# 3.6 Drehungen in der Ebene

## 3.6.1 Die Drehmatrix

Gelegentlich müssen wir die Lage eines Teilchens in einem *ebenen* Koordinatensystem beschreiben, das gegenüber einem festen System um φ gedreht ist.

In der linearen Algebra lernen wir, dass eine Drehung durch eine lineare Transformation **r'** = R·**r** beschrieben werden kann. Unsere erste Aufgabe ist es, die Matrixelemente des Drehoperators R zu bestimmen.

Im festen Koordinatensystem C(O,**i**,**j**) hat der Punkt P die Koordinaten (x,y). Im System C'(O,**i'**,**j'**) lauten die Koordinaten desselben Punktes (x',y'). Das um φ gedrehte System C' hat denselben Ursprung wie C. φ ist der Winkel zwischen den positiven Halbachsen x' und x.

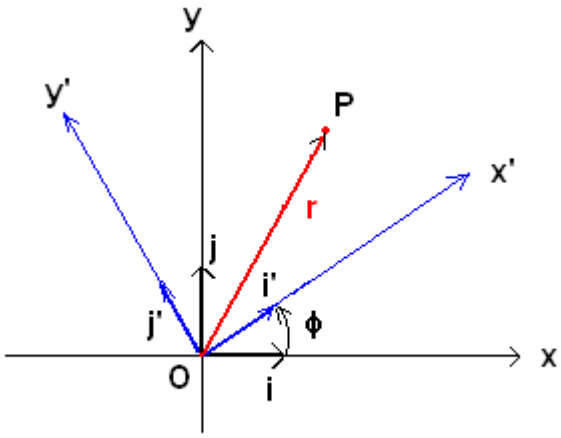



Fig. 3.6-1

Die Matrixdarstellung von R finden wir mit Hilfe der Skalarprodukte der Einheitsvektoren:

**i·i'** = cos φ, **j·j'** = cos φ, **i·j'** = cos(φ+π/2) = - sin φ, **j·i'** = cos(π/2-φ) = sin φ

Es gilt: **r** = x·**i** + y·**j** = x'·**i'** + y'·**j'**. Wenn wir **r** mit **i'** und **j'** multiplizieren, erhalten wir x' = **i'**·**r** und y' = **j'**·**r**, und wenn wir dann **r** durch x·**i** + y·**j** ersetzen, ergibt sich x' = **i'** ·( x **i** + y·**j**) = x·**i'**·**i** + y·**i'**·**j** = x·cos φ + y·sin φ und y' = **j'** ·**r** = **j'**·( x **i** + y·**j**) = x·(-sin φ) + y·cos φ. Wir erhalten also:

$$x' = x\cdot\cos\varphi + y\cdot\sin\varphi$$

$$y' = x\cdot(-\sin\varphi) + y\cdot\cos\varphi \quad (1)$$

Dieses Ergebnis können wir in Matrixform schreiben:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \bullet \begin{pmatrix} x \\ y \end{pmatrix} \qquad (2)$$

Die Matrix der Drehung ist also:

$$R = \begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \qquad (3)$$

Ebenso können wir (x,y) in Abhängigkeit von (x',y') berechnen: Wir multiplizieren **r** mit **i** und **j** und substituieren **r** durch x'·**i'** + y'·**j'**. Es folgt

$$x = x'\cdot\cos\varphi + y'\cdot(-\sin\varphi) \text{ und } y = x'\cdot\sin\varphi + y'\cdot\cos\varphi \quad (4)$$

Eine andere Art, die Beziehungen (4) zu erhalten, besteht in der Auflösung des Systems (1) nach x und y. Das kann man natürlich vonhand machen, aber wozu haben wir MuPAD?

