sich durch eine Tabelle der Breite 6 grafisch untermauern (Bild 2). Man erhält zwei Spalten, auf denen sich Primzahlen befinden können: Die Spalte an der Position 1 und die Spalte an der Position 5. Die weißen Lücken entstehen in diesem Falle nicht nur durch Primfaktoren, sondern zusätzlich durch Zahlen mit einer 5 am Ende

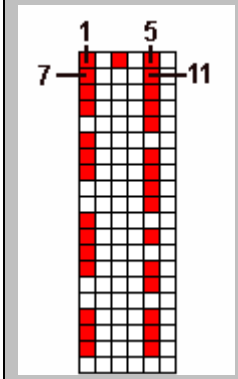


Bild 2

Vielleicht gelingt es dem einen oder anderen Leser, anhand dieser „Sechser-Tabelle“, die ich an dieser Stelle nicht weiter verfolgen möchte, weitere Gesetzmäßigkeiten zu entdecken.

Zurück zur Dreißiger-Tabelle: Aus den obigen Aussagen lassen sich folgende Schlussfolgerungen ziehen:

1) Wenn eine Zahl eine Primzahl ist, dann liegt sie zwingend auf den Spaltenpositionen
1, 7, 11, 13, 17, 19, 23, 29.
2) Nicht alle Zahlen auf diesen Spalten sind Primzahlen.
3) Bei den Lücken handelt es sich um Produkte aus Primzahlen (Primfaktoren)

Träfe Punkt 2 nicht zu, wäre dies zu schön, um wahr zu sein: Dann hätten wir nämlich ein einfaches Bildungsgesetz zur Herleitung der Primzahlen:

$n*30 + 15 \pm 2^x$ (für x = 1 bis 4 und n = 1 bis Unendlich für Primzahlen ab 29 aufwärts)

Solch ein Gesetz wurde jedoch bis heute nicht gefunden: Auf den Spalten gibt es Lücken, die auf den ersten Blick völlig unregelmäßig erscheinen. In Wahrheit werden diese Lücken jedoch durch sogenannte Primfaktoren erzeugt, also durch Produkte aus Primzahlen. In meinem ersten Beitrag in diesem Blog habe ich gezeigt, dass es sich bei diesen Lücken um eine Überlagerung verschiedener, periodischer Lückenreihen handelt: Dem Produkt aus Primzahlen und der Zahl 30. Auf diese Tatsache wollen wir hier jedoch nicht näher eingehen.

Betrachten wir stattdessen die Konsequenz aus Punkt 1:

Ab der Zahl 30 wiederholen sich alle (potenziellen) Primzahlen: 30 + 1, 30 + 7, 30 + 11 und so weiter. Wenn uns also jemand auffordert, ganz schnell alle Primzahlen von 1 bis 100 aufzuzählen, so brauchen wir nur die Primzahlen unter 30 auswendig zu lernen. Alle weiteren Primzahlen lassen sich durch einfache Addition herleiten. Doch halt: Was ist mit den Ausnahmen, den Primfaktoren?
Wenn wir die Zahlen über 30 betrachten, so gibt es zwischen 0 und 100 nur ganze drei davon:
7 * 7 = 49
7 * 11 = 77
7 * 13 = 91

Wenn wir uns diese 3 Zahlen zu den Primzahlen unter 30 noch zusätzlich merken, können wir so manch einen durch unsere Künste im „Schnellrechnen“ verblüffen, indem wir ihm rasend schnell alle Primzahlen von 1 bis 100 nennen.

Methode zur Berechnung aller Primzahlen bis zu einer Obergrenze n

Aus obigen Überlegungen lassen sich tatsächlich alle Primzahlen bis zu einer durch den Computer vorgegebenen Grenze berechnen. Diese Berechnung erfolgt nicht auf Grund einer Formel – die gibt es, wie schon gesagt, leider nicht. Die folgende Berechnung geschieht durch das Wegstreichen von Zahlen aus einer vorgegebenen Reihe von „Primzahlkandidaten".

Die Primzahlkandidaten erhält man, wie in Bild 1 gezeigt, anhand der Spalten in der Dreißiger-Tabelle (Unter „Primzahlkandidaten" verstehe ich alle Zahlen in den durch Primzahlen markierten Spalten der der Dreißiger-Tabelle):

Zu den Vielfachen von 30 werden alle Primzahlen unter 30 ganz einfach hinzuaddiert:

30 + 1
30 +7
30 + 11 … usw.
……
30 + 29

60 + 1
60 +7
60 + 11 … usw.
……
60 + 29

90 + 1
90 +7
90 + 11 … usw.
……
90 + 29

Der entsprechende VB-Programmcode könnte wie unten gezeigt aussehen. Die Zahl 10000 ist nur ein Beispiel und wird nur durch die Leistungsfähigkeit des betreffenden Computers bestimmt. Das Array mit dem Namen „Array" (es kann natürlich auch einen anderen Namen besitzen) kann vom Typ „Integer" sein und ist eindimensional (dies entspräche einer Tabelle mit nur einer Spalte). Die Zahl 10000 in der Klammer entspricht dem um 1 erhöhten Wert der maximalen Schleifenvariablen:

```
Dim Array(10000) as Integer
```

```
    For n = 1 to 9999

    Array( n*30 + 1) = 1
    Array(n*30 + 7) = 1
    Array(n*30 + 11) = 1
    Array(n*30 + 13) = 1
    Array(n*30 + 17) = 1
    Array(n*30 + 19) = 1
    Array(n*30 + 23) = 1
    Array(n*30 + 29) = 1

    Next n
```

Im Array werden nicht die Primzahlen selbst eingetragen, sondern es wird an den Adressen, die den gefundenen Primzahlkandidaten entsprechen, einfach der Wert 1 gespeichert, was dem Ankreuzen einer Zahl in einer Liste entspricht. Eine Primzahl wird dann ganz einfach nur durch ihre Speicheradresse im Array definiert. Das spart Speicherplatz und Rechengeschwindigkeit.

