3.6 Descrição de Rotações no Plano

3.6.1 Matriz de Rotação

Às vezes temos que descrever a posição de uma partícula num referencial *plano* que é girado de um ângulo φ com respeito a outro sistema fixo.

Na álgebra linear aprendemos que uma rotação pode ser descrita por uma transformação linear: **r'** = R·**r**. O nosso primeiro objetivo é determinar os elementos da matriz do operador de rotação R.

No sistema de coordenadas fixo C(O,**i**,**j**), o ponto P tem as coordenadas (x,y). No sistema C'(O,**i'**,**j'**) as coordenadas do mesmo ponto são (x',y'). O sistema C', girado de um ângulo φ, tem a mesma origem que C. φ indica o βngulo que o semi-eixo positivo x' forma com o semi-eixo positivo x.

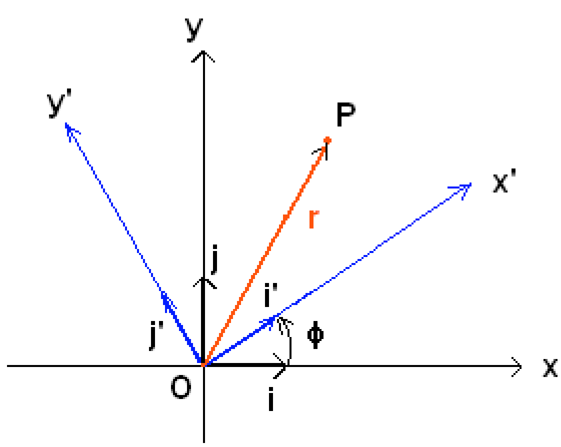

Fig. 3.6-1

A representação matricial de R encontramos por meio dos produtos escalares dos vetores unitários:

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

Agora, **r** = x·**i** + y·**j** = x'·**i'** + y'·**j'**. Multiplicando **r** por **i'** e **j'**, obtemos x' = **i'·r** e y' = **j'·r** e substituindo **r** por x·**i** + y·**j**, teremos x' = **i'** ·(x **i** + y·**j**) = x·**i'·i** + y·**i'·j** = x·cos φ + y·sen φ e y' = **j'** ·**r** = **j'**·(x **i** + y·**j**) = x·(-sen φ) + y·cos φ. Temos, entyo :

$$x' = x \cdot \cos\varphi + y \cdot \operatorname{sen}\varphi$$

$$y' = x \cdot (-\operatorname{sen}\varphi) + y \cdot \cos\varphi \qquad (1)$$

Este resultado podemos escrever em forma matricial:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\varphi & \operatorname{sen}\varphi \\ -\operatorname{sen}\varphi & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \qquad (2)$$

A matriz de rotação R tem, então, a forma:

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

Analogamente podemos calcular (x,y) em função dos (x',y'): Multiplicando **r** por **i** e **j** e substituindo **r** por x'·**i'** + y'·**j'**, temos

$$x = x' \cdot \cos\varphi + y' \cdot (-\operatorname{sen}\varphi) \text{ e } y = x' \cdot \operatorname{sen}\varphi + y' \cdot \cos\varphi \qquad (4)$$

Outra maneira para chegar as relações (4), seria a solução das equações (1) com respeito a x e y. É possível fazer isso a mão, o, se isso for difícil de mais, com ajuda de MuPAD:

Programa 1:

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

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

Também pode-se usar a matriz inversa na equação **r** = R^{-1}·**r'**:

Programa 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 Aplicações

Nas aplicações queremos muitas vezes girar o ponto P e manter o sistema de coordenadas fixo. Em tal caso temos x' = r cos(α+φ) = r cos α cos φ - r sen α sen φ, sendo α o βngulo que o vetor **r** faz com o eixo-x. O vetor **r'** faz o ângulo α + φ com este eixo. Mas r cosα = x e r senα = y. Então : x' = x cosφ - y senφ.