**Programm 1:**

```
sol:=linsolve({x1=x*cos(f)+y*sin(f),y1=-x*sin(f)+
y*cos(f)},{x,y}):

  simplify(%)
```

$$[x = x1\cdot\cos(f) - y1\cdot\sin(f),\ y = y1\cdot\cos(f) + x1\cdot\sin(f)]$$

Man kann auch die inverse Matrix in der Gleichung **r** = $R^{-1}$·**r'** benutzen:

**Programm 2:**

```
reset():

mat:=Dom::Matrix():export(linalg):

R:=mat([[cos(fi),sin(fi)],[-sin(fi),cos(fi)]]):

r1:=mat([x1,y1]):

r:=R^(-1)*r1:

x:=simplify(r[1]);

y:=simplify(r[2]);
```

```
x1 cos(fi) - y1 sin(fi)

y1 cos(fi) + x1 sin(fi)
```

## 3.6.2 Anwendungen

In den Anwendungen wollen wir oft den Punkt P drehen und das Koordinatensystem fest halten. In einem solchen Fall gilt x' = r cos(α+φ) = r cosα cos φ – r sinα sinφ, worin α der Winkel zwischen **r** und der x-Achse ist. Der Vektor **r'** bildet mit dieser Achse den Winkel α + φ. Aber r cosα = x und r sinα = y. Also: x' = x cosφ - y sinφ.

Entsprechen gilt: y' = r sin(α +φ) = y cos φ + x sin φ. Die Matrix für die Drehung eines Punktes P um den Winkel φ entgegen dem Uhrzeigersinn ist

$$R_P = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \quad (5)$$

(Die Drehung eines Ortsvektors entgegen dem Uhrzeigersinn entspricht einer Drehung des Koordinatensystems im Uhrzeigersinn. $R_P$ ist die Inverse von (3) und wird in diesem Fall dadurch erhalten, dass man φ durch –φ ersetzt.)

Mit **Programm 3** können wir die gegensinnige Drehung des Punktes (x,y) um den Winkel φ =35° illustrieren.

Beachten Sie auch die Anwendung von **ViewingBox**!

**Programm 3:**

```
reset():://Drehung eines Punktes oder eines Pfeils
um den Punkt (0,0)

fi:=35*PI/180:
x:=3:y:=5:
mat:=Dom::Matrix():export(linalg):
R:=mat([[cos(fi),-sin(fi)],[sin(fi),cos(fi)]]):
r:=mat([x,y]):
r1:=R*r://Transformationsgleichung
x1:=float(r1[1]);
y1:=float(r1[2]);
ar:=plot::Arrow2d([0,0],[x,y],Color=RGB::Blue):
ar1:=plot::Arrow2d([0,0],[x1,y1],Color=RGB::Red):
plot(ar,ar1,ViewingBox=[-6..6,0..8],

Scaling=Constrained)

-0.4104260489

5.81648953
```

