

Wegen $\pi = 4 \cdot \arctan(1)$ kann die Reihenentwicklung des Arkustangens verwendet werden.

$$\arctan(x) \approx \sum_{k=0}^n (-1)^k \cdot \frac{x^{2k+1}}{2k+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \dots + (-1)^n \cdot \frac{x^{2n+1}}{2n+1} ; |x| \leq 1$$

Wollte man jedoch Pi mittels $\arctan(1)$ auf 5 Dezimalen genau berechnen, so müsste man bereits 100000 Glieder der Reihe berechnen. Daher verwendet man Kombinationen mehrerer \arctan -Reihen.

Beispiele:

- $\pi = 16 \cdot \arctan(1/5) - 4 \cdot \arctan(1/239)$ Machin 1706
- $\pi = 48 \cdot \arctan(1/18) + 32 \cdot \arctan(1/57) - 20 \cdot \arctan(1/239)$ Gauß
- $\pi = 176 \cdot \arctan(1/57) + 28 \cdot \arctan(1/239) - 49 \cdot \arctan(1/682) + 96 \cdot \arctan(1/12943)$ Störmer 1896

Wir betrachten im folgenden die MACHIN-Reihe . Zu berechnen sind daher

$\arctan(1/5) = 1/5 - (1/5)^3/3 + (1/5)^5/5 - (1/5)^7/7 + \dots$ sowie
 $\arctan(1/239) = 1/239 - (1/239)^3/3 + (1/239)^5/5 - (1/239)^7/7 + \dots$

Ersichtlich ist hier keine Multiplikation erforderlich, sondern Addition, Subtraktion, Division .

Zunächst müssen wir die für jede \arctan -Reihe benötigte Zahl $2n+1$ bestimmen:

Sei nk die Anzahl der gewünschten korrekten Nachkommastellen (Dezimalen).

Verändert nun der n -te Summand der Reihe die nk -te Dezimale nicht mehr, so ist die gewünschte Genauigkeit erreicht. Dies ist dann der Fall, wenn der n -te Summand $x^{2n+1}/(2n+1)$ kleiner ist als 10^{-nk} .

Ansatz also: $x^{2n+1} / (2n+1) < 10^{-nk}$ mit $0 < x < 1 ; n \in \mathbb{IN}$

Umformungen: $\lg(x^{2n+1} / (2n+1)) < -nk$
 $\lg(x^{2n+1}) - \lg(2n+1) < -nk$
 $(2n+1) \cdot \lg(x) - \lg(2n+1) < -nk$

Lässt man $\lg(2n+1)$ weg, so bekommt man ein etwas größeres n (größere Sicherheit).

Daher erhalten wir $(2n+1) \cdot \lg(x) < -nk$ bzw. (da $\lg(x) < 0$) $2n+1 >= -nk / \lg(x)$ und schließlich mit Substitution $x = 1/z$ ($\lg(x) = -\lg(z)$)

$2n+1 \geq \frac{nk}{\lg(z)} ; 2n+1 \text{ ungerade}; z = \frac{1}{x}$

(z.B. $x = 1 / 239 \rightarrow z = 239$)

Proberechnung für $nk = 8$ und $z = 5 \rightarrow 2n+1 = 13$:

Man erhält $(1/5)^{13}/13 = 6 \cdot 10^{-11} << 10^{-8}$. Das Ergebnis ist sogar auf 10 Dezimalen genau !

Beispiele (In der Praxis verwendet man noch „Schutzstellen“, so dass nk z.B. zu $nk+10$ wird !):

nk	nk+10	2n+1 für arctan(1/5)	2n+1 für arctan(1/239)
100	110	159	47
1000	1010	1445	425
10000	10010	14323	4209
20000	20010	28629	8415
100000	100010	143083	42051
200000	200010	286151	84095
1000000	1000010	1430691	420457
2000000	2000010	2861369	840907
10000000 = 10 ⁷	10000010	14306781	4204517
100000000 = 10 ⁸	100000010	143067671	42045115

Hinweise zu den verwendeten Datenstrukturen:

Zur Berechnung der Nachkommastellen von π wird das **10000000000 - System** verwendet, d.h. anstelle der Basis 10 verwenden wir nun die Basis $10000000000 = 10^{10}$.

Die Ziffern zur Basis 10000000000 werden in einem Array namens x abgespeichert.

Bei z.B. gewünschten 10000 Dezimalen bedeutet dies ein array, welches von x[0] bis zu x[10000] reicht. Die Vorkommastelle 3 wird stets in x[0] abgelegt, die Dezimalen in x[1] bis x[n].

