

Taylorpolynome sind ganzzrationale Funktionen $T(x)$, welche eine bestimmte andere Funktion $f(x)$ in der Umgebung einer vorgegebenen Stelle x_0 approximieren.

$$T(x) = \sum_{k=0}^n a_k (x - x_0)^k = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \dots + a_n(x - x_0)^n$$

Je höher der Grad n des Polynoms, desto besser wird die Approximation in der Umgebung von x_0 . Für $n = \infty$ spricht man von einer **Taylorreihe**, welche identisch ist mit der Funktion f .

Erzeugung eines Taylorpolynoms:

Das Prinzip für die Bildung des Taylorpolynoms T ist, **dass jede Ableitung von $T(x)$ an der Stelle x_0 mit der entsprechenden Ableitung von $f(x)$ übereinstimmen muss**.

Es gilt also für i von 0 bis n : $T^{(i)}(x_0) = f^{(i)}(x_0)$

Daraus folgen :

$$f^{(0)}(x_0) = a_0$$

$$f^{(1)}(x) = a_1 + 2a_2(x-x_0) + 3a_3(x-x_0)^2 + 4a_4(x-x_0)^3 + \dots + na_n(x-x_0)^{n-1} \rightarrow f^{(1)}(x_0) = a_1$$

$$f^{(2)}(x) = 2a_2 + 6a_3(x-x_0) + 12a_4(x-x_0)^2 + 20a_5(x-x_0)^3 + \dots + n(n-1)a_n(x-x_0)^{n-2} \rightarrow f^{(2)}(x_0) = 2a_2$$

$$f^{(3)}(x) = 6a_3 + 24a_4(x-x_0) + 60a_5(x-x_0)^2 + \dots + n(n-1)(n-2)a_n(x-x_0)^{n-3} \rightarrow f^{(3)}(x_0) = 6a_3$$

Allgemein gilt: $f^{(i)}(x_0) = i! \cdot a_i$ bzw.

$$a_i = \frac{f^{(i)}(x_0)}{i!}$$

Dann ist

$$T(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

Taylorpolynom für f , entwickelt an der Stelle x_0

Hierfür ein Beispiel: $f(x) = \sin(x)$; $x_0 = 0$. (genauere Analyse weiter unten)

$$f'(x) = \cos(x)$$

$$f''(x) = -\sin(x)$$

$$f'''(x) = -\cos(x)$$

$$f^{(4)}(x) = \sin(x) = f(x)$$

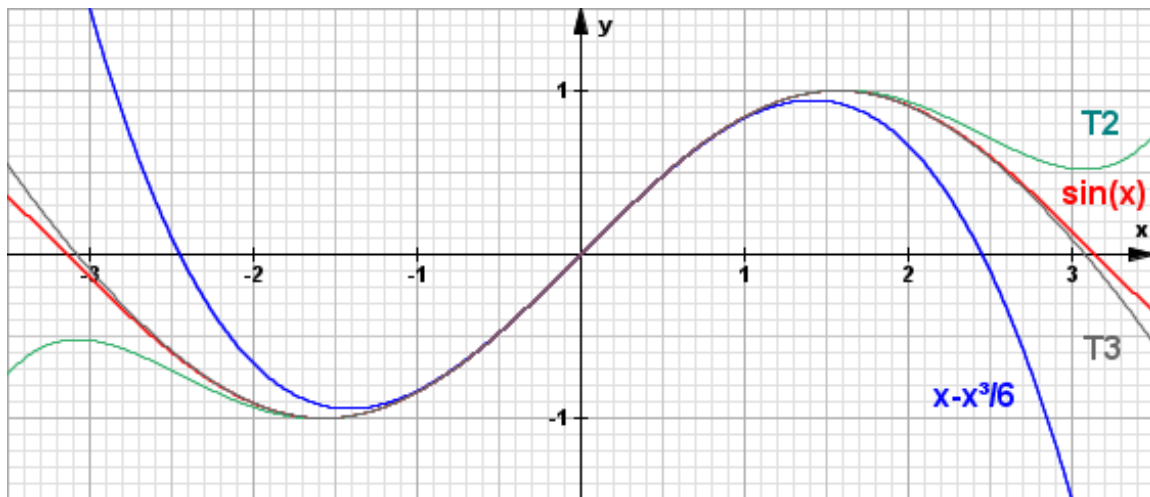
Also ist $f'(0) = 1$ $f''(0) = 0$ $f'''(0) = -1$ $f^{(4)}(0) = 0$ usw.