Mit dem obigen Programm erhalten wir eine Liste, in der alle potenziellen Primzahlen von 1 bis 9999 * 30 + 29 durch das Markieren einer 1 „angekreuzt" sind. Folgende Tabelle zeigt die ersten 30 Adressen des Arrays und deren Inhalte:

Array					
Adresse	**Inhalt**	**Adresse**	**Inhalt**	**Adresse**	**Inhalt**
1	1	11	1	21	0
2	0	12	0	22	0
3	0	13	1	23	1
4	0	14	0	24	0
5	0	15	0	25	0
6	0	16	0	26	0
7	1	17	1	27	0
8	0	18	0	28	0
9	0	19	1	29	1
10	0	20	0	30	0

Die Nullen werden bei der Arraydeklaration automatisch eingesetzt und müssen nicht eingetragen werden. So weit, so gut. Nun wissen wir jedoch, dass nicht alle Markierungen in dieser Tabelle richtig sind. Es gilt, die falschen Markierungen heraus zu finden und die entsprechenden Einsen in der Tabelle wieder zu entfernen. Nichts einfacher als das: Mit einem weiteren Programm berechnen wir anhand der Primzahlen unter 30 alle sich daraus ergebenden Primfaktoren. Um die Rechenweise zu verstehen, ist folgender Gedankengang, der auch im ersten Beitrag dieses Blogs ausführlich beschrieben wurde, wichtig:

Stellen wir uns vor, die Primzahlkandidaten aus obiger Tabelle seien auf einer unendlich langen Messlatte (zum Beispiel auf einem Zollstock) markiert. Nun kopieren wir die Primzahlen von 1 bis 29 auf eine zweite Messlatte und dehnen diese 30 cm lange Messlatte um den Faktor 7. Wir erhalten dadurch eine Reihe von Zahlen, die mit 1 beginnen und mit 210 (7*30) enden. Auf dieser Tabelle sind mit dem Faktor 7 multiplizierten Primzahlen von 7 bis 29 enthalten: 49, 77, 91, 119 und so weiter. Diese Zahlen sind auch in unserer Tabelle mit den Primzahlkandidaten markiert und können dort nun gestrichen werden.

Doch bei der Zahl 210 ist die Arbeit mit der um den Faktor 7 gedehnten „Messlatte“ noch nicht zu Ende: Nun legen wir die 210 Zentimeter lange „Siebener-Latte“ an der Zahl 210 an und können das gleiche Spiel für alle Zahlen über 210 wiederholen: 210 + 49, 210 + 77, 210 + 91 und so weiter. Auch diese Zahlen sind keine Primzahlen, obwohl sie sich, wenn wir einen Blick zurück auf Bild 1 werfen, in den Spalten unter den Primzahlen von 1 bis 29 befinden. Rein rechnerisch lassen sich diese Zahlen durch folgende Programmzeilen ermitteln. An den Adressen der gefundenen Werte werden die Einsen aus dem Array (die eine Primzahl signalisieren) wieder gelöscht:

```
For n = 0 To 95
Array(n * 210 + 7 * 7) = 0
Array(n * 210 + 11 * 7) = 0
Array (n * 210 + 13 * 7) = 0
Array (n * 210 + 17 * 7) = 0
Array (n * 210 + 19 * 7) = 0
Array (n * 210 + 23 * 7) = 0
Array (n * 210 + 29 * 7) = 0
Array (n * 210 + 31 * 7) = 0
Next n
```

Mit 95 als Maximum der Schleifenvariable n werden Zahlen bis 95 mal 210 erfasst. Doch mit der 7 allein kommen wir nicht weit. Das Gleiche muss mit den Zahlen 11, 13, 17, 19, 23 und 29 geschehen. Die entsprechenden Programmzeilen sehen dabei wie folgt aus:

```
For n = 0 To 60
Array(n * 330 + 7 * 11) = 0
Array(n * 330 + 11 * 11) = 0
Array (n * 330 + 13 * 11) = 0
Array (n * 330 + 17 * 11) = 0
Array (n * 330 + 19 * 11) = 0
Array (n * 330 + 23 * 11) = 0
Array (n * 330 + 29 * 11) = 0
```

```
Array (n * 330 + 31 * 11) = 0
Next n


For n = 0 To 51
Array(n * 390 + 7 * 13) = 0
Array(n * 390 + 11 * 13) = 0
Array (n * 390 + 13 * 13) = 0
Array (n * 390 + 17 * 13) = 0
Array (n * 390 + 19 * 13) = 0
Array (n * 390 + 23 * 13) = 0
Array (n * 390 + 29 * 13) = 0
Array (n * 390 + 31 * 13) = 0
Next n
```

… und so weiter…

Der rechte Ausdruck in der Klammer nimmt nacheinander alle Primzahlwerte von 7 bis 29 an. Die Schleifenvariable n wird in jedem Neudurchlauf mit 30 * x multipliziert, wobei x bei jeder neuen Schleife nacheinander die Werte 7, 11, 13, 17, 19, 23, 29 durchläuft (210, 330, 390…). Die Anzahl der Schleifendurchläufe verringert sich von Mal zu Mal, damit das Produkt aus n* 210 bzw. n* 330 usw. ungefähr immer den gleichen Wert annimmt.

