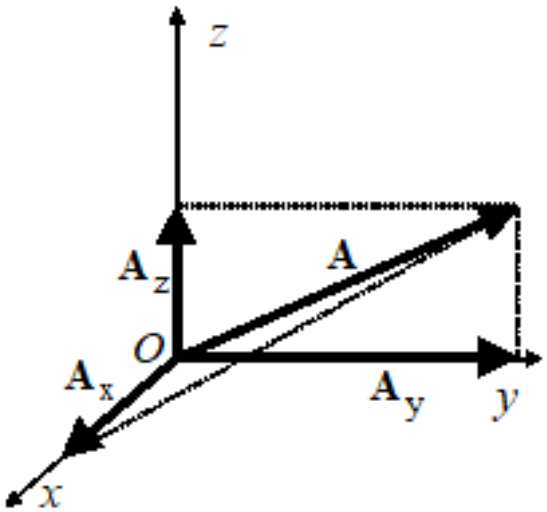
Vectors, Lists, Matrices

Vectors and Lists

Physical definition of vector is an arrow having length and direction. Many physical quantities are vectors, such as position **r**, velocity **v**, angular momentum **L**, etc. Vector can be defined by its components that are projections of the vector onto the axes of a coordinate system. Components of vectors is what is actually used in operations on them. Vectors, in general, can have any number of components.



Vector in terms of its components (x,y,z = 1,2,3) can be written as

$\mathbf{A} = A_1\,\mathbf{e}_1 + A_2\,\mathbf{e}_2 + A_3\,\mathbf{e}_3$,

where $\mathbf{e}_1$ is a unit vector (vector of length 1) directed along the x-axis etc.

Components of vectors can be stored as Lists. Lists in *Mathematica* (called arrays in some other systems) store elements within a single mathematical object, and many operations can be performed on Lists as the whole rather than performing operations on individual elements. This saves a lot of computer time and is a big advantage of *Mathematica* and similar products. One always has to try to formulate many-variable problems in the vector form, i.e., in form of Lists, gaining in elegance and speed.

The simplest form of a List is

```
MyList = {0, 4, a, x, f[x], image, sound, list, whatever}
```

Elements of a List can be any objects. In particular, one can write 3d vectors as

```
Avec = {A1, A2, A3};
Bvec = {B1, B2, B3};
```

Vector components (list elements) can be obtained from Lists as follows

```
Avec[[1]]
```

A_1

```
Avec[[2]]
```

A_2

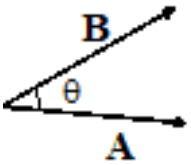
etc.

Products of Vectors

One can define dot product and cross product of two vectors.

Dot product

The dot product of **A** and **B** is a scalar defined by



$$\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}|\,|\mathbf{B}|\,\text{Cos}[\theta] = AB\,\text{Cos}[\theta]$$

In components, the dot product can be written as

$$\mathbf{A} \cdot \mathbf{B} = A_1 B_1 + A_2 B_2 + A_3 B_3 = \sum_{\alpha=1}^{3} A_\alpha B_\alpha$$

Excercise: Prove that the two definitions above are the same.

The dot product is implemented in Mathematica as

```
Avec.Bvec
```

$A_1 B_1 + A_2 B_2 + A_3 B_3$

The length of a vector, or its absolute value, follows from the Pythagoros theorem

$$|\mathbf{A}| = \sqrt{A_1^2 + A_2^2 + A_3^2} = \sqrt{\mathbf{A} \cdot \mathbf{A}} = \sqrt{\mathbf{A}^2}$$

Mathematica gives

```
√(Avec.Avec)
```

$\sqrt{A_1^2 + A_2^2 + A_3^2}$

whereas

$\sqrt{\text{Avec}^2}$

$\{\sqrt{A_1^2}, \sqrt{A_2^2}, \sqrt{A_3^2}\}$

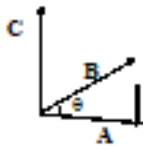
is not what we are looking for.