Jedes 2. Glied fällt weg und für die restlichen Glieder alternieren die Vorzeichen.

Das Taylorpolynom für $\sin(x)$ lautet dann:

$$\sin(x) \approx T(x) = \sum_{k=0}^n \frac{(-1)^k}{(2k+1)!} x^{2k+1} = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} \pm \dots$$

Grafische Veranschaulichung:



Anmerkungen:

T2 ist das Polynom für $k = 0$ bis 2, also bis zum Term $x^5 / 120$.

T3 ist das Polynom für $k = 0$ bis 3, also bis zum Term $x^7 / 5040$.

Fazit: Je höher der Grad n gewählt wird, um so besser ist die Approximation !

Weitere Beispiele für Taylorreihen :

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \pm \dots \quad ; x \in \mathbb{R}$$

$$\arctan(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \pm \dots \quad ; |x| \leq 1$$

$$\ln(1+x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \pm \dots \quad ; -1 < x \leq 1$$

Achtung: Hier beginnt der Summenindex mit $k=1$.

Die Reihen für $\arctan(x)$ und $\ln(1+x)$ **konvergieren sehr langsam**.

Weiter unten werden schnellere Alternativen erläutert.

Computerprogrammierung:

Zur Berechnung von Approximationswerten mit dem Computer muss ein Programm die Potenzsummen der Taylorreihe bilden können, was aber mit einer for-Schleife recht einfach erledigt werden kann. Da mit größer werdendem Exponenten die (Potenz-) Summanden in der Regel immer kleiner werden, ist es aus numerischen Gründen ratsam, die Summanden “von hinten nach vorne” abzuarbeiten.

Beispiel zur Verdeutlichung des numerischen Problems (JAVA 7):

$$\sum_{k=0}^{17} \frac{x^k}{k!} = 1,0100501670841682 \quad , \quad \text{aber} \quad \sum_{k=17}^0 \frac{x^k}{k!} = 1,0100501670841680$$

Das auf 17 Dezimalen genaue Ergebnis ist übrigens: 1,01005016708416805

JAVA-Methode:

```
public static double f(double x, int k) {
    // Beispielfunktion für Taylorreihen
    // hier: ln(x)
    return 2.0 / (2 * k + 1) * Math.pow((x - 1) / (x + 1), 2 * k + 1);
}

public static double taylorReihe(double x, int n) {
    // berechnet sum(f(x,k),k,0,n)
    double sum = 0;
    for (int k = n; k >= 0; k--)
        sum = sum + f(x, k);
    return sum;
}
```

1) Die Taylorreihe für e^x :

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \dots + \frac{x^n}{n!} + R_n ; x > 0$$

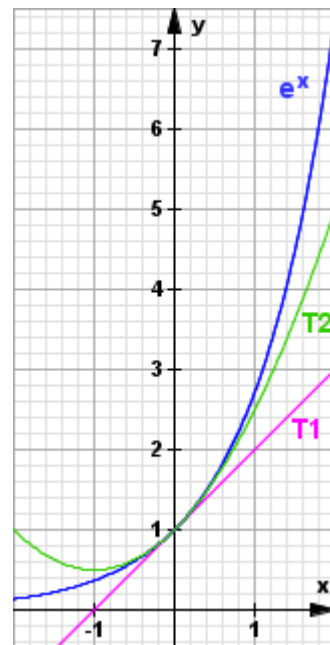
Für $x < 0$ verwende man $e^{-x} = 1/e^x$.

Anmerkung: Ersetzt man x durch $x \cdot \ln(a)$, so kann man mit der Formel

auch $e^{x \cdot \ln(a)} = a^x$ berechnen.

Die e^x -Formel lässt sich geschickt umformen, so dass man ohne die Fakultäten und Potenzen auskommt:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + \frac{x}{1} \cdot \left(1 + \frac{x}{2} \cdot \left(1 + \frac{x}{3} \cdot \left(1 + \frac{x}{4} \cdot \left(1 + \dots + \frac{x}{n-1} \cdot \left(1 + \frac{x}{n}\right)\right)\right)\right)\right) + R_n ; x > 0$$



Für das sog. "Restglied" R_n gilt: $R_n = \frac{x^{n+1}}{(n+1)!} \cdot e^{\vartheta \cdot x}$ mit $0 < \vartheta < 1$

Die Reihe konvergiert am besten in der Nähe von 0 !