Mein mit VB geschriebenes Programm besitzt eine Funktion, die die in der Tabelle (Array) enthaltenen Einsen und Nullen grafisch interpretiert und als farbliche Markierung in ein Gitter einträgt. Es zeigt daher, wenn ich diese Funktion anwende, alle Primzahlen, wie bereits in Bild 1 dargestellt, an und markiert zusätzlich die durch obige Rechenalgorithmen gefundenen Primzahlen mit blauer Farbe (Bild 3). Die Matrixbreite hat diesmal einen Wert von 90, damit mehr Zahlen auf dem Bildschirm angezeigt werden können.

Wie das Bild zeigt, kann man auf diese Weise sehr schnell alle Primzahlen bis zur Zahl 1679 aus den 8 Primzahlen unter der Zahl 30 herleiten. Die Zahl 1681 (eingekreist) ist die erste Nicht-Primzahl, die der obige Algorithmus aus der Reihe der Primzahlkandidaten nicht identifizieren und „rauswerfen“ konnte.

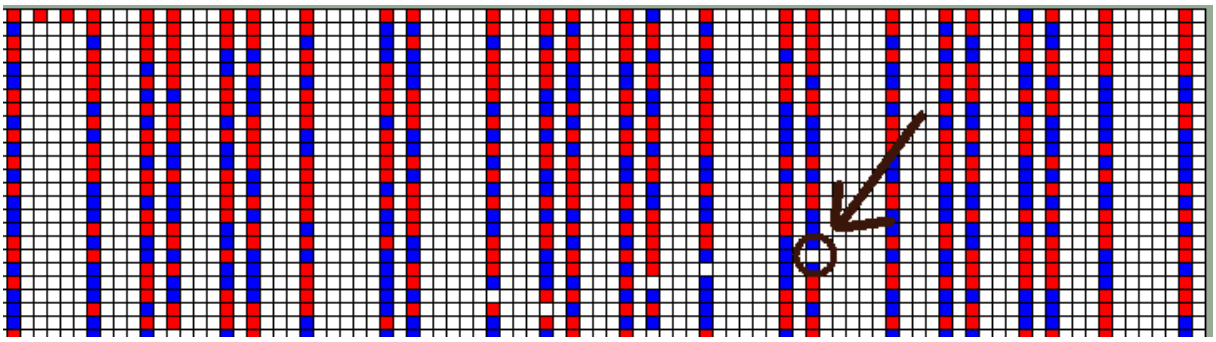


Bild 3

Programm zur Berechnung aller Primzahlen bis 10199

Weitere Experimente mit dem Programmcode haben gezeigt, dass auch bei Erhöhung der Schleifenvariablen keine weiteren „Nichtprimzahlen“ mehr aus der Reihe der Kandidaten eliminiert werden konnten. Folgendes Programm kann jedoch mehr. Die Zeilen zur Deklaration das Haupt-Arrays und zur Berechnung der Primzahlkandidaten wurden weggelassen.

Erhöht man in der allgemein geschriebenen Zeile

Array (n * P + 31 * P) = 0

den Primzahlwert P auf über 30 bis hin zur Zahl 97, so lassen sich bereits mit der Schleifenzahl n = 50 alle Nicht-Primzahlen bis zum einem Wert knapp unter 10201 aus der Reihe der Primzahlkandidaten streichen. Damit auch die Primzahlen bis 97 mittels einer Schleife aufgerufen werden können, wird ein zweites Array (p) deklariert, in welchem alle Primzahlen von 1 bis 97 (mittels lästiger Tipparbeit) eingetragen sind:

Dim p(50) As Integer

p(1) = 1; p(2) = 7; p(3) = 11; p(4) = 13; p(5) = 17; p(6) = 19; p(7) = 23; p(8) = 29; p(9) = 31; p(10) = 37; p(11) = 41; p(12) = 43; p(13) = 47; p(14) = 47; p(15) = 53; p(16) = 59; p(17) = 61; p(18) = 67; p(19) = 71; p(20) = 73; p(21) = 73; p(22) = 79; p(23) = 83; p(24) = 89; p(25) = 97

Nun lassen sich alle oben gezeigten Schleifen zu einer einzigen, verschachtelten Schleife zusammenfassen:

```
For m = 2 To 25
For n = 0 To 50
        Array (n * p(m) * 30 + 7 * p(m)) = 0
        Array (n * p(m) * 30 + 11 * p(m)) = 0
        Array (n * p(m) * 30 + 13 * p(m)) = 0
        Array (n * p(m) * 30 + 17 * p(m)) = 0
        Array (n * p(m) * 30 + 19 * p(m)) = 0
        Array (n * p(m) * 30 + 23 * p(m)) = 0
        Array (n * p(m) * 30 + 29 * p(m)) = 0
        Array (n * p(m) * 30 + 31 * p(m)) = 0
Next n
Next m
```