Die Grundrechenarten + - / für das Rechnen im 10000000000 – System :

Beispiel für die Summenbildung mit 3 Registern der Länge 10:

Zahl	x[0]	x[1]	x[2]
a	0000006053	3001528335	2970071009
b	0000000247	9459107378	8450300256

Die Summe $c = a + b$ wird nun "von rechts nach links mit Übertrag" wie folgt gebildet.

1. x[2] von a + x[2] von b = x[2] von c = 1420371265 ; Übertrag = 1
2. x[1] von a + x[1] von b + Übertrag = x[1] von c = 2460635714; Übertrag = 1
3. x[0] von a + x[0] von b + Übertrag = x[0] von c = 6301 ; Übertrag = 0

Bei der Pi-Berechnung kann ein Übertrag von x[0] nicht vorkommen, denn x[0] hat immer den Wert 3 .

Die Zeichenkette x[0] , x[1] x[2] stellt die Summe c dar : 6301,24606357141420371265

Beispiel für die Differenzbildung mit 3 Registern der Länge 10:

Zahl	x[0]	x[1]	x[2]
a	0000006053	3001528335	2970071009
b	0000000247	9459107378	8450300256

Die Differenz $c = a - b$ wird nun "von rechts nach links mit Übertrag" wie folgt gebildet.

1. x[2] von a - x[2] von b = x[2] von c = -5480229247 < 0
Falls Differenz negativ, dann setze Differenz = Differenz + Basis und setze Übertrag = 1
also x[2] von c = 4519770753 ; Übertrag = 1
2. x[1] von a - x[1] von b - Übertrag = x[1] von c = -6457579044 < 0
also x[1] von c = 3542420956 ; Übertrag = 1
3. x[0] von a - x[0] von b - Übertrag = x[0] von c = 0000005805 ; Übertrag = 0

Die Zeichenkette x[0] , x[1] x[2] stellt die Differenz c dar : 5805,35424209564519770753

Beispiel für die Quotientenbildung mit 3 Registern der Länge 10 (Beispiel aus der Machin-Reihe):

Zahl	x[0]	x[1]	x[2]
a	0000000001	0000000000	0000000000
b	= 57121 (Datentyp long !)		

Der Quotient $c = a / b$ wird nun "von links nach rechts mit Übertrag" wie folgt gebildet.

1. tmp = x[0] + Übertrag*Basis = 1
tmp div b = x[0] von c = 0
tmp mod b = Übertrag = 1
2. tmp = x[1] + Übertrag*Basis = 10000000000
tmp div b = x[1] von c = 175066
Nullen einfügen ! x[1] von c = 0000175066
tmp mod b = Übertrag = 55014
3. tmp = x[2] + Übertrag*Basis = 550140000000000
tmp div b = x[2] von c = 9631133908
tmp mod b = Übertrag = 41132

Die Zeichenkette x[0] , x[1] x[2] stellt den Quotienten c dar : 0,00001750669631133908

Hier wird übrigens klar, dass wir mindestens ein zusätzliches Register mitführen müssen, damit Rundungsfehler bzw. Divisionsreste aufgefangen werden.

Algorithmen zu den Grundrechenarten (mit beliebiger Genauigkeit):

Gerechnet wird im System mit der Basis 10000000000 (Ziffern von 0 bis 9999999999).
Somit bearbeiten wir immer 10-stellige Zahlen, die in Registern abgespeichert werden.
Wegen der ungenauen Division verwenden wir noch 2 zusätzliche Register (Schutzstellen).
Für eine gewünschte Genauigkeit mit nk Dezimalen benötigen wir daher außer dem Register x[0] noch weitere $nk \div 10 + 2$ Register. nk sollte möglichst durch 10 teilbar sein .

Zur Berechnung der Teilterme der ArkusTangensSummen benötigen wir Registerfelder der Form

`long[] reg = new long [n+1]` mit $n = nk \div 10 + 2$.

Konkret werden folgende Registerfelder benötigt:

regSum5 für die Berechnung des $\arctan(1/5)$

regSum239 für die Berechnung des $\arctan(1/239)$

regXPot für die Aufnahme der jeweiligen x-Potenz der ArkusTangensSumme.

regXdurchK für die Aufnahme des jeweiligen $x^{2k+1}/(2k+1)$ -Wertes der ArkusTangensSumme.

Für eine Berechnung von Pi auf 10^6 Dezimalen werden somit ca. 400000 Register je 8 Bytes benötigt.
Das sind mehr als 3MB für die Register.

Die Berechnung der ArkusTangensSummen beginnt mit dem jeweiligen x-Wert.

Für die erste Summe ist das $x=1/5$. Die Summe hat also den Anfangswert $1/5$.

In einer k-Schleife kann man dann bei jedem Umlauf durch 25 dividieren, denn $x^3 = (1/5)^3 = 1/5/25$ und $x^5 = (1/5)^5 = 1/5/25/25$ usw. .