Daher sollte man den Exponenten x ggfs. durch die folgende Umformung verkleinern : $e^x = \left(e^{\frac{x}{m}}\right)^m$.

Zum Beispiel kann man $m = (\text{int})(5x)$ für $x > 0,2$ wählen . Dann gilt: $x/m \approx 0,2$.

Man berechnet also erst $e^{\frac{x}{m}} \approx e^{0,2}$ mit der Summenformel und potenziert das Ergebnis mit m .

Für den Fall $e^{0,2}$ ist der maximale Fehler laut Restglied $\frac{0,2^{n+1}}{(n+1)!} \cdot e^{\vartheta \cdot 0,2} < \frac{0,2^{n+1}}{(n+1)!} \cdot e^{0,2} \approx \frac{1,222}{5^{n+1} \cdot (n+1)!}$

Will man z.B. $e^{0,2}$ auf 15 Dezimalen (= Nachkommastellen) genau berechnen, so gilt der Ansatz:

$$\frac{1,222}{5^{n+1} \cdot (n+1)!} < 5 \cdot 10^{-16} \Leftrightarrow 5^{n+1} \cdot (n+1)! > 2,444 \cdot 10^{15}$$

Durch Probieren findet man, dass für diese gewünschte Genauigkeit $n = 11$ ausreicht .

Es zeigt sich, dass sogar $n = 10$ genügt, weil die Abschätzung doch sehr großzügig war !

Mit Java7 (und $n = 10$) ergibt sich $e^{0,2} = 1.2214027581601694$ (In der Tat: 15 Dezimalen genau !).

Für e^x mit $x > 1$ reicht $n = 10$ nicht mehr aus, wie man an den folgenden Beispielen erkennen kann:

$e^2 = 7,389056098930625$	13 Dezimalen korrekt	(richtig: 7,38905609893065022723...)
$e^3 = 20,085536923187565$	12 Dezimalen korrekt	(richtig: 20,0855369231876677409...)
$e^{10} = 22026,465794806336$	9 Dezimalen korrekt	(richtig: 22026,4657948067165169...)

Bei diesen Exponenten x pflanzt sich der Fehler um ein Vielfaches fort, da ja das Ergebnis von $e^{0,2}$ noch mit m potenziert werden muss (Potenzierung vergrößert den Fehler um ca. Faktor m !).

Eine Erhöhung des n -Wertes auf $n = 11$ bringt eine Verbesserung bei $e^2 = 7.389056098930651$ (14 richtige Dezimalen). Dies wird jedoch auch durch weitere Erhöhung von n nicht mehr genauer. Ähnlich bei $e^3 = 20,08553692318767$

Hinweis:

Setzt man in der e^x -Formel $x = 1$, so erhält man die **Eulersche Zahl** $e \approx 2,7182818284590452353$.

Beispielrechnung für e mit Restgliedabschätzung:

Das Restglied lässt sich speziell für e (e^x mit $x = 1$) geschickt abschätzen:

$$\text{Für } x = 1 \text{ gilt: } R_n = \frac{1}{(n+1)!} \cdot e^{\vartheta} \text{ mit } 0 < \vartheta < 1; \Rightarrow R_n < \frac{e}{(n+1)!}$$

Wir wollen e auf 15 Dezimalen genau berechnen, d.h. $R_n = \frac{e}{(n+1)!} < \varepsilon = 5 \cdot 10^{-16}$.

Die Umformung liefert $(n+1)! > 2e \cdot 10^{15} \approx 5,437 \cdot 10^{15}$. Durch Probieren findet man $n \geq 17$.

In der Tat ist $\sum_{k=0}^{17} \frac{1}{k!} = 2,7182818284590455$ (JAVA7) auf 15 Dezimalen genau!

Die 16. Dezimale ist allerdings falsch.