Die äußere Schleife mit der Variablen p sorgt dafür, dass die Primzahlen bis 97 an die Stelle „p(m)“ eingesetzt werden. Mit etwas Grübelarbeit dürfte es möglich sein, den aus acht Zeilen bestehenden, eingerückten Code mittels einer weiteren Schleifenverschachtelung auf eine einzige Zeile zu reduzieren. In den obigen Listings handelte es sich aus Gründen der Vereinfachung nur um ein Array mit einer Spalte. Um, wie in Bild 3 und 4 gezeigt, weiße und blaue Kästchen darzustellen, muss das Array über zwei Spalten verfügen. Wie auch immer: Das Ergebnis (Bild 4) sieht wie folgt aus:

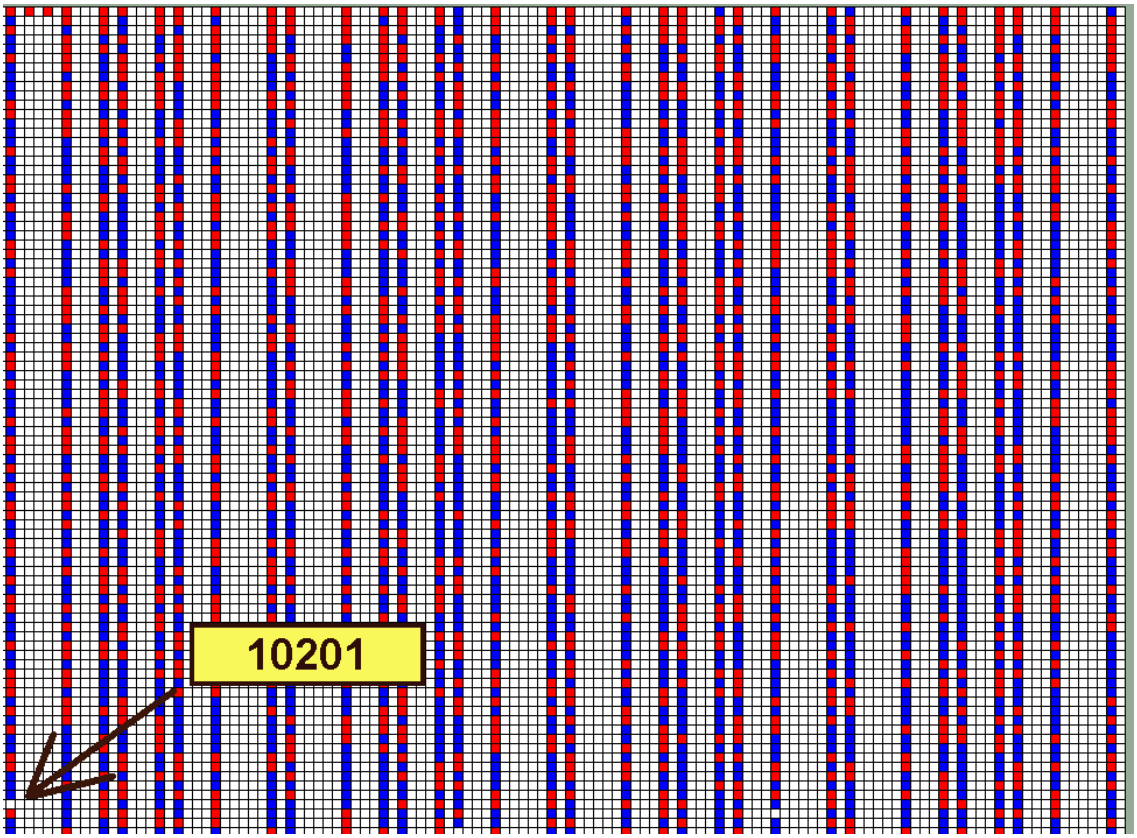


Bild 4

Die blauen Kästchen markieren alle mit obigem Code gefundenen Nicht-Primzahlen. Sie befinden sich, das wurde überprüft, in obigem Bild genau an der Stelle, an denen sich vorher (in den entsprechenden Spalten) weiße Kästchen befanden. Es wurden keine roten Kästchen blau überschrieben und es blieben bis zum Wert 10201 keine weißen Kästchen frei. Das bedeutet, dass mit der oben gezeigten Rechenmethode aus allen Primzahlen von 1 bis 97 alle Primzahlen von 1 bis 10199 berechnet werden können. Das entspricht ungefähr dem Verhältnis 1 zu 100. Um die Kästchen nicht bis zur Zahl 10201 durchzählen zu müssen, habe ich noch eine Funktion im Programm eingebaut, die mir die Nummer eines per Maus angeklickten Kästchens in einem Textfenster ausgibt.

Kennt man alle Primzahlen von 1 bis n, so lassen sich mit dem oben gezeigten Verfahren alle Primzahlen bis x * n berechnen. Kenn man jedoch alle Primzahlen von 1 bis x * n, so lassen sich daraus alle Primzahlen von 1 bis x * x * n berechnen und so weiter – im Prinzip also alle Primzahlen.
Dabei wäre noch zu ermitteln, ob es sich bei x um einen konstanten Faktor handelt, oder ob x nicht-linear wächst.
Noch ein wichtiger Hinweis: Die beim Programmstart im Array enthaltenen Primzahlen (Einsen) wurden natürlich schon vor langer Zeit ein einziges Mal auf konventionelle Weise berechnet und danach ein für alle Mal in einer Datei gespeichert. Bei Bedarf können die benötigten Primzahlen dann ganz einfach von der Festplatte in ein entsprechend konfiguriertes Array geladen werden.