Jede Potenz von $(1/5)$ wird in der k-Schleife noch durch $(2k+1)$ dividiert .

Abwechselnd werden die so gebildeten Potenzterme zur aktuellen Summe addiert oder von ihr subtrahiert.

Die zweite Summe mit $x=1/239$ wird entsprechend behandelt.

Der Näherungswert für Pi ist dann $4 \cdot (4 \cdot \text{Summe1} - \text{Summe2})$.

Anmerkung:

Da ein einzelnes Register vom Java-Datentyp long ist, darf jegliches Operationsergebnis die Zahl $\text{longmax} = 2^{63}-1 = 9223372036854775807 = 9,223372036854775807 \cdot 10^{18}$ nicht überschreiten.
Insbesondere muss bei der Division darauf geachtet werden, dass der Term $\text{dividend}[i] + \text{uebertrag} \cdot \text{cBasis}$ unter diesem Wert bleibt .

Da $\text{dividend}[i] < 10000000000$ und $\text{cBasis} = 10000000000$, muss $\text{uebertrag} < 922337203$ sein .

Der uebertrag tritt zum Beispiel beim Term $(2n+1)$ auf . Falls 10^8 Dezimalen berechnet werden sollen, so ist $(2n+1)$ höchstens gleich 143067671 (siehe Tabelle).

Dies liegt noch unterhalb des Wertes 922337203 .

Deswegen sollten mit der gewählten Datenstruktur fehlerfrei $10^8 = 100$ Millionen Dezimalen berechnet werden können. Die Rechenzeit wäre hierfür allerdings gewaltig !

Anmerkung: Genauere Untersuchungen zeigen, dass man mit dem Java-Datentyp „long“ sogar im 10^{12} - System rechnen kann.

Geschwindigkeitssteigerung (Optimierung):

Da die Summanden nach jedem Umlauf immer kleiner werden, muss man nur solche Register in die Rechnung miteinbeziehen, für deren Index (Registernummer) nr gilt: $(1/z)^k / k < 10^{-10nr}$; $z = 5$; $z = 239$
Dies ist für Nummern ab $nr = \lceil (\lg(z) \cdot k + \lg(k)) / 10 \rceil$ der Fall.

Die Register mit den Nummern 0 bis nr-1 müssen also nicht neu berechnet werden.

Die Rechenzeit verringert sich durch diese Maßnahme um ca. 40% !