Analogamente: y' = r sen(α +φ) = y cos φ + x sen φ. A matriz de rotaηão de um ponto P de um ângulo φ -no sentido anti-horário- é, então,

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

(A rotação anti-horária de um vetor-posição corresponde a uma rotação do sistema de coordenadas no sentido horário. R_P é a matriz inversa da (2) e é obtida, neste caso, substituindo φ por -φ.)

Com o **Programa 3** podemos ilustrar a rotação do ponto (x,y) de um ângulo φ, no caso 35° anti-horário, -note também a aplicação de **`ViewingBox`**:

Programa 3:

```
• reset()://rotação de um ponto ou uma seta em torno de (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://equação da transformação
  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
```

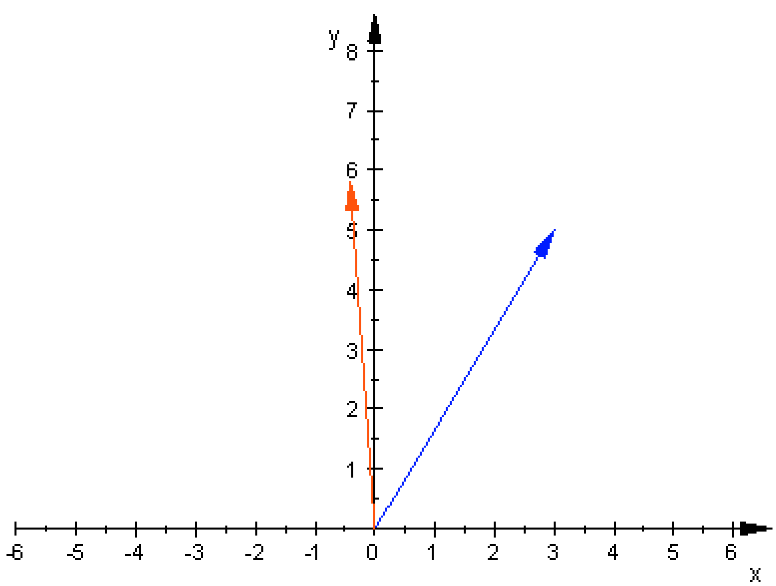

Fig. 3.6-2

MuPAD tem a função **Transform2d** que simplifica um pouco o programa 3 e que é bastante útil quando se quer transformar *varios* objetos, utilizando a função **plot::Scene2d**. No próximo programa 4 fazemos uso de **Transform2d**. Desafortunadamente, não é possível usar a função **Color=RGB** junto com a instrução **Transform2d**, o que tem por consequência que o objeto transformado é representado da mesma cor que o original, no caso, vermelho.

Programa 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);
```

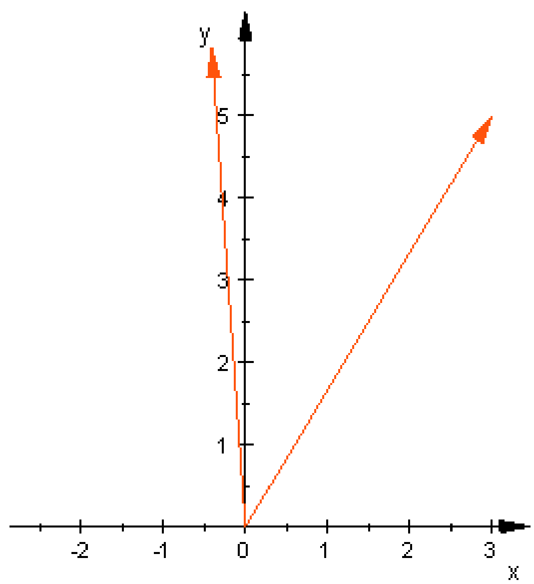

Fig. 3.6-3

Com a instrução `ar1::Matrix2d(R)` podemos ver a forma da matriz R assim como com `a:=float(R).` Mas, a forma do resultado produzido por `float(R)` é muito mais intuitivo. Isso é fortemente ilustrado no caso dos 30 grados, sem uso de `float`.