Mit der Dreißiger-Methode können wir übrigens auch sofort ausschließen, dass es sich bei einer willkürlich gewählten Zahl um eine Primzahl handelt: Teile die Zahl durch 30! Wenn es sich beim Rest nicht um die Primzahlen unter 30, sondern um die Zahlen 3, 9, 21 oder 27 handelt, ist die Zahl garantiert keine Primzahl. Im anderen Falle könnte sie eine sein – muss aber nicht. Die mit 5 endenden Zahlen können wir aus der oben genannten Reihe ausklammern, da alle mit 5 endenden Zahlen natürlich immer durch 5 teilbar und daher keine Primzahlen sind.

Da wir gerade bei den Ausnahmen sind: Links oben in der Primzahltabelle tauchen immer zwei rote Kästchen auf, an die sich keine nach unten gehenden Spalten anschließen: Die Zahlen 3 und 5. Sie bilden eine Ausnahme, haben für alle Primzahlen über 30 keinerlei Bedeutung und werden daher in den vorliegenden Betrachtungen nicht weiter berücksichtigt.

Warum habe ich gerade eine Spaltenbreite von 30 gewählt? Nun – ich habe den Eindruck, dass die dort vorherrschende Symmetrie zur Zahl 15 etwas Besonderes darstellt und auf diese Spaltenbreite verweist. Mathematisch beweisen kann ich dies natürlich nicht – und auch nicht meine sich daraus ergebenden Ausführungen. Auch bin ich nicht in der Lage, die richtige Anzahl der erforderlichen Mindest-Schleifendurchläufe für das oben gezeigte Programm zu berechnen. Vielleicht eine Herausforderung für den einen oder anderen Mathematiker unter den Lesern – falls dieses Thema nicht schon (auch dies entzieht sich meiner Kenntnis) längst bekanntes Gedankengut ist.

In diesem Punkt bin ich jedoch optimistisch und denke, dass einige Fakten um die Primzahlen bis heute Vielen noch nicht ganz klar geworden sind. So ist zum Beispiel auf zahlreichen Internetseiten von „geheimnisvollen“ Linien und Mustern zu lesen, die sich ergeben, wenn man, wie oben gezeigt, Primzahlen in Tabellen mit verschiedener Spaltenbreite einträgt. Besonders deutlich zeigen sich solche Muster, wenn man oben gezeigte Kästchen zu Pixeln schrumpfen lässt und größere Spaltenbreiten verwendet. Die ach so geheimnisvollen Muster sind allein auf die Tatsache zurückzuführen, dass sich die Primzahlen, wie gezeigt, im Abstand von 30 und im Abstand von Vielfachen von 30 mit Einschränkungen wiederholen (obwohl man dies ja durchaus auch als ein „Geheimnis“ bezeichnen könnte).

Je nach „Zeilenumbruch“ (Tabellenbreite) ergeben sich einmal senkrechte und einmal schräge Linien, deren Zwischenräume (das sind die durch Primfaktoren bedingten Unregelmäßigkeiten) mit allerlei wirren, strukturlosen Zufallsmustern gefüllt sind. Das gilt auch für die Ulam-Spirale, mit der es, wie viele behaupten, ebenfalls eine geheimnisvolle Bewandtnis habe. Mit der 30er-Matrix ist das vermeintliche Geheimnis schnell gelüftet, denn die sich durch die Zahl 30 ergebenden Regelmäßigkeiten offenbaren sich meines Erachtens auch dann, wenn man Zahlen nicht tabellenartig, sondern spiralförmig anordnet.

Grafikspielereien

Im Folgenden ein paar Grafikbeispiele. Zunächst drei Muster, in denen an Stelle größerer Kästchen kleine Pixel verwendet wurden. Alle Nicht-Primzahlen sind schwarz dargestellt, alle Primzahlen weiß. Durch den gewonnenen Platz sind auch größere Tabellenbreiten bis über 500 möglich. Die folgende Bilderserie zeigt, dass sich bereits bei Breiten-Änderungen um den Wert 1 die Muster komplett ändern:

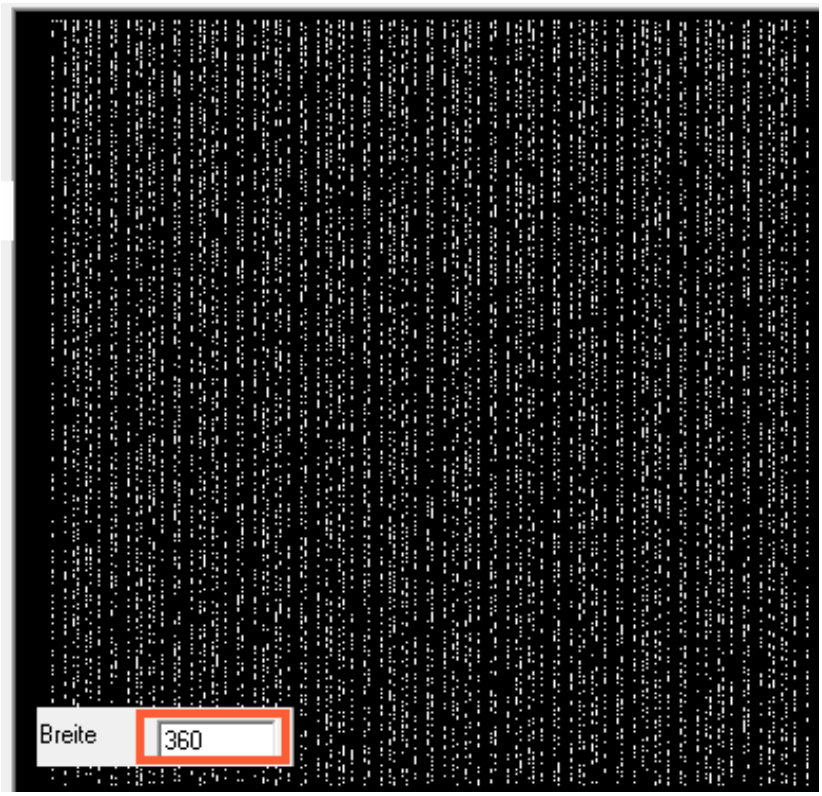


Bild 5: Bei der Breite 360 sind ganz deutlich vertikale Strukturen zu erkennen. Kein Wunder, da 360 ohne Rest durch 30 teilbar ist.