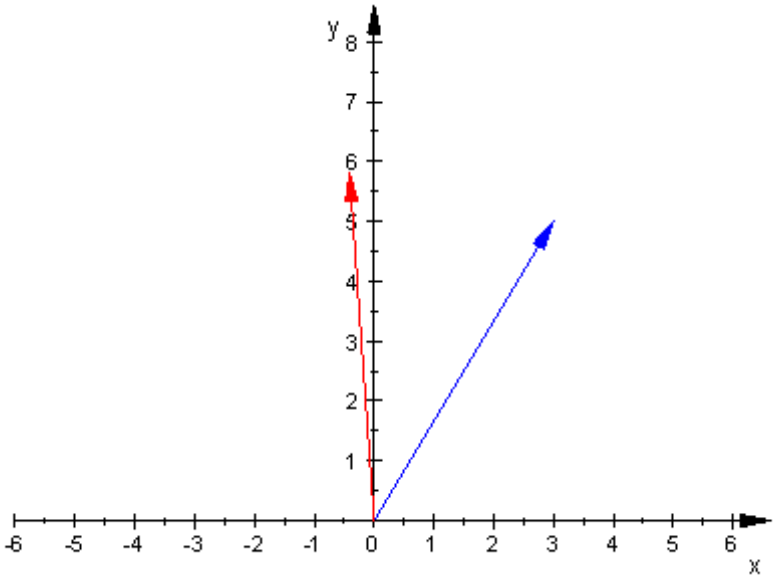



Fig. 3.6-2

In MuPAD gibt es die Funktion **`Transform2d`**, die das vorige Programm 3 etwas vereinfacht und die sehr nützlich ist, wenn es darum geht, *mehrere* Objekte zu transformieren, wobei die Funktion **`plot::Scene2d`** benutzt wird. Im folgenden **Programm 4** benutzen wir **`Transform2d`**. Leider kann man die Funktion **`Color=RGB`** nicht zusammen mit **`Transform2d`** einsetzen, was dann dazu führt, dass das transformierte Objekt dieselbe Farbe hat wie das Original, in unserem Fall ist es rot.

**Programm 4:**

```
reset():

fi:=35*PI/180:
x:=3:y:=5:
R := matrix([[cos(fi), -sin(fi)], [sin(fi),
cos(fi)]]):
ar := plot::Arrow2d([0, 0], [x, y], Color =
RGB::Red):
ar1:= plot::Transform2d(R,ar):

plot(ar,ar1,Scaling = Constrained, Layout =
Vertical);
```

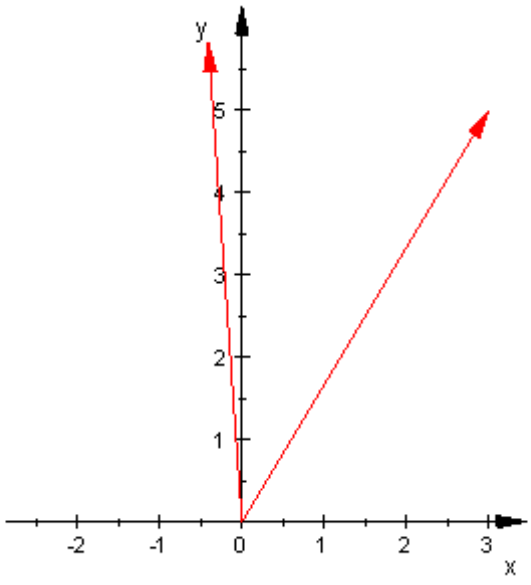



Fig. 3.6-3

Mit der Anweisung **`ar1::Matrix2d(R)`** können wir die Gestalt der Matrix R sehen, sowie auch mit **`a:=float(R).`** Aber das mit **`float(R)`** produzierte Resultat ist wesentlich ansprechender, was sich besonders bei $30^o$ zeigt, wo wir **`float`** nicht benutzen

```
a:=float(R);

a1:=ar1::Matrix2d(R)
```

$$\begin{pmatrix} 0.8191520443 & -0.5735764364 \\ 0.5735764364 & 0.8191520443 \end{pmatrix}$$

$$[0.8191520443,\ -0.5735764364,\ 0.5735764364,\ 0.8191520443]$$

Drehung um $30^o$:

```
a:=R;// ohne float

a1:=ar1::Matrix2d(R)
```

$$\begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$$

$$[0.8660254038,\ -0.5,\ 0.5,\ 0.8660254038]$$

Wenn der Vektor **r** im Punkt $P_0 = (x_0,y_0)$ beginnt und in $P = (x,y)$ endet, haben wir eine Drehung des Vektors $P_0P$ um den Punkt $P_0$. In diesem Fall lautet die Transformationsgleichung

$$\mathbf{r'} = R(\mathbf{r} - \mathbf{r_o}) + \mathbf{r_o}\ ,$$

denn es ist der Vektor $\mathbf{r} - \mathbf{r_o}$ , der sich um $P_0$ dreht.

Das folgende **Programm 5** zeigt diesen Fall.

