This matrix has a zero diagonal and is an example of a 2×2 skew-symmetric matrix. The matrix has only one unique element and we can unpack it using the Toolbox function `vex`

```
>> vex(S)
ans =
    0.3000
```

to recover the rotation angle.

The inverse of a logarithm is exponentiation and using the builtin MATLAB matrix exponential function `expm`

`expm` is different to the builtin function `exp` which computes the exponential of each element of the matrix. $\text{expm}(A) = I + A + A^2/2! + A^3/3! + \cdots$

```
>> expm(S)
ans =
    0.9553   -0.2955
    0.2955    0.9553
```

the result is, as expected, our original rotation matrix. In fact the command

```
>> R = rot2(0.3);
```

is equivalent to

```
>> R = expm(  skew(0.3)  );
```

Formally we can write

$$\boldsymbol{R} = \mathrm{e}^{[\theta]_\times} \in \mathbf{SO}(2)$$

where θ is the rotation angle, and the notation $[\cdot]_\times : \mathbb{R} \mapsto \mathbb{R}^{2\times 2}$ indicates a mapping from a scalar to a skew-symmetric matrix.

2.1.2 Pose in 2-Dimensions

2.1.2.1 Homogeneous Transformation Matrix

Now we need to account for the translation between the origins of the frames shown in Fig. 2.6. Since the axes $\{V\}$ and $\{A\}$ are parallel, as shown in Figs. 2.6 and 2.7, this is simply vectorial addition

$$\begin{pmatrix} {}^A x \\ {}^A y \end{pmatrix} = \begin{pmatrix} {}^V x \\ {}^V y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.8}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} {}^B x \\ {}^B y \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.9}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{pmatrix} \begin{pmatrix} {}^B x \\ {}^B y \\ 1 \end{pmatrix} \tag{2.10}$$

or more compactly as

$$\begin{pmatrix} {}^A x \\ {}^A y \\ 1 \end{pmatrix} = \begin{pmatrix} {}^A\boldsymbol{R}_B & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix} \begin{pmatrix} {}^B x \\ {}^B y \\ 1 \end{pmatrix} \tag{2.11}$$

where $\boldsymbol{t} = (x, y)$ is the translation of the frame and the orientation is ${}^A\boldsymbol{R}_B$. Note that ${}^A\boldsymbol{R}_B = {}^V\boldsymbol{R}_B$ since the axes of frames $\{A\}$ and $\{V\}$ are parallel. The coordinate vectors for point $\mathbf{P}$ are now expressed in homogeneous form and we write

A vector $\boldsymbol{p} = (x, y)$ is written in homogeneous form as $\tilde{\boldsymbol{p}} \in \mathbb{P}^2$, $\tilde{\boldsymbol{p}} = (x_1, x_2, x_3)$ where $x = x_1 / x_3$, $y = x_2 / x_3$ and $x_3 \neq 0$. The dimension has been increased by one and a point on a plane is now represented by a 3-vector. To convert a point to homogeneous form we typically append an element equal to one $\tilde{\boldsymbol{p}} = (x, y, 1)$. The tilde indicates the vector is homogeneous.

Homogeneous vectors have the important property that $\tilde{\boldsymbol{p}}$ is equivalent to $\lambda \tilde{\boldsymbol{p}}$ for all $\lambda \neq 0$ which we write as $\tilde{\boldsymbol{p}} \simeq \lambda \tilde{\boldsymbol{p}}$. That is $\tilde{\boldsymbol{p}}$ represents the same point in the plane irrespective of the overall scaling factor. Homogeneous representation is important for computer vision that we discuss in Part IV. Additional details are provided in Sect. C.2.

$$
\begin{aligned}
{}^A\tilde{\boldsymbol{p}} &= \begin{pmatrix} {}^A\boldsymbol{R}_B & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix} {}^B\tilde{\boldsymbol{p}} \\
&= {}^A\boldsymbol{T}_B \, {}^B\tilde{\boldsymbol{p}}
\end{aligned}
$$

and ${}^A\boldsymbol{T}_B$ is referred to as a homogeneous transformation. The matrix has a very specific structure and belongs to the special Euclidean group of dimension 2 or $\boldsymbol{T} \in \mathbf{SE}(2) \subset \mathbb{R}^{3\times 3}$.