- ```
  a:=float(R);
  a1:=ar1::Matrix2d(R)
  ```

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

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

Rotaçaõ de 30^o:

- ```
  a:=R;
  a1:=ar1::Matrix2d(R)
  ```

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

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

Se o vetor **r** começa no ponto $P_0 = (x_0,y_0)$ e termina em P = (x,y), temos uma rotação do vetor P_0P em torno do ponto P_0. A transformação se escreve, neste caso, como

$\mathbf{r'} = R(\mathbf{r} - \mathbf{r_o}) + \mathbf{r_o}$, pois é o vetor $\mathbf{r} - \mathbf{r_o}$ o que gira ao redor de P_0.

O programa 5 mostra este caso.

Programa 5:

- ```
  reset()://rotação de uma seta ao redor de 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)
```

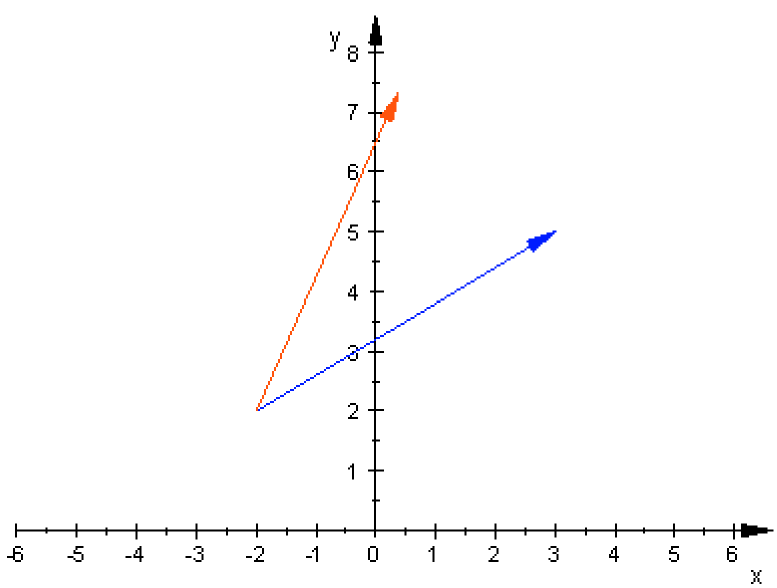

Fig. 3.6-4

No seguinte Programa 6 fazemos uso da função `plot::Scene2d`, para representar a transformação de três vetores.

Com `Layout = Vertical` obtemos os resultados da transformação verticalmente debaixo do gráfico dos vetores originais.

Programa 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));
```