**Programm 5:**

```
reset():Drehung eines Pfeils um P₀

fi:=35*PI/180:
x:=3:y:=5:
x0:=-2:y0:=2:

mat:=Dom::Matrix():export(linalg):
R:=mat([[cos(fi),-sin(fi)],[sin(fi),cos(fi)]]):
r:=mat([x,y]):
r0:=mat([x0,y0]):
r1:=R*(r-r0)+r0:
x1:=float(r1[1]);
y1:=float(r1[2]);

ar:=plot::Arrow2d([x0,y0],[x,y],Color=RGB::Blue):
ar1:=plot::Arrow2d([x0,y0],[x1,y1],Color=RGB::Red):

plot(ar,ar1,ViewingBox=[-6..6,0..8],
Scaling=Constrained)
```

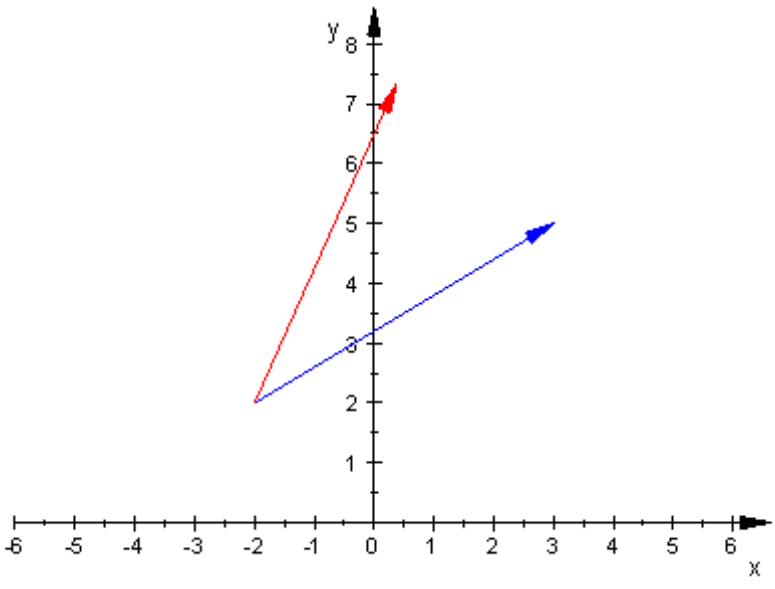



Fig. 3.6-4

Im folgenden **Programm 6** benutzen wir die Funktion **`plot::Scene2d`**, um die Transformation von drei Vektoren zu veranschaulichen.

Mit **`Layout = Vertical`** erhalten wir die Ergebnisse der Transformation senkrecht unter den Originalen.

**Programm 6:**

```
reset():
fi:=35*PI/180:
R := matrix([[cos(fi), -sin(fi)], [sin(fi),
cos(fi)]]):
x1 := plot::Arrow2d([0, 0], [3, 5], Color = RGB::Red):
x2 := plot::Arrow2d([0, 0], [-3, 1], Color =
RGB::Green):
x3 := plot::Arrow2d([0, 0], [2, -5], Color =
RGB::Blue):
plot(plot::Scene2d(x1, x2, x3),
plot::Scene2d(plot::Transform2d(R,x1,x2,x3),

Scaling = Constrained, Layout = Vertical));
```