By comparison with Eq. 2.1 it is clear that ${}^A\boldsymbol{T}_B$ represents translation and orientation or relative pose. This is often referred to as a *rigid-body motion*.

$$
\boldsymbol{T} = \begin{pmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix}
$$

A concrete representation of relative pose ξ is $\xi \sim \boldsymbol{T} \in \mathbf{SE}(2)$ and $\boldsymbol{T}_1 \oplus \boldsymbol{T}_2 \mapsto \boldsymbol{T}_1 \boldsymbol{T}_2$ which is standard matrix multiplication

$$
\boldsymbol{T}_1 \boldsymbol{T}_2 = \begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{t}_1 \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{R}_2 & \boldsymbol{t}_2 \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{R}_1 \boldsymbol{R}_2 & \boldsymbol{t}_1 + \boldsymbol{R}_1 \boldsymbol{t}_2 \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix}
$$

One of the algebraic rules from page 21 is $\xi \oplus 0 = \xi$. For matrices we know that $\boldsymbol{T}\boldsymbol{I} = \boldsymbol{T}$, where $\boldsymbol{I}$ is the identify matrix, so for pose $0 \mapsto \boldsymbol{I}$ the identity matrix. Another rule was that $\xi \ominus \xi = 0$. We know for matrices that $\boldsymbol{T}\boldsymbol{T}^{-1} = \boldsymbol{I}$ which implies that $\ominus \boldsymbol{T} \mapsto \boldsymbol{T}^{-1}$

$$
\boldsymbol{T}^{-1} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{R}^T & -\boldsymbol{R}^T \boldsymbol{t} \\ \boldsymbol{0}_{1\times 2} & 1 \end{pmatrix}
$$

For a point described by $\tilde{\boldsymbol{p}} \in \mathbb{P}^2$ then $\boldsymbol{T} \cdot \tilde{\boldsymbol{p}} \mapsto \boldsymbol{T}\tilde{\boldsymbol{p}}$ which is a standard matrix-vector product.

To make this more tangible we will show some numerical examples using MATLAB and the Toolbox. We create a homogeneous transformation which represents a translation of (1, 2) followed by a rotation of 30°

```
>> T1 = transl2(1, 2) * trot2(30, 'deg')
T1 =
    0.8660   -0.5000    1.0000
    0.5000    0.8660    2.0000
         0         0    1.0000
```

Many Toolbox functions have variants that return orthonormal rotation matrices or homogeneous transformations, for example, `rot2` and `trot2`.

The function `transl2` creates a relative pose with a finite translation but zero rotation, while `trot2` creates a relative pose with a finite rotation but zero translation. We can plot this, relative to the world coordinate frame, by

```
>> plotvol([0 5 0 5]);
>> trplot2(T1, 'frame', '1', 'color', 'b')
```

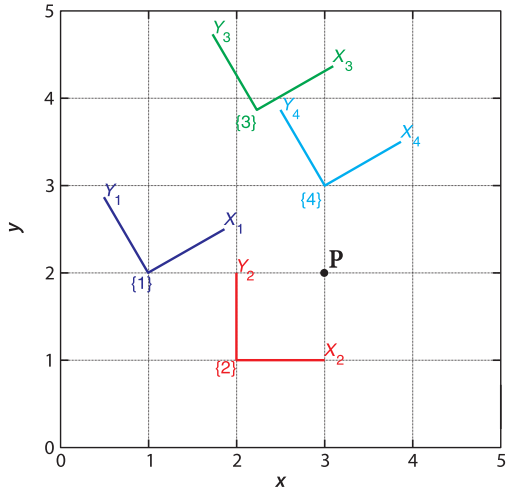

Fig. 2.8.
Coordinate frames drawn using the Toolbox function `trplot2`

The options specify that the label for the frame is {1} and it is colored blue and this is shown in Fig. 2.8. We create another relative pose which is a displacement of (2, 1) and zero rotation

```
>> T2 = transl2(2, 1)
T2 =
     1     0     2
     0     1     1
     0     0     1
```

which we plot in red

```
>> trplot2(T2, 'frame', '2', 'color', 'r');
```

Now we can compose the two relative poses

```
>> T3 = T1*T2
T3 =
    0.8660   -0.5000    2.2321
    0.5000    0.8660    3.8660
         0         0    1.0000
```

and plot it, in green, as

```
>> trplot2(T3, 'frame', '3', 'color', 'g');
```

We see that the displacement of (2, 1) has been applied with respect to frame {1}. It is important to note that our final displacement is not (3, 3) because the displacement is with respect to the rotated coordinate frame. The noncommutativity of composition is clearly demonstrated by

```
>> T4 = T2*T1;
>> trplot2(T4, 'frame', '4', 'color', 'c');
```

and we see that frame {4} is different to frame {3}.