JAVA-Methode für e^x :

```
public static double expReihe(double x, int n) {
    // berechnet sum(x^k/k!, k, 0, n)

    if (x == 0)
        return 1;
    boolean xNegativ = (x < 0);
    if (xNegativ)
        x = -x;
    boolean reduziert = (x > 0.2);
    int j = 1;
    if (reduziert) {
        // falls x > 0.2: e^x = (e^(x/j))^j ; j = [5x]
        j = (int) (5 * x);
        x = x / j;
    }

    // Beginn der Iteration
    double sum = 0.0;
    for (int k = n + 1; k > 0; k--)
        sum = sum / k * x + 1.0;

    if (reduziert)
        sum = Math.pow(sum, j);
    if (xNegativ)
        sum = 1 / sum;
    return sum;
}
```

2) Die Taylorreihe für $\ln(x)$ bzw. $\ln(1+x)$:

$$\ln(1+x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \pm \dots \quad ; \quad -1 < x \leq 1$$

Da diese Reihe sehr langsam konvergiert, konstruiert man mittels $\ln(1+x) - \ln(1-x) = \ln((1+x)/(1-x))$ eine neue, besser konvergierende Reihe.

$$\ln(1+x) - \ln(1-x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k} - \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(-x)^k}{k} = x - \frac{x^2}{2} + \frac{x^3}{3} \mp \dots - (-x - \frac{(-x)^2}{2} + \frac{(-x)^3}{3} \mp \dots)$$

Man erkennt, dass sich Potenzen mit geraden Exponenten wegschubtrahieren. Daher bleibt folgendes:

$$\ln\left(\frac{1+x}{1-x}\right) = \ln(1+x) - \ln(1-x) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots\right) = 2 \cdot \sum_{k=0}^{\infty} \frac{x^{2k+1}}{2k+1}$$

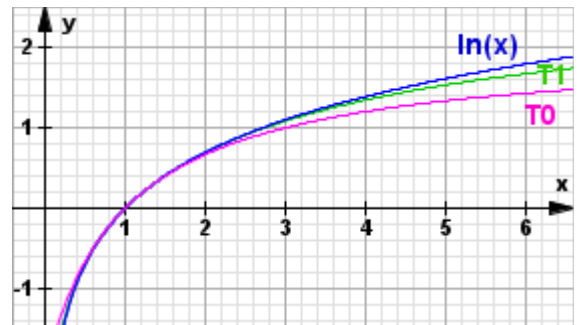
Die Substitution $y = \frac{1+x}{1-x} \Leftrightarrow x = \frac{y-1}{y+1}$ liefert $\ln(y) = 2 \cdot \sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot \left(\frac{y-1}{y+1}\right)^{2k+1}$

Durch Umbenennung erhalt man endlich die $\ln(x)$ - Reihe:

$$\ln(x) = 2 \cdot \sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot \left(\frac{x-1}{x+1}\right)^{2k+1} + R_n = 2 \cdot \left(\frac{x-1}{x+1} + \frac{1}{3} \cdot \left(\frac{x-1}{x+1}\right)^3 + \dots + \frac{1}{2n+1} \cdot \left(\frac{x-1}{x+1}\right)^{2n+1}\right) + R_n \quad ; \quad x > 0$$

Fur das Restglied gilt:

$$|R_n| = \frac{(x-1)^2}{2 \cdot |x|} \cdot \left(\frac{x-1}{x+1}\right)^{2n}$$



Diese Reihe lasst sich mittels der Substitution $z = (x-1)/(x+1)$ einfacher notieren:

$$\ln(x) = 2 \cdot \sum_{k=0}^{\infty} \frac{z^{2k+1}}{2k+1} = 2 \cdot \left(z + \frac{z^3}{3} + \frac{z^5}{5} + \dots + \frac{z^{2n+1}}{2n+1} + \dots\right) \quad \text{mit} \quad z = \frac{x-1}{x+1}$$

Die Reihe konvergiert am besten in der Nahe von $x = 1$.

Deswegen sollte man x in Faktoren zerlegen, die in der Nahe von 1 liegen, falls $x > \sqrt{2} \approx 1,414$.

Vorgehensweise:

Man zerlegt x in $2^m \cdot 2^{-m} \cdot x$ mit geeignetem m . So erhalt man wegen $\ln(x) = \ln(ab) = \ln(a) + \ln(b)$ folgendes:

$$\ln(x) = \ln(2^m) + \ln(x/2^m) = \ln(2^{2m/2}) + \ln(x/2^m) = 2m \cdot \ln(2^{1/2}) + \ln(x/2^m) = 2m \cdot \ln(\sqrt{2}) + \ln(x/2^m)$$

Dabei sollte $x/2^m$ zwischen $1/\sqrt{2}$ und $\sqrt{2}$ liegen. Wie erreicht man das?

Am besten dividiert man x so lange durch 2, bis das Ergebnis unter $\sqrt{2}$ liegt.