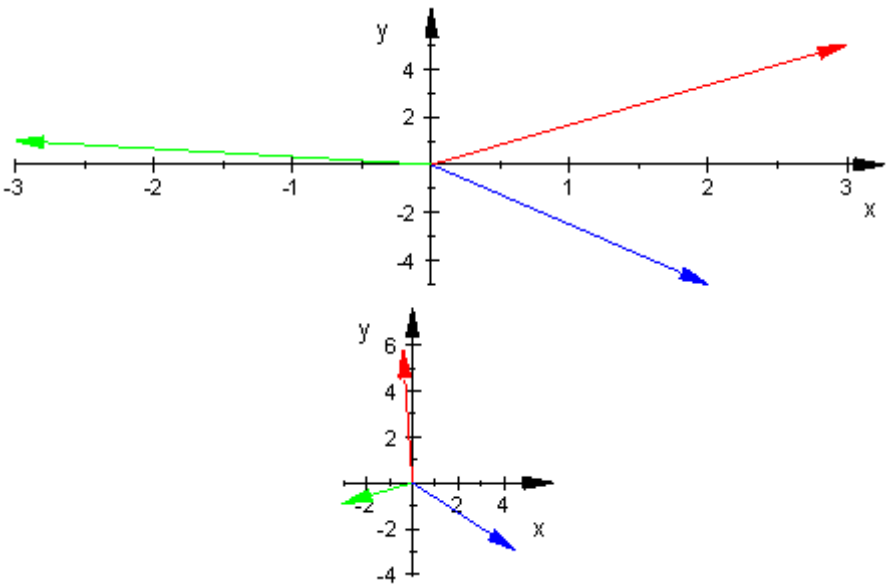



Fig. 3.6-5

Im letzten Programm sehen wir die Drehung eines Polygons mit Hilfe der Funktion **`plot::Polygon2d`**. Ohne diese Funktion müssten wir Punkte in den Graphen setzen und sie mit Linien verbinden. Informationen über diese Methode –und auch über Transformationen im Allgemeinen- kann man im Internet finden:

http://www.mupad.de/schule/literatur/index.shtml

**Programm 7:**

```
fi:=35*PI/180:
x:=3:y:=5:
R := matrix([[cos(fi), -sin(fi)], [sin(fi), cos(fi)]]):
ar := plot::Arrow2d([0, 0], [x, y], Color = RGB::Blue):
pol:=plot::Polygon2d([[0,0],[x,y],[x-5,y-3],[0,0]],Color =
RGB::Red):
pol1:= plot::Transform2d(R,pol,ar):
plot(ar,pol,pol1,Scaling = Constrained, Layout = Vertical);
```

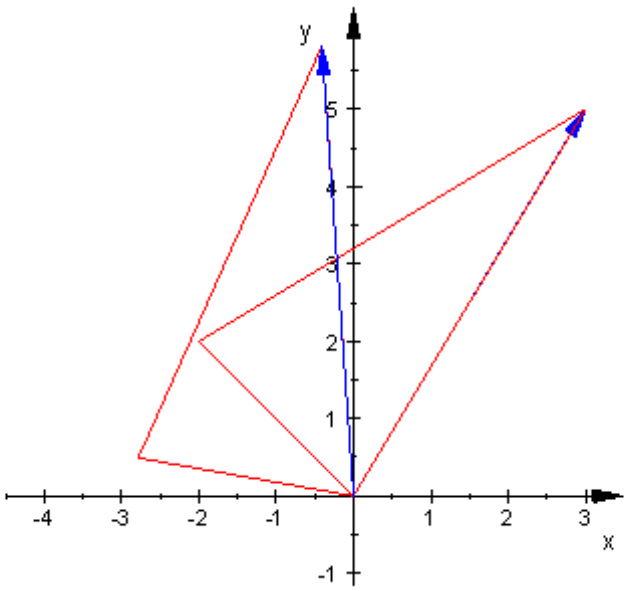



Fig. 3.6-6

## 3.6.3 Beschreibung von Drehungen mit Hilfe komplexer Zahlen.

Der komplexen Zahl $z = x + i\,y$ entspricht ein Punkt P in der festen komplexen Ebene C. Seine rechtwinkligen Koordinaten sind (x,y).

Da $x = r\cos\varphi$ , $y = r\sin\varphi$ und $r = (x^2 + y^2)^{1/2}$ , ergibt sich $z = x + i\,y = r(\cos\varphi + i\sin\varphi)$, und es ist $r = |z|$.