Now we define a point (3, 2) relative to the world frame

```
>> P = [3 ; 2 ];
```

which is a column vector and add it to the plot

```
>> plot_point(P, 'label', 'P', 'solid', 'ko');
```

To determine the coordinate of the point with respect to {1} we use Eq. 2.1 and write down

$${}^0\boldsymbol{p} = {}^0\xi_1 \bullet {}^1\boldsymbol{p}$$

and then rearrange as

$$
\begin{aligned}
{}^{1}\boldsymbol{p} &= {}^{1}\xi_{0} \bullet {}^{0}\boldsymbol{p} \\
&= \left({}^{0}\xi_{1}\right)^{-1} \bullet {}^{0}\boldsymbol{p}
\end{aligned}
$$

Substituting numerical values

```
>> P1 = inv(T1) * [P; 1]
P1 =
    1.7321
   -1.0000
    1.0000
```

where we first converted the Euclidean point coordinates to *homogeneous form* by appending a one. The result is also in homogeneous form and has a negative y-coordinate in frame {1}. Using the Toolbox we could also have expressed this as

```
>> h2e( inv(T1) * e2h(P) )
ans =
    1.7321
   -1.0000
```

where the result is in Euclidean coordinates. The helper function `e2h` converts Euclidean coordinates to homogeneous and `h2e` performs the inverse conversion.

2.1.2.2 Centers of Rotation

We will explore the noncommutativity property in more depth and illustrate with the example of a pure rotation. First we create and plot a reference coordinate frame {0} and a target frame {X}

```
>> plotvol([-5 4 -1 5]);
>> T0 = eye(3,3);
>> trplot2(T0, 'frame', '0');
>> X = transl2(2, 3);
>> trplot2(X, 'frame', 'X');
```

and create a rotation of 2 radians (approximately 115°)

```
>> R = trot2(2);
```

and plot the effect of the two possible orders of composition

```
>> trplot2(R*X, 'framelabel', 'RX', 'color', 'r');
>> trplot2(X*R, 'framelabel', 'XR', 'color', 'r');
```

The results are shown as red coordinate frames in Fig. 2.9. We see that the frame {RX} has been rotated about the origin, while frame {XR} has been rotated about the origin of {X}.

What if we wished to rotate a coordinate frame about an arbitrary point? First of all we will establish a new point **C** and display it

```
>> C = [1 2]';
>> plot_point(C, 'label', ' C', 'solid', 'ko')
```

and then compute a transform to rotate about point **C**

```
>> RC = transl2(C) * R * transl2(-C)
RC =
   -0.4161   -0.9093    3.2347
    0.9093   -0.4161    1.9230
         0         0    1.0000
```

and applying this

```
>> trplot2(RC*X, 'framelabel', 'XC', 'color', 'r');
```

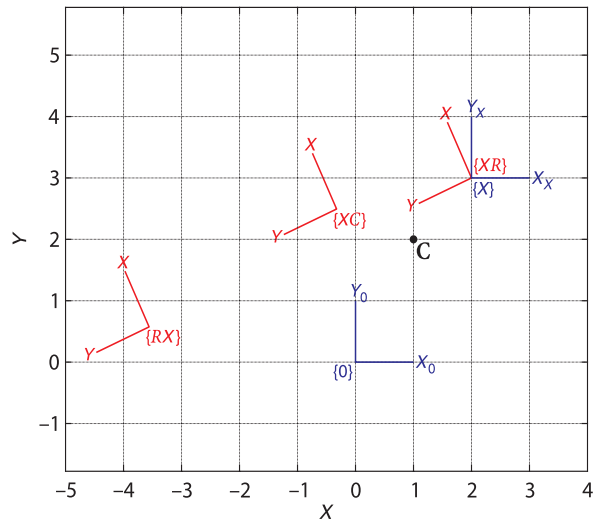

Fig. 2.9.
The frame $\{X\}$ is rotated by 2 radians about $\{0\}$ to give frame $\{RX\}$, about $\{X\}$ to give $\{XR\}$, and about point **C** to give frame $\{XC\}$

we see that the frame has indeed been rotated about point **C**. Creating the required transform was somewhat cumbersome and not immediately obvious. Reading from right to left▸ we first apply an origin shift, a translation from **C** to the origin of the reference frame, apply the rotation about that origin, and then apply the inverse origin shift, a translation from the reference frame origin back to **C**. A more descriptive way to achieve this is using twists.