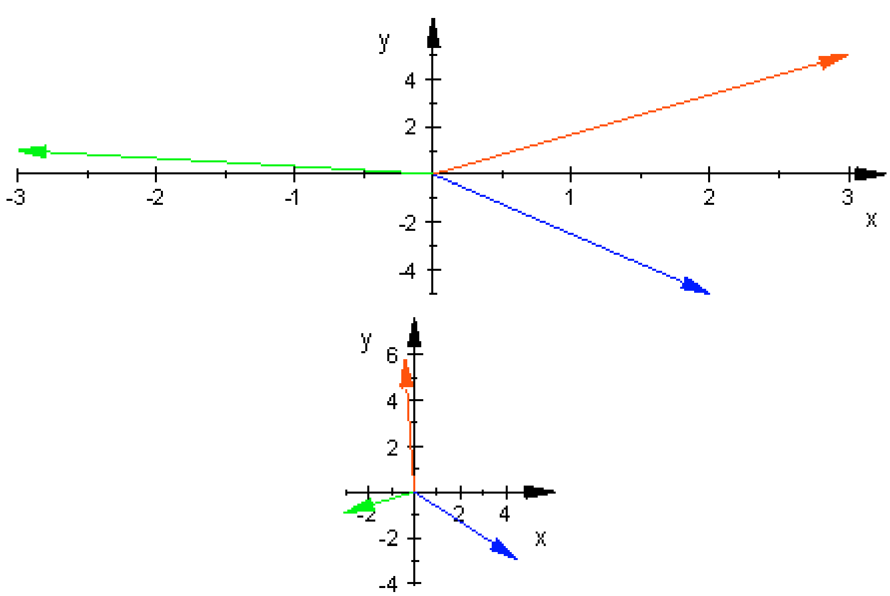

Fig. 3.6-5

No último programa vemos a rotação de um polígono por meio da função **plot::Polygon2d**. Sem esta função, deveriamos colocar pontos no gráfico e conectá-los por meio de linhas. Informação sobre este procedimento, e também sobre transformações em geral, pode-s obter no site:

http://www.mupad.de/schule/literatur/index.shtml, -infelizmente em alemão.

Programa 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);
```

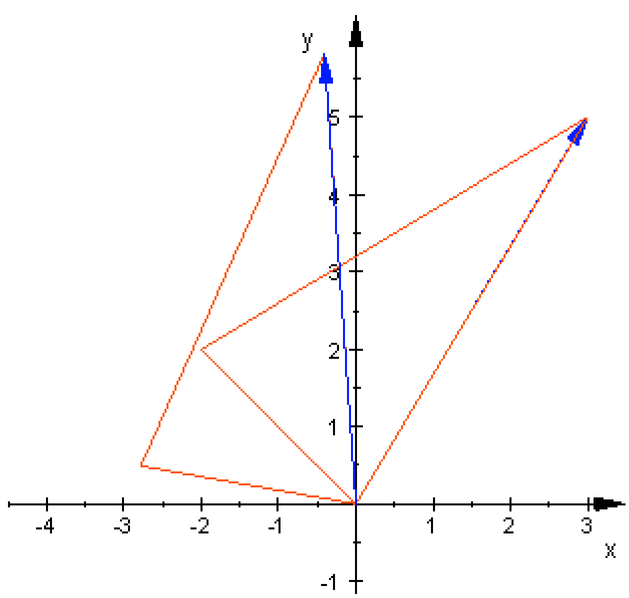

Fig. 3.6-6

3.6.3 Descrição de rotações por meio de números complexos.

Os números complexos proporcionam outra maneira de descrever uma rotação no plano, pois este pode ser considerado como o plano dos números complexos. Ao número complexo $z = x + i\,y$ corresponde o ponto P, tendo este os coordenadas retangulares (x,y) no sistema fixo C. Sendo $x = r\cos\varphi$, $y = r\,\mathrm{sen}\,\varphi$ e $r = (x^2 + y^2)^{1/2}$, resulta $z = x + i\,y = r(\cos\varphi + i\,\mathrm{sen}\,\varphi)$. $r = |z|$ ı o módulo de z.

No sistema girado C', a representação de P é $z' = x' + i\,y'$. Ambos os números têm o mesmo módulo $r = (x^2 + y^2)^{1/2}$, mas, o argumento (= ângulo) de z' é de φ graus menor do que o argumento de z. (Os ângulos terão valores positivos quando medidos no sentido anti-horário, e serão expressos em radianos -a menos de especificação em contrário.)

$e^{i\varphi} = \cos\varphi + i\cdot\mathrm{sen}\,\varphi$ é a fórmula de Euler (1707-1783) que podemos considerar como definição de $e^{i\varphi}$. Com esta definição podemos escrever o número complexo **z** na forma exponential (forma polar): $z = r\cdot e^{i\varphi}$. O ângulo φ podemos calcular como $\varphi = \arctan(y/x)$, arco tangente de (y/x).

Entre z e z' existe, então, a relação $z' = z\cdot e^{-i\varphi}$ ou $z = z'\cdot e^{i\varphi}$. Usando coordenadas retangulares, escreve-se

$$x' + iy' = (x + i\,y)\cdot(\cos\varphi - i\,\mathrm{sen}\,\varphi) \qquad (6)$$

Esta equação contem as duas equações reais (1):

$$\begin{aligned} x' &= x\cos\varphi + y\,\mathrm{sen}\,\varphi \\ y' &= -x\,\mathrm{sen}\,\varphi + y\cos\varphi \end{aligned} \qquad (7)$$

Com uma mudança do expoente na fórmula de Euler, por exemplo $\varphi = \omega t$, pode-se escrever

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

As derivadas primeira e segunda de $z = z'\cdot e^{i\omega t}$ são

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

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

Em notação real, obtém-se, para um observador no sistema C', as acelerações

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

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

Isso é um sistema de duas equações diferenciais acopladas.
a'_x e a'_y são as acelerações devidas a interações com outros corpos, por exemplo, com o chão de uma plataforma por meio do atrito.

Na próxima seção aplicaremos estas equações no caso do movimento numa plataforma girante (carrossel). Para simplificar a escrita, escrevemos as equações (9) no futuro sem acentos, se não existe perigo de confusão.

Os números complexos são, também, de grande utilidade na descrição de movimentos oscilatórios. Na próxima seção e no capítulo 6 vamos voltar aos números complexos para facilitar o trabalho na busca de soluções de certas equações diferenciais.

3.6.4 Números complexos com MuPAD

MuPAD aceita também os números complexos, mas, note, que a unidade imaginária **i** deve ser escrita em forma de maiúscula: $(-1)^{1/2}$ = **I**.

Veja primeiro alguns exemplos básicos: adição, subtração, multiplicação e divisão. O módulo de z calcula-se com `abs(z)`, o argumento com `arg(z)` e o conjugado de z com `conjugate(z)`.

Você pode estudar as noções básicas sobre números complexos no seguinte site:

http://pessoal.sercomtel.com.br/matematica/medio/213/ncomplex.htm