Die Java-Klasse (ohne Optimierung):

```
public class MachinPiLong {

    // Pi = 4 * [ 4*arctan(1/5) - arctan(1/239) ] ; MACHIN-Reihe
    // arctan(x) = x - x^3/3 + x^5/5 - x^7/7 + ... +(-1)^n*x^(2n+1)/(2n+1)

    public final static int cMaxStellen = 10000000; // = 10^7 ; evtl. auch höher setzen
    public final static long cBasis = 10000000000L; // 10 hoch 10
    public final static int cRegStellen = 10;
    public final static int cZusatzRegister = 2;
    public final static int cRegisterAnzahl = cMaxStellen/cRegStellen + cZusatzRegister+1;

    public static long[] RegSumme(long[] sum1, long[] sum2, int n) {
        // sum1 = sum1 + sum2
        long sum, uebertrag = 0L;
        for (int i = n; i >= 0; i--) {
            sum = sum1[i] + sum2[i] + uebertrag;
            uebertrag = sum / cBasis;
            sum1[i] = sum % cBasis;
        }
        return sum1;
    }

    public static long[] RegDifferenz(long[] minuend, long[] subtrahend, int n) {
        // minuend = minuend - subtrahend
        long diff, uebertrag = 0L;
        for (int i = n; i >= 0; i--) {
            diff = minuend[i] - subtrahend[i] - uebertrag;
            if (diff < 0L) {
                uebertrag = 1L;
                diff = diff + cBasis;
            } else
                uebertrag = 0L;
            minuend[i] = diff;
        }
        return minuend;
    }

    public static long[] RegQuotient(long[] dividend, long divisor, int n) {
        // dividend = dividend / divisor
        long quot, uebertrag = 0L;
        for (int i = 0; i <= n; i++) {
            quot = dividend[i] + uebertrag * cBasis;
            dividend[i] = quot / divisor;
            uebertrag = quot % divisor;
        }
        return dividend;
    }

    public static long[] RegQuotient2(long[] dividend, long divisor, int n) {
        // speziell für regXdurchK; eigenes Registerfeld erforderlich !
        long[] regXdurchK = new long[n+1];
        long quot, uebertrag = 0L;
        for (int i = 0; i <= n; i++) {
            quot = dividend[i] + uebertrag * cBasis;
            regXdurchK[i] = quot / divisor;
            uebertrag = quot % divisor;
        }
        return regXdurchK;
    }
}
```

```

static int k1max,k2max,anzahlRegs;
static long[] reg1, regSum5, regSum239, regXpot, regXdurchK;

public static void init(int gesStellen) {
    int z = gesStellen + cZusatzRegister*cRegStellen;
    k1max = (int) (1.0 + z / Math.Log10(5));
    k2max = (int) (1.0 + z / Math.Log10(239));
    anzahlRegs = z / cRegStellen;
    regSum5 = new long[anzahlRegs+1]; regSum5[0]=1L;
    regSum239 = new long[anzahlRegs+1]; regSum239[0]=1L;
    regXpot = new long[anzahlRegs+1]; regXpot[0]=1L;
    regXdurchK = new long[anzahlRegs+1];
}

public static void berechnung() {
    boolean negativ;
    int k;
    // berechne arctan (1/5)
    regXpot = RegQuotient(regXpot, 5, anzahlRegs); // x = 1/5
    regSum5 = RegQuotient(regSum5, 5, anzahlRegs); // summe = 1/5
    negativ = true;
    k = 1;
    while (k < k1max) {
        k += 2;
        regXpot = RegQuotient(regXpot, 25, anzahlRegs);
        regXdurchK = RegQuotient2(regXpot, k, anzahlRegs);
        if (negativ) {
            regSum5 = RegDifferenz(regSum5, regXdurchK, anzahlRegs);
            negativ = false;
        } else { // Vorzeichen +
            regSum5 = RegSumme(regSum5, regXdurchK, anzahlRegs);
            negativ = true;
        }
    } // while

    // berechne arctan (1/239)
    regXpot = new long[anzahlRegs+1]; // altes Feld löschen
    regXpot[0] = 1L; // Zahl 1 für Kehrwertbildung !
    regXpot = RegQuotient(regXpot, 239, anzahlRegs); // x = 1/239
    regSum239 = RegQuotient(regSum239, 239, anzahlRegs); // summe = 1/239
    negativ = true;
    k = 1;
    while (k < k2max) {
        k += 2;
        regXpot = RegQuotient(regXpot, 57121, anzahlRegs);
        regXdurchK = RegQuotient2(regXpot, k, anzahlRegs);
        if (negativ) {
            regSum239 = RegDifferenz(regSum239, regXdurchK, anzahlRegs);
            negativ = false;
        } else { // Vorzeichen +
            regSum239 = RegSumme(regSum239, regXdurchK, anzahlRegs);
            negativ = true;
        }
    } // while

    // Gesamtterm berechnen
    regSum5 = RegSumme(regSum5, regSum5, anzahlRegs); // 2*atan(1/5)
    regSum5 = RegSumme(regSum5, regSum5, anzahlRegs); // 4*atan(1/5)
    regSum5 = RegDifferenz(regSum5, regSum239, anzahlRegs); // 4*atan(1/5)-atan(1/239)
    regSum5 = RegSumme(regSum5, regSum5, anzahlRegs); // 2*(4*atan(1/5)-atan(1/239))
    regSum5 = RegSumme(regSum5, regSum5, anzahlRegs); // 4*(4*atan(1/5)-atan(1/239))
}

```

```

public static void ausgabe() {
    // Konsole
    String kette = "";
    System.out.print(regSum5[0]+",");
    for (int j=1; j<=anzahlRegs-cZusatzRegister;j++) {
        if ( (j-1) % 20 == 0) System.out.println();
        kette = String.valueOf(regSum5[j]);
        while (kette.length() < cRegStellen) kette='0'+kette;
        System.out.print(kette+" ");
    }
    System.out.println();
}

public static String ausgabePiString() {
    String s="Pi = ";
    String kette = "";
    s = s + regSum5[0] + ",";
    for (int j = 1; j <= anzahlRegs - cZusatzRegister; j++) {
        kette = String.valueOf(regSum5[j]);
        while (kette.length() < cRegStellen)
            kette = '0' + kette;
        s = s + kette;
    }
    return s;
}
}
}

```

Einige Messwerte bzgl. des Zeitaufwands (Java7 – optimierte Version) :

Anzahl der Dezimalen	1000	2000	10000	20000	50000	100000	200000	1000000=10 ⁶
die letzten 6 Dezimalen	...201989	...759009	...375678	...755178	...236041	...624646	...759928	...458151
Zeit / s Core I7-920		0,03	0,23	0,75	4,4	17,5	69,4	1725,7
Zeit / s Core I5- 4570S		0,03	0,14	0,44	2,6	10	43,6	1038,9
Zeit / s Core I7-920 (nicht optimiert !)		0,04	0,46	1,3	7,3	29,2	116,5	3003

Zum Vergleich: Derive5 (mit Core I7-920) benötigt für 10⁶ Dezimalen ca. 100 min. = 6000s !