Im gedrehten System C' hat P die Darstellung $z' = x' + i\,y'$. Beide Zahlen haben denselben Betrag $r = (x^2 + y^2)^{1/2}$, aber das Argument (=Winkel) von z' ist um $\varphi$ Grad kleiner als das Argument von z. (Die Winkel haben positive Werte, wenn sie im Gegen-Uhrzeigersinn gemessen werden. Sofern nichts Anderes gesagt wird, werden sie in Radiant angegeben.)

$e^{i\varphi} = \cos\varphi + i\cdot\sin\varphi$ ist die Eulersche Formel (1707-1783), die wir als Definition von $e^{i\varphi}$ ansehen können. Mit dieser Definition können wir die komplexe Zahl **z** in Exponentialform schreiben (= Polarform): $z = r\cdot e^{i\varphi}$. Den Winkel $\varphi$ kann man als $\varphi = \arctan(y/x)$ berechnen.

Zwischen z und z' besteht also die Beziehung $z' = z\cdot e^{-i\varphi}$ oder $z = z'\cdot e^{i\varphi}$. In rechtwinkligen Koordinaten kann schreiben

$$x' + iy' = (x + i\,y)\cdot(\cos\varphi - i\sin\varphi) \quad (6)$$

Diese Gleichung enthält die beiden reelen Gleichungen (1):

$$\begin{aligned} x' &= x\cos\varphi + y\sin\varphi \\ y' &= -x\sin\varphi + y\cos\varphi \end{aligned} \quad (7)$$

Wenn man als Exponent in der Eulerformel $\varphi = \omega t$ wählt, kann man schreiben

$$e^{i\omega t} = \cos(\omega t) + i\,\mathrm{sen}(\omega t) \quad (8)$$

Die beiden ersten Ableitungen von $z = z'\cdot e^{i\omega t}$ lauten

$$dz/dt = dz'/dt\cdot e^{i\omega t} + i\omega z'\cdot e^{i\omega t}$$

$$d^2z/dt^2 = d^2z'/dt^2\cdot e^{i\omega t} + 2i\omega dz'/dt\cdot e^{i\omega t} - \omega^2 z'\cdot e^{i\omega t}$$

In reeller Schreibweise erhält ein Beobachter in C' die Beschleunigungen

$$d^2x'/dt^2 = a'_x + 2\omega\, dy'/dt + \omega^2 x'$$

$$d^2y'/dt^2 = a'_y - 2\omega\, dx'/dt + \omega^2 y' \qquad (9)$$

Das ist ein System von zwei gekoppelten Differentialgleichungen. $a'_x$ und $a'_y$ sind die Beschleunigungen infolge von Wechselwirkungen mit anderen Körpern, z.B mit dem Boden einer Plattform als Folge der Reibung.

Im folgenden Abschnitt wenden wir diese Gleichungen auf den Fall von Bewegungen auf einem Karussell an. Um die Schreibweise zu vereinfachen, verzichten wir in Zukunft auf die Akzente in den Gleichungen (9), wenn keine Gefahr zu Verwechslungen besteht.

Auch bei der Behandlung von Schwingungsbewegungen sind komplexe Zahlen von groβem Nutzen. Im folgenden Abschnit und im 6. Kapitel werden wir auf die komplexen Zahlen zurückkommen, um die Lösung von gewissen Differentialgleichungen vorzubereiten.

In http://de.wikibooks.org/wiki/Komplexe_Zahlen#2._Wechselstromrechnungen habe ich eine Einführung in komplexe Zahlen mit Beispielen gegeben. Vielleicht finden Sie diese Darstellung nützlich.

## 3.6.4 Komplexe Zahlen in MuPAD

MuPAD erlaubt auch das Arbeiten mit komplexen Zahlen, aber achten Sie darauf, dass die Einheit **i** als Majuskel geschrieben werden muss: $(-1)^{1/2}$ = **I**.