Cross product

Cross product of two vectors **A** and **B** is a vector **C**

$\mathbf{A} \times \mathbf{B} = \mathbf{C}$

that is perpendicular to both **A** and **B** (direction according to the screw rule)



and whose absolute value is given by

$|\mathbf{C}| = |\mathbf{A}||\mathbf{B}| \text{Sin}[\theta]$

In components, the cross product can be written via a rang-3 determinant

$\mathbf{A} \times \mathbf{B} = \text{Det}\left[\begin{pmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ A_1 & A_2 & A_3 \\ B_1 & B_2 & B_3 \end{pmatrix}\right] = (A_2 B_3 - A_3 B_2)\, \mathbf{e}_1 + \ldots$

Mathematica calculates cross products as

Avec × Bvec

$\{-A_3 B_2 + A_2 B_3, A_3 B_1 - A_1 B_3, -A_2 B_1 + A_1 B_2\}$

In this formula × (cross) should not be confused with × (Times).

The cross product can also be built in components as

$C_\alpha = \in_{\alpha\beta\gamma} A_\beta B_\gamma$ – summation over repeated indices!

Here $\in_{\alpha\beta\gamma}$ is the fully antisymmetric 3rd rank tensor:

$\in_{123} = \in_{312} = \in_{231} = 1$

$\in_{132} = \in_{321} = \in_{132} = -1$

and all other components being zero. In *Mathematica* it can be realized with the help of the Signature function

$\mathbf{Cvec} = \text{Table}\left[\sum_{\beta=1}^{3}\sum_{\gamma=1}^{3} \text{Signature}[\{\alpha, \beta, \gamma\}]\, \text{Avec}[[\beta]]\, \text{Bvec}[[\gamma]], \{\alpha, 1, 3\}\right]$

$\{-A_3 B_2 + A_2 B_3, A_3 B_1 - A_1 B_3, -A_2 B_1 + A_1 B_2\}$

Vector product is antisymmetric

$\mathbf{A} \times \mathbf{B} = -\mathbf{B} \times \mathbf{A}$

This follows both from the geometrical definition of the vector product and its definition in components. To check it, compute

```
Avec × Bvec + Bvec × Avec
```

```
{0, 0, 0}
```

or

```
Avec × Bvec == - Bvec × Avec
```

```
True
```

Vector product has a priority over scalar product, and scalar product has a priority over normal product of lists.

More complicated vector expressions

Mixed vector product

Mixed vector product is a dot product of a vector and a cross product of two other vectors and it is invariant with respect to the cyclic permutations of the three vectors

```
A · (B × C) = C · (A × B) = B · (C × A)
```

The geometrical interpretation of the mixed product is the volume of the parallelepiped built on the three vectors. In components one has

```
Avec = {A1, A2, A3};   Bvec = {B1, B2, B3};   Cvec = {C1, C2, C3};
Expand[Avec.Bvec × Cvec]
```

```
- A3 B2 C1 + A2 B3 C1 + A3 B1 C2 - A1 B3 C2 - A2 B1 C3 + A1 B2 C3
```

To check symmetry relations, compute

```
Avec.Bvec × Cvec == Cvec.Avec × Bvec // Simplify
```

```
True
```

etc.

Double cross product

Double cross product can be simplified to dot products

```
A × (B × C) ≡ B (A · C) - C (A · B)
```

Check this identity

```
Avec × (Bvec × Cvec) == Bvec (Avec.Cvec) - Cvec (Avec.Bvec) // Simplify
```

```
True
```

Dot product of two cross products

```
(A × B) · (C × D) ≡ (A · C) (B · D) - (A · D) (B · C)
```

Check this identity

```
Avec = {A₁, A₂, A₃};   Bvec = {B₁, B₂, B₃};
Cvec = {C₁, C₂, C₃};   Dvec = {D₁, D₂, D₃};
```

```
(Avec × Bvec) . (Cvec × Dvec) == (Avec.Cvec) (Bvec.Dvec) - (Avec.Dvec) (Bvec.Cvec) // Simplify
```

```
True
```

Note: Here brackets () are unnecessary because of priorities of operations.

Vector functions

Vector functions of scalar arguments can be defined as Lists of ordinary functions

```
Fvec[x_] := {1, x, x²}
```

```
Fvec[x][[3]]
```

x^2

Scalar functions of vector arguments can be defined as functions of Lists

```
F[XVec_] := √(XVec[[1]]² + XVec[[2]]² + XVec[[3]]²)  (* Absolute value of a vector *)
F[{1, 2, 3}]
```

$\sqrt{14}$

Finally, vector functions of vector arguments can be defined as Lists of functions of Lists

```
Gvec[XVec_] := {1, XVec[[1]], XVec[[2]] XVec[[3]]}
Gvec[{2, 3, 4}]
```

```
{1, 2, 12}
```

Vector plots

Vector functions can be plotted with VectorPlot, StreamPlot and other commands. For instance, the electric field of a positive point charge up to numerical factors has the

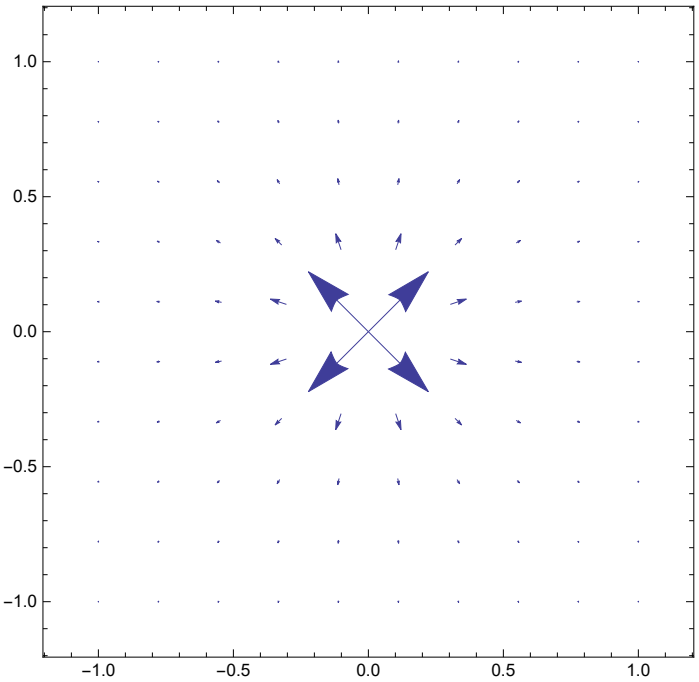
$$\mathbf{E}[\mathbf{r}] = \frac{\mathbf{r}}{\mathbf{r}^3} .$$

In the x-y plane (z = 0) there are only x and y components of E. So that in *Mathematica* we can define

```
Evec2[x_, y_] := {x / (x² + y²)^(3/2), y / (x² + y²)^(3/2)}
```

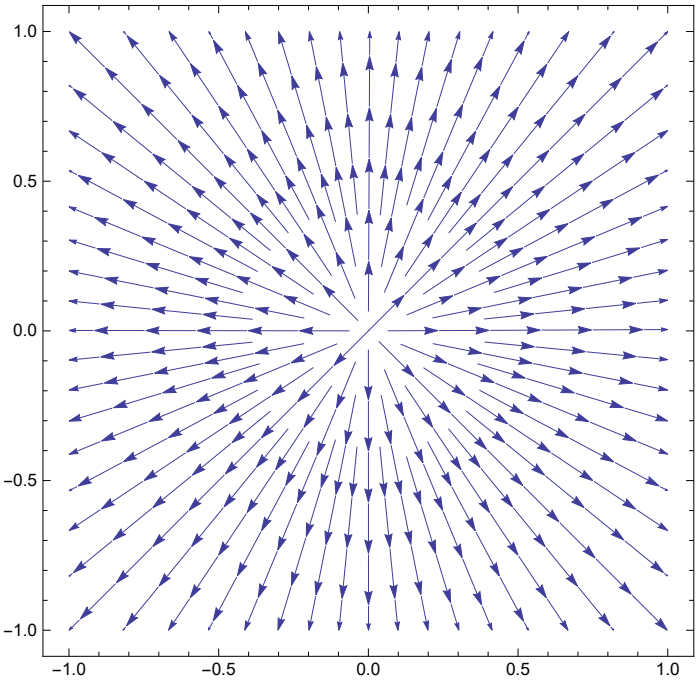
Now the vector plot of this function in the x-y plane has the form

```