`RC` left multiplies `X`, therefore the first transform applied to `X` is `transl(-C)`, then `R`, then `transl(C)`.

2.1.2.3 Twists in 2D

The corollary to what we showed in the last section is that, given any two frames we can find a rotational center that will *rotate* the first frame into the second. For the case of pure translational motion the rotational center will be at infinity. This is the key concept behind what is called a twist.

We can create a rotational twist about the point specified by the coordinate vector `C`

```
>> tw = Twist('R', C)
tw =
( 2 -1; 1 )
```

and the result is a `Twist` object that encodes a twist vector with two components: a 2-vector *moment* and a 1-vector *rotation*. The first argument `'R'` indicates a rotational twist is to be computed. This particular twist is a *unit twist* since the magnitude of the rotation, the last element of the twist, is equal to one.

To create an **SE**(2) transformation for a rotation about this unit twist by 2 radians we use the `T` method

```
>> tw.T(2)
ans =
   -0.4161   -0.9093    3.2347
    0.9093   -0.4161    1.9230
         0         0    1.0000
```

which is the same as that computed in the previous section, but more concisely specified in terms of the center of rotation. The center is also called the pole of the transformation and is encoded in the twist

```
>> tw.pole'
ans =
     1     2
```

For a unit-translational twist the rotation is zero and the moment is a unit vector.

If we wish to perform translational motion in the direction (1, 1) the relevant unit twist is

```
>> tw = Twist('T', [1 1])
tw =
( 0.70711 0.70711; 0 )
```

and for a displacement of $\sqrt{2}$ in the direction defined by this twist the SE(2) transformation is

```
>> tw.T(sqrt(2))
ans =
     1     0     1
     0     1     1
     0     0     1
```

which we see has a null rotation and a translation of 1 in the x- and y-directions.

For an arbitrary planar transform such as

```
>> T = transl2(2, 3) * trot2(0.5)
T =
    0.8776   -0.4794    2.0000
    0.4794    0.8776    3.0000
         0         0    1.0000
```

we can compute the twist vector

```
>> tw = Twist(T)
tw =
( 2.7082 2.4372; 0.5 )
```

and we note that the last element, the rotation, is not equal to one but is the required rotation angle of 0.5 radians. This is a nonunit twist. Therefore when we convert this to an SE(2) transform we don't need to provide a second argument since it is implicit in the twist

```
>>  tw.T
ans =
    0.8776   -0.4794    2.0000
    0.4794    0.8776    3.0000
         0         0    1.0000
```

and we have regenerated our original homogeneous transformation.

2.2 Working in Three Dimensions (3D)

The 3-dimensional case is an extension of the 2-dimensional case discussed in the previous section. We add an extra coordinate axis, typically denoted by z, that is orthogonal to both the x- and y-axes. The direction of the z-axis obeys the *right-hand rule* and forms a *right-handed coordinate frame*. Unit vectors parallel to the axes are denoted $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{z}}$ such that

In all these identities, the symbols from left to right (across the equals sign) are a cyclic rotation of the sequence *xyz*.

$$\hat{\boldsymbol{z}} = \hat{\boldsymbol{x}} \times \hat{\boldsymbol{y}}, \quad \hat{\boldsymbol{x}} = \hat{\boldsymbol{y}} \times \hat{\boldsymbol{z}}; \quad \hat{\boldsymbol{y}} = \hat{\boldsymbol{z}} \times \hat{\boldsymbol{x}} \tag{2.12}$$

A point **P** is represented by its x-, y- and z-coordinates (x, y, z) or as a bound vector

$$\boldsymbol{p} = x\hat{\boldsymbol{x}} + y\hat{\boldsymbol{y}} + z\hat{\boldsymbol{z}}$$

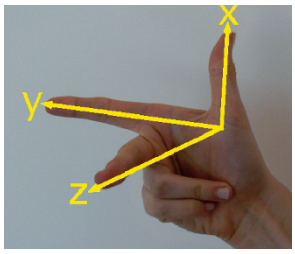

Figure 2.10 shows a red coordinate frame $\{B\}$ that we wish to describe with respect to the blue reference frame $\{A\}$. We can see clearly that the origin of $\{B\}$ has been

Right-hand rule. A right-handed coordinate frame is defined by the first three fingers of your right hand which indicate the relative directions of the x-, y- and z-axes respectively.