```
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
```

A função **solve** funciona também quando o conjunto solução de uma **equação algébrica** consta somente de números complexos. Veja os seguintes exemplos:

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

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

Ao tentar obter o conjunto solução para esta equação sobre o conjunto dos números reais, obteremos como resposta o conjunto vazio, isto é $S = \emptyset = \{\ \}$:

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

$\emptyset$

Se buscarmos o conjunto solução sobre o conjunto dos números complexos, obtemos

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

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

No próximo caso obtemos o conjunto solução completo:

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

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

Se querermos só as soluções reais, temos que escrever:

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

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

Para calcular a forma polar de um número complexo z = x + i y, utilizamos as relações $z = r\cdot e^{i\varphi}$, onde $r = (x^2 + y^2)^{1/2}$ e $\varphi = \arctan(y/x)$ ou $\varphi = \arg(z)$:

Exemplos:

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

$$\frac{\pi}{3} \qquad \frac{\pi}{3}$$

- ```
  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))

$$e^{i \cdot \arctan\left(\frac{1}{2}\right)} \cdot \sqrt{5}$$

Veja também o seguinte exemplo, onde usamos a função **rectform** que proporciona sempre a forma z = x + y i em coordenadas retangulares -daí o nome. As funciones **simplify** e **Simplify** têm, neste caso, o mesmo efeito. A nova função **Simplify** parece ser, em geral, mais potente do que a velha **simplify.**

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

$$\frac{\left(i \cdot \sqrt{3}+1\right)^{10}}{\left(i \cdot \sqrt{3}-1\right)^{10}}$$

- **Simplify(%)**

$$\frac{i}{2} \cdot \sqrt{3}-\frac{1}{2}$$

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

rectform(%)

$$-\frac{1}{2}+i \cdot \frac{\sqrt{3}}{2}$$