Bild 6: Bei Verringerung der Breite um 1 auf den Wert 359 werden aus senkrechten Linien plötzlich schräge Linien – ein Phänomen, das man auch an alten Schwarzweißfernsehgeräten bei falsch eingestellter Synchronisation feststellen konnte.

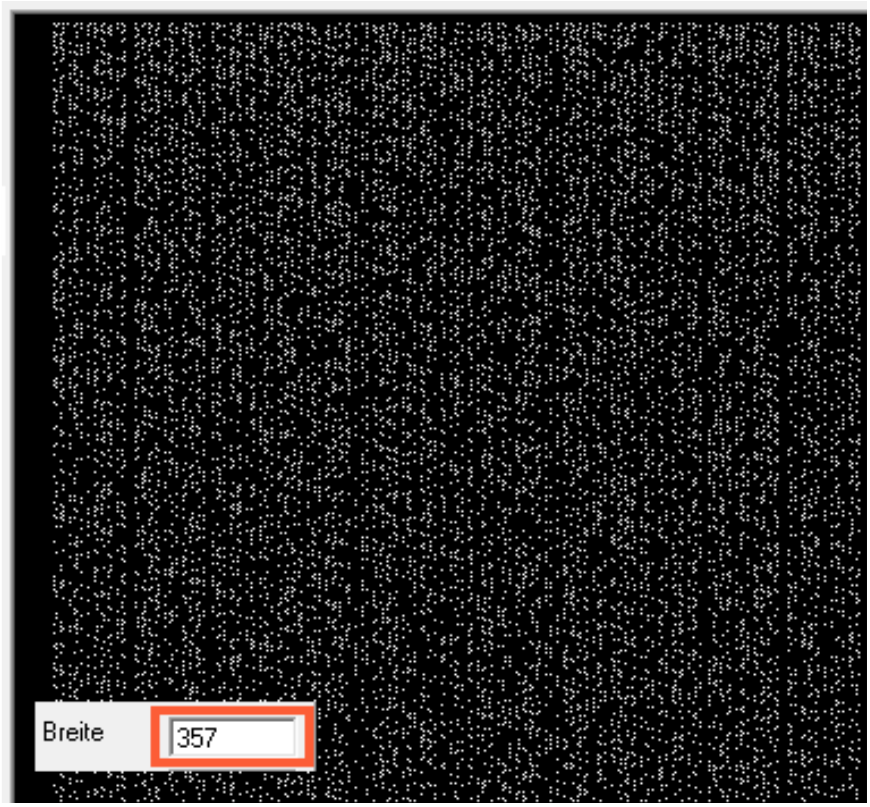
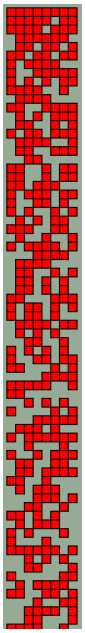
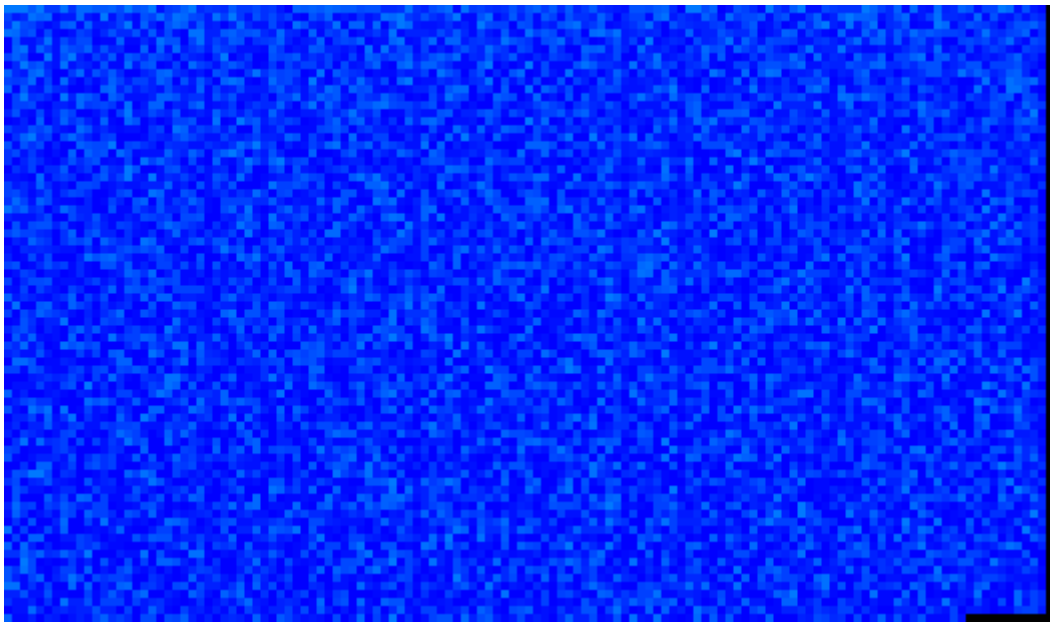