Beispiel: $x = 2500$

$2500/2 = 1250$ $1250/2 = 1125$ $1125/2 = 562,5$ Ergebnis: $2500/2^{11} = 1,2207... < \sqrt{2}$

Mit $m=11$ erhalten wir: $\ln(2500/2^{11}) = 22 \ln(\sqrt{2}) + \ln(2500/2^{11})$

Tipp: Falls $0 < x < 1$, dann verwende man $\ln(1/x) = \ln(1) - \ln(x) = -\ln(x)$ bzw **$\ln(x) = -\ln(1/x)$**

Beispielrechnung für $\ln(2)$ mit Restgliedabschätzung:

Wir wollen $\ln(2)$ auf 15 Dezimalen genau berechnen, d.h. $|R_n| = \frac{1}{2 \cdot 2} \cdot \left(\frac{1}{3}\right)^{2n}$

Aus dem Ansatz $R_n < 5 \cdot 10^{-16}$ folgt:

$$\frac{1}{4} \cdot \left(\frac{1}{3}\right)^{2n} < 5 \cdot 10^{-16} \Leftrightarrow 3^{2n} > \frac{1}{2} \cdot 10^{15} \Leftrightarrow 2n \cdot \lg(3) > \lg\left(\frac{1}{2}\right) + 15 \Leftrightarrow n > \frac{15 - \lg(2)}{2 \cdot \lg(3)} \approx 15,4$$

Daher sollte $n=16$ für die vorgegebene Genauigkeit genügen. Wir berechnen also (mit JAVA 7)

$$\ln(2) \approx \sum_{k=0}^{16} \frac{2}{2k+1} \cdot \left(\frac{1}{3}\right)^{2k+1} = 0.6931471805599455 \cdot \text{Dies ist auf 15 Dezimalen genau !}$$

Beispielrechnung für $\ln(100)$ mit Restgliedabschätzung:

$$\frac{99^2}{2 \cdot 100} \cdot \left(\frac{99}{101}\right)^{2n} < 5 \cdot 10^{-16} \Leftrightarrow \left(\frac{101}{99}\right)^{2n} > 9801 \cdot 10^{13} \Leftrightarrow 2n \cdot \lg\left(\frac{101}{99}\right) > \lg(9801) + 13 \Leftrightarrow n > \frac{13 + \lg(9801)}{2 \cdot \lg\left(\frac{101}{99}\right)} \approx 978,1$$

Immerhin müssen wir hier schon bis $n=979$ rechnen, um die vorgegebene Genauigkeit zu erreichen.

$$\text{Wir berechnen also (mit JAVA 7) } \ln(100) \approx \sum_{k=0}^{979} \frac{2}{2k+1} \cdot \left(\frac{99}{101}\right)^{2k+1} = 4.60517018598809 \cdot$$

14 Dezimalen sind genau, die 15. wird nicht angezeigt (müsste aber definitiv 1 sein !)

Fazit: Die oben angegebene Transformation **$\ln(x) = 2m \cdot \ln(\sqrt{2}) + \ln(x/2^m)$** ist aus Geschwindigkeitsgründen auf jeden Fall der direkten Methode vorzuziehen.

Die Berechnung **mit Transformation** liefert für $n = 9$ (10; 11; ...) jeweils das gleiche Ergebnis:
 $\ln(100) = 4.605170185988093$ (14 Stellen genau; in der 15. Stelle 3 statt richtigerweise 1)

Alternativ: Man kann auch **$\ln(x) = 2 \cdot \ln(\sqrt{x})$** verwenden (evtl. mehrmals), um x zu verkleinern !

JAVA-Methode:

```
public static double lnReihe(double x, int n) {
    //  $\ln(x) = 2 \cdot \sum (z^{2k+1} / (2k+1), k, 0, n)$ ;  $z = (x-1)/(x+1)$ 
    if (x <= 0)
        throw new ArithmeticException("x muss positiv sein !");
    if (x == 1)
        return 0;
    boolean kleiner1 = (x < 1);
    if (kleiner1)
        x = 1 / x;
    int m = 0;
    boolean reduziert = (x > 1.414);
    if (reduziert) {
        //  $\ln(x) = 2m \cdot \ln(\sqrt{2}) + \ln(x/2^m)$ 
        do {
            x = x / 2;
            m = m + 1;
        } while (x > 1.414);
    }
    double z = (x - 1) / (x + 1);
    double sum = 0;
    int expo;
    for (int k = n; k >= 0; k--) {
        expo = 2 * k + 1;
        sum = sum + Math.pow(z, expo) / expo;
    }
    sum = 2 * sum;
    if (reduziert) {
        double sum2 = 0;
        x = Math.sqrt(2);
        z = (x-1)/(x+1);
        for (int k = n; k >= 0; k--) {
            expo = 2 * k + 1;
            sum2 = sum2 + Math.pow(z, expo) / expo;
        }
        sum = sum + 4 * m * sum2;
    }
    if (kleiner1)
        sum = -sum;
    return sum;
}
```