Schauen wir uns zunächst einige einfache Beispiele an: Addition, Subtraktion, Multiplikation und Division. Den Betrag von z berechnet man mit **abs(z)**, das Argument mit **arg(z),** und die konjugiert-komplexe Zahl mit **conjugate(z)**.

```
reset():
grau:=180/PI:
z1:=-0.5-0.866*I:
z2:=-1 +I:
z1+z2:
z1-z2:
p:=z1*z2;
q:=z1/z2;
float(arg(z1)*grau);
float(arg(z2)*grau);
abs(z1),abs(z2),abs(p),abs(q)
```

```
1.366 + 0.366 I
- 0.183 + 0.683 I
-120.0007278
135.0
1/2

0.9999779998, 2 , 1.414182449, 0.7070912247
```

Die Funktion **solve** ist auch dann verwendbar, wenn die Lösungsmenge einer **algebraischen Gleichung** nur komplexe Zahlen enthält. Verleichen Sie die Beispiele:

```
solve(8*x^2+12*x+10=0,x)
```

$$\left\{-\frac{i}{4}\cdot\sqrt{11}-\frac{3}{4},\frac{i}{4}\cdot\sqrt{11}-\frac{3}{4}\right\}$$

Wenn man versucht, die Lösungsmenge der folgenden Gleichung im Bereich der reellen Zahlen zu finden, erhält man als Antwort die leere Menge, d.h. S = Ø = { }:

```
assume(x,Type::Real):
solve(x^2-3*x+7=0,x)
```

$\varnothing$

Sucht man die Lösung aber unter den komplexen Zahlen, erhält man

```
assume(x,Type::Complex):
solve(x^2-3*x+7=0,x)
```

$$\left\{\frac{3}{2}-\frac{i}{2}\cdot\sqrt{19},\ \frac{i}{2}\cdot\sqrt{19}+\frac{3}{2}\right\}$$

Im folgenden Fall erhalten wir die vollständige Lösungsmenge:

- `solve(6*z^4-25*z^3+32*z^2+3*z-10=0,z)`

$$\left\{-\frac{1}{2},\ \frac{2}{3},\ 2-i,\ 2+i\right\}$$

Wenn wir nur die reellen Lösungen sehen wollen, müssen wir schreiben:

```
assume(z,Type::Real):
solve(6*z^4-25*z^3+32*z^2+3*z-10=0,z)
```

$$\left\{-\frac{1}{2},\ \frac{2}{3}\right\}$$

Um die Polarform einer komplexen Zahl z = x + i y zu finden, benutzen wir die Beziehungen $z = r\cdot e^{i\varphi}$, mit $r = (x^2 + y^2)^{1/2}$ , und φ = arctan(y/x) oder φ = arg(z):

**Beispiele:**

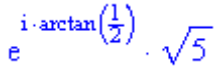
- ```
  z:=2+2*sqrt(3)*I:
  x:=Re(z):y:=Im(z):
  arctan(y/x);
  arg(z)
  ```

$$\frac{\pi}{3}\quad\frac{\pi}{3}$$

- ```
  z:=-3-4*I:
  abs(z)*exp(I*arg(z))
  ```

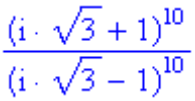
$$-5\cdot e^{i\cdot\arctan\left(\frac{4}{3}\right)}$$

- **z:=2+I:**
  **abs(z)*exp(I*arg(z))**

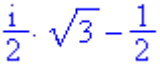


Schauen Sie sich auch das folgende Beispiel an, in dem wir die Funktion **rectform** einsetzen, die immer die Form z = x + i y liefert, also mit rechtwinkligen Koordinaten. Die Funktionen **simplify** und **Simplify** haben in diesem Fall dieselbe Wirkung. Die neue Funktion **Simplify** scheint jedoch im Allgemeinen wirksamer zu sein als **simplify.**

- **((1+sqrt(3)*I)/(1-sqrt(3)*I))^10**



- **Simplify(%)**



- **((1+sqrt(3)*I)/(1-sqrt(3)*I))^10**

**rectform(%)**