Bild 7: Bei einer Breite von 357 treten die (diesmal wieder senkrechten) Linien weniger deutlich hervor. Mit etwas Phantasie und zusammengekniffenen Augen glaubt man, wolkenartige Strukturen zu erkennen. Ganz gleich, welchen Wert man auch für den Zeilenumbruch (die Breite) wählt: Mehr als solch senkrechte oder schräge Linien mit unregelmäßigen Unterbrechungen sind nicht zu erkennen. Auf diese Weise scheint man dem Geheimnis der Primzahlen also nicht auf die Spur zu kommen.





Bilder 8a, 8b: Links: Stellt man die relevanten Zeilen der 30er-Tabelle ohne Zwischenräume dar, so erinnert dies an einen 8-Bit-Binärcode (**Bild 8a**, links). Verwandelt man diesen Binärcode in Helligkeitswerte (von 0 bis 255) und stellt diese Werte wiederum in einer Tabelle mit einer bestimmten Breite dar (**Bild 8b**, rechts), so ergeben sich wiederum Muster, die jedoch diesmal nicht aus schwarzen und weißen, sondern aus Pixeln unterschiedlicher Intensität bestehen. Ganz gleich, wie man die Breite jedoch wählt: Auch hier ist außer wolkenartigen Strukturen und senkrechten oder schrägen Linien nichts zu sehen. Die Farbe hat übrigens nichts mit dem Ergebnis zu tun: Das Bild wurde, damit es schöner aussieht, nachträglich blau eingefärbt. Da immer acht Primzahlen zu einem Helligkeitswert zusammengefasst wurden, benötigt man logischerweise acht Mal so viele Primzahlen wie bei den Bildern 5, 6 und 7.

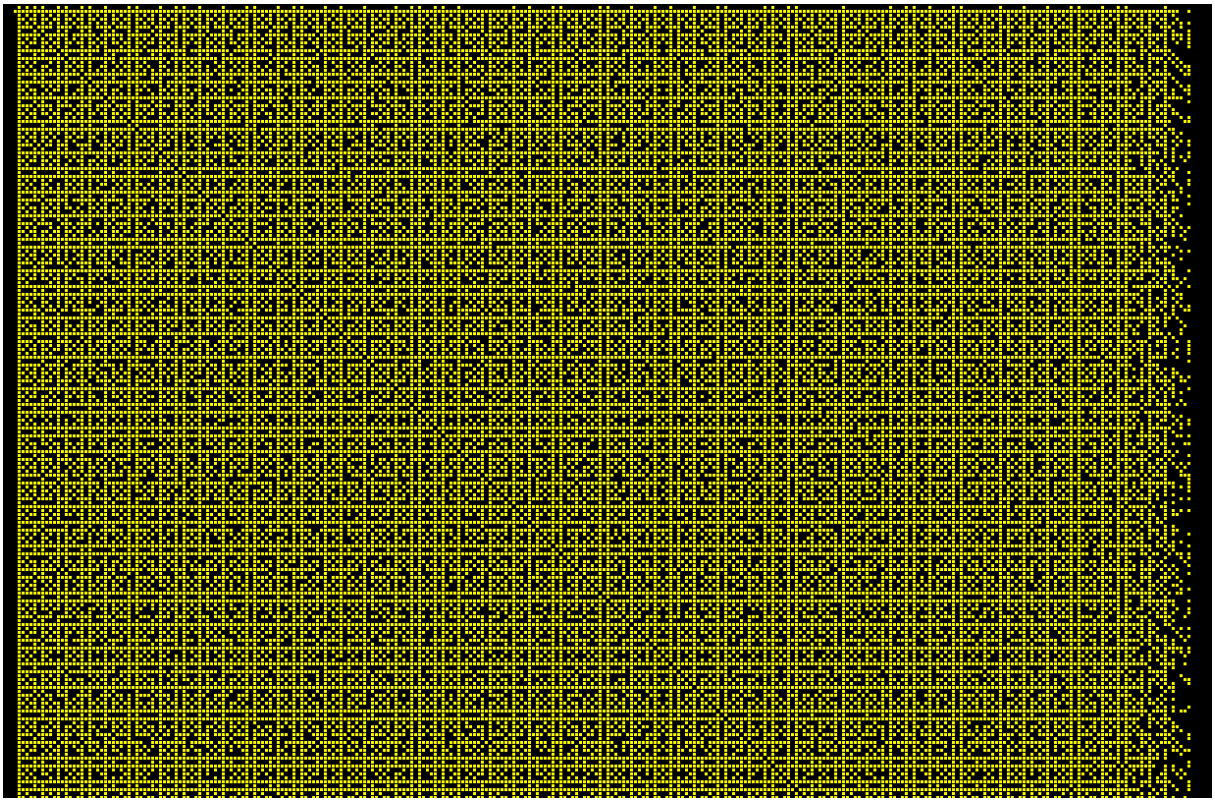


Bild 9 (siehe Bild 10)