3) Die Taylorreihe für $\sin(x)$:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \pm \dots + (-1)^k \frac{x^{2n+1}}{(2n+1)!} + R_n ; x \in \mathbb{R}$$

Grafik auf Seite 2 !

Für das sog. "Restglied" R_n gilt:
$$R_n = (-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!} \cdot \cos(\vartheta \cdot x) \text{ mit } 0 < \vartheta < 1$$

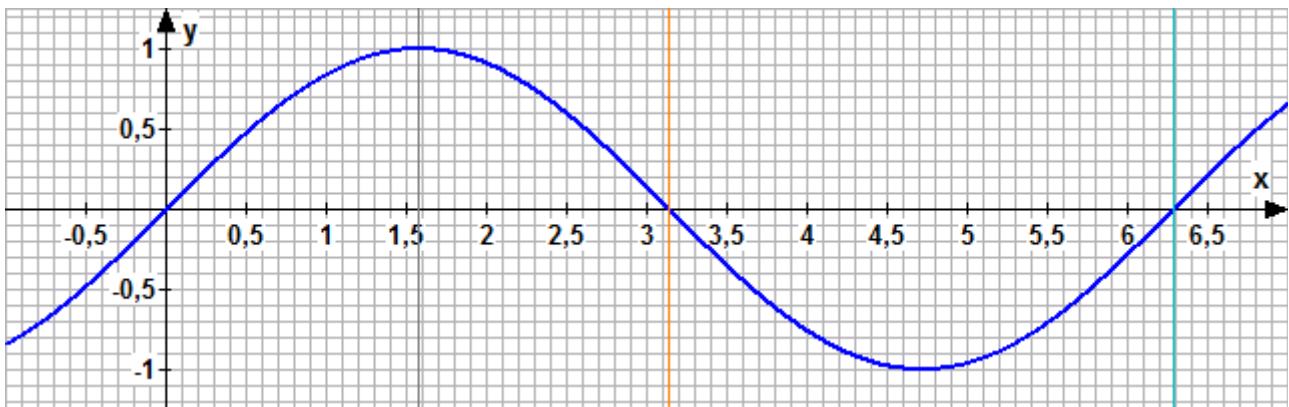
Das Restglied liegt zwischen $(-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!}$ und $(-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!} \cdot \cos(x)$. Maximaler Fehler: $\frac{x^{2n+1}}{(2n+1)!}$

Wählt man jedoch $0 < x < \pi / 2$, dann ist der Fehler kleiner als $\frac{\left(\frac{\pi}{2}\right)^{2n+1}}{(2n+1)!}$

So benötigt man $n=10$ für 15 richtige Dezimalen . Evtl. reicht auch $n=9$ oder ein noch kleineres n .

Wie transformiert man nun eine Zahl x so, dass sie in den Bereich $[0 ; \pi / 2]$ fällt ??

Dazu verwendet man die Periodizität des Sinus sowie andere Eigenschaften (s. Grafik) :



Für $x = 0$ gilt $\sin(0) = 0$ und für $x = \pi / 2$ gilt $\sin(\pi / 2) = 1$

Für $x < 0$ verwendet man: $\sin(x) = -\sin(-x)$.

Für $x \geq 2\pi$ verwendet man: $\sin(x) = \sin(x - n \cdot 2\pi)$; man ersetzt $x = x - 2\pi$, so lange $x \geq 2\pi$.

Für $\pi \leq x < 2\pi$ verwendet man: $\sin(x) = -\sin(x - \pi)$.

Für $\pi / 2 < x < \pi$ verwendet man: $\sin(x) = \sin(\pi - x)$.

Beispielrechnung für $\sin(2)$ mit Restgliedabschätzung:

$x=2$ muss zuerst transformiert werden mittels $\sin(2) = \sin(\pi - 2)$.

Wir wollen $\sin(2) = \sin(\pi - 2)$ auf 15 Dezimalen genau berechnen, d.h. $\frac{(\pi - 2)^{2n+1}}{(2n+1)!} \cdot \cos(\pi - 2) < 5 \cdot 10^{-16}$

Durch Probieren findet man $n = 9$. Daher ist zu berechnen:

$$\sin(2) \approx \sum_{k=0}^9 (-1)^k \frac{(\pi - 2)^{2k+1}}{(2k+1)!} = 0,9092974268256816 \text{ Sogar alle 16 Stellen (mit JAVA 7) sind korrekt !}$$

Beispielrechnung für $\sin(-25)$:

$x=-25$ muss zuerst transformiert werden:

$$\sin(-25) = -\sin(25) = -\sin(25 - 3 \cdot 2\pi) = \sin(25 - 3 \cdot 2\pi - \pi) = \sin(\pi - (25 - 3 \cdot 2\pi - \pi)) .$$

Vereinfacht ist das $\sin(8\pi - 25)$.

Da $8\pi - 25 \approx 0,13$ sehr klein ist, reicht hier eine Rechnung mit $n = 5$:

$$\sin(-25) \approx \sum_{k=0}^5 (-1)^k \frac{(8\pi - 25)^{2k+1}}{(2k+1)!} = 0,13235175009777206 \quad \text{14 Stellen (mit JAVA 7) sind korrekt !}$$

Betrachtung von \cos und \tan :

Der Cosinus lässt sich wegen $\cos(x) = \sin(x + \pi / 2)$ auf den Sinus zurückführen !

Der Tangens lässt sich wegen $\tan(x) = \sin(x) / \cos(x) = \sin(x) / \sin(x + \pi / 2)$ auf den Sinus zurückführen !

JAVA-Methoden:

```
public static double sinReihe(double x, int n) {
    // berechnet sum((-1)^k * x^(2k+1) / (2k+1)!, k, 0, n)
    final double doppelPi = 2 * PI;
    final double halbePi = PI / 2;
    boolean negativ = false;
    if (x != 0) {
        // transformiere ggfs.
        negativ = (x < 0);
        if (negativ)
            x = -x;
        while (x >= doppelPi)
            x = x - doppelPi;
        if (x >= PI) {
            x = x - PI;
            negativ = !negativ;
        }
        if (x > halbePi)
            x = PI - x;
    }
    if (x == 0)
        return 0;
    int vz = 1;
    double sum = 0.0;
    for (int k = 0; k <= n; k++) {
        sum = sum + vz / fak(2 * k + 1) * Math.pow(x, 2 * k + 1);
        vz = -vz;
    }
    if (negativ)
        return -sum;
    else
        return sum;
}

public static double cosReihe(double x, int n) {
    return sinReihe(x + PI / 2, n);
}

public static double tanReihe(double x, int n) {
    return sinReihe(x, n) / sinReihe(x + PI / 2, n);
}
```

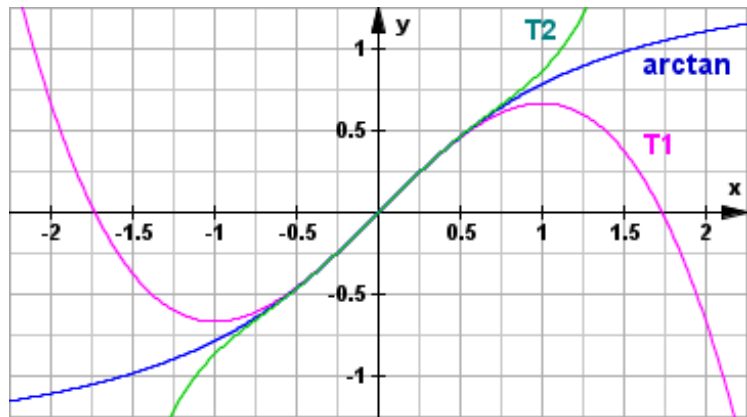
4) Die Taylorreihe für $\arctan(x)$:

$$\arctan(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} \pm \dots + (-1)^k \frac{x^{2n+1}}{2n+1} + R_n ; |x| \leq 1$$

Für das sog. Restglied R_n gilt:

$$R_n = (-1)^n \cdot \frac{x^{2n+1}}{2n+1} \cdot \frac{1}{1+\vartheta \cdot x^2} \text{ mit } 0 < \vartheta < 1$$

Die Reihe konvergiert ziemlich langsam und am besten in der Nähe von 0 !



Das Restglied liegt zwischen $(-1)^n \cdot \frac{x^{2n+1}}{2n+1}$ und $(-1)^n \cdot \frac{x^{2n+1}}{2n+1} \cdot \frac{1}{1+x^2}$. Maximaler Fehler also: $\frac{x^{2n+1}}{2n+1}$

Soll $\arctan(1)$ auf nk Dezimalen genau sein, muss n so gewählt werden, dass: $\frac{1}{2n+1} < 5 \cdot 10^{-(nk+1)}$.

Beispiel: $nk=15$. $\frac{1}{2n+1} < 5 \cdot 10^{-16} \Leftrightarrow 2n+1 > 2 \cdot 10^{15} \Leftrightarrow n \geq 10^{15}$ (evtl. genügt ein kleineres n)

Wählt man jedoch $x < 0,1$, dann ist der Fehler kleiner als $\frac{0,1^{2n+1}}{2n+1} = \frac{1}{(2n+1) \cdot 10^{2n+1}}$.

Für nk Nachkommastellen gilt dann: $\frac{1}{(2n+1) \cdot 10^{2n+1}} < 5 \cdot 10^{-(nk+1)} \Leftrightarrow (2n+1) \cdot 10^{2n+1} > 2 \cdot 10^{nk}$ bzw.

$$\Leftrightarrow \lg(2n+1) + 2n+1 > \lg(2) + nk \Leftrightarrow \lg(2n+1) + 2n > nk + \lg(2) - 1$$

Die folgende Tabelle zeigt, wie groß der maximale Folgenindex n mindestens sein muss, damit nk Dezimalen richtig sind (Voraussetzung $x < 0,1$) :

nk	8	9	10	11	12	13	14	15	16	17	18	19	20
n	4	4(5)	5	5	6	6	7	7	8	8	9	9	10

Fazit: Offensichtlich muss man für diesen Fall $n = nk \text{ div } 2$ wählen .

Wie aber bekommt man die x -Werte unter die Grenze von $0,1$??

Zur Transformation von $|x| > 1$ auf den Bereich $]0;1[$ verwendet man

$$\arctan(x) = 2 \cdot \arctan\left(\frac{x}{1+\sqrt{1+x^2}}\right)$$

Die Quadratwurzel lässt sich nach Heron schnell berechnen .

In der Regel ist eine **mehrfache** Anwendung der Formel erforderlich. Zum Beispiel gilt

$$\arctan(5) = 2 \cdot \arctan\left(\frac{5}{1+\sqrt{1+5^2}}\right) = 2 \cdot \arctan(0,81\dots) = 4 \cdot \arctan(0,35\dots) = 8 \cdot \arctan(0,17\dots) = 16 \cdot \arctan(0,08\dots)$$

Für große x kann man auch auf $\arctan(x) = \frac{\pi}{2} - \arctan\left(\frac{1}{x}\right)$ zurückgreifen, muss dann aber π kennen.

Beispielrechnung für arctan(5) mit Restgliedabschätzung:

x=5 muss zuerst transformiert werden (siehe oben). $\arctan(5) = 16 \cdot \arctan(0.08604899056617473)$.

Wir wollen arctan(5) auf 15 Dezimalen genau berechnen, d.h.

$$\frac{0,08604899\dots^{2n+1}}{2n+1} \cdot \frac{1}{1+0} < 5 \cdot 10^{-16} \quad \text{Laut obiger Tabelle reicht } n = 7. \quad \text{Daher ist zu berechnen:}$$

$$\arctan(5) \approx 16 \cdot \sum_{k=0}^7 (-1)^k \frac{0,0840848515\dots^{2k+1}}{2k+1} = 1,3734007669450157 \quad \text{15 Stellen (JAVA 7) sind korrekt !}$$

JAVA-Methode:

```
public static double arctanReihe(double x, int n) {
    // berechnet sum((-1)^k * x^(2k+1) / (2k+1), k, 0, n)
    if (x == 0)
        return 0;
    double faktor = 1;
    boolean negativ = (x < 0);
    if (negativ)
        x = -x;
    while (x > 0.1) {
        // transformiere
        x = x / (1 + Math.sqrt(1 + x * x));
        faktor = faktor * 2;
    }
    int vz = 1;
    double sum = 0.0;
    for (int k = 0; k <= n; k++) {
        sum = sum + vz * Math.pow(x, 2 * k + 1) / (2 * k + 1);
        vz = -vz;
    }
    sum = sum * faktor;
    if (negativ)
        return -sum;
    else
        return sum;
}
```