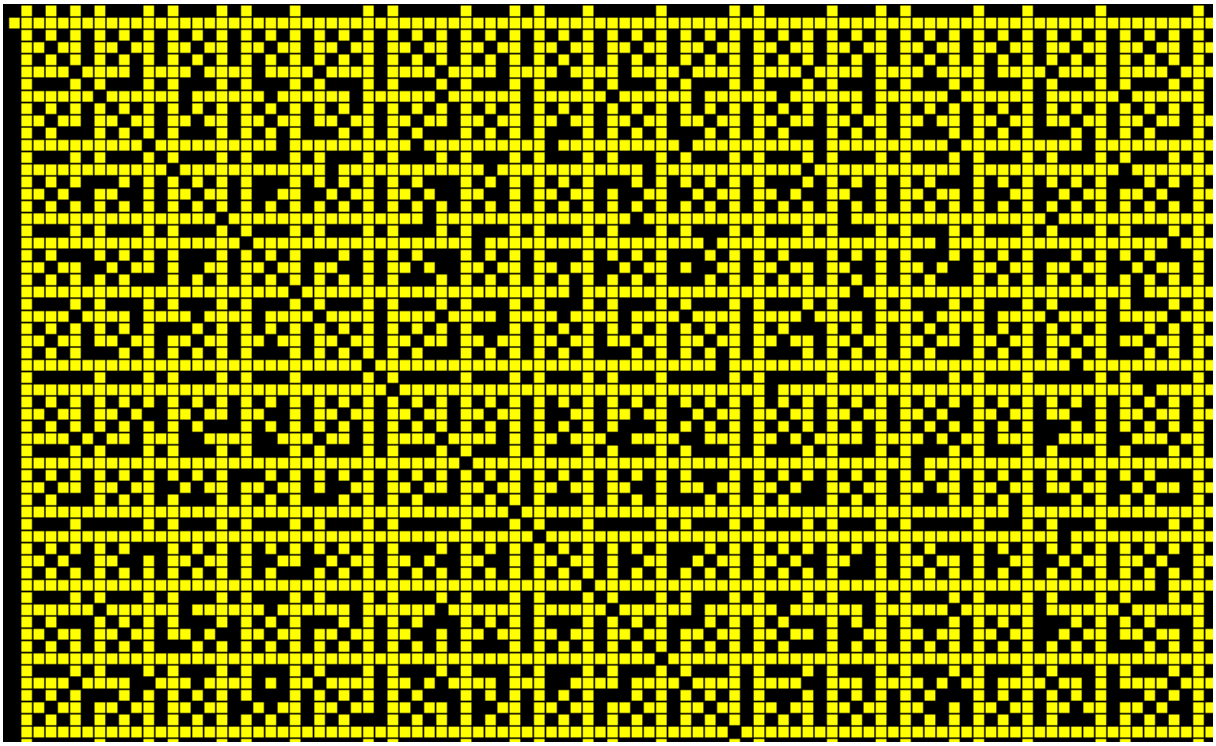


Bild 10: Vergrößerter Ausschnitt aus Bild 9

Bild 10 ist wie folgt zustande gekommen:
Innerhalb einer Schleife wurde jedes Mal eine neue Matrix berechnet, wobei sich die Breite jedes Mal um den Wert 1 erhöhte. Dabei wurde die alte Matrix nicht gelöscht: Die neue Matrix wurde über die alte geschrieben, wobei jedoch nur die (gelben) Primzahlwerte, nicht jedoch die (schwarzen) Nichtprimzahlen gezeichnet wurden. Somit entsteht sozusagen eine Art „Primzahl-Überlagerung“ die

zu seltsamen Kästchenmustern führt. Diese Kästchen sind jedoch einfach zu erklären, wenn man die einzelnen Zeilen betrachtet:

Zeile 1 (die oberste Zeile) beginnt immer wieder von vorn. Durch das Überschreiben ändert sich nichts, so dass hier alle Primzahlen von 1 bis zur Matrixbreite abgebildet sind.

Zeile 2 springt bei jedem Durchlauf um 1. Irgendwann wurde jede Primzahl an jeder Stelle abgebildet, so dass am Ende eine durchgezogene Linie entsteht.

Zeile 3 springt immer um den Wert 2, so dass schließlich nur jedes zweite Kästchen gelb werden kann.

Bedenkt man, dass jede weitere Zeile stets um einen um 1 höheren Wert mit sich selbst überschrieben wird, so lässt sich das gesamte Muster leicht erklären. Interessant ist die Symmetrie zur Diagonalen, die von 0 aus nach rechts unten verläuft.

Man kann sich Bild 10 übrigens auch als eine 3-D-Matrix vorstellen, wobei in Richtung der hinzugekommenen Z-Achse alle neu berechneten Tabellen, die sich in ihrer Breite jeweils um 1 unterscheiden, aufeinander gestapelt werden. Stellt man sich die berechneten Markierungen der einzelnen Tabellen als undurchsichtige Flächen oder Würfel vor, so ergibt sich das in Bild 10 gezeigte Bild, wenn man in Richtung Z-Achse durch den so entstandenen Würfel blickt.

Ob das gezeigte Muster jedoch typisch für Primzahlen ist oder auch bei willkürlichen Verteilungen entsteht, kann ich spontan nicht sagen, obwohl ich Letzteres vermute. Hier besteht noch Raum für entsprechende Experimente und Überlegungen, über die ich vielleicht in einem weiteren Artikel berichten werde.

Für Hinweise, Tipps, konstruktive Kritik und Anregungen bin ich unter der E-Mail-Adresse:

kurt.diedrich@tele2.de

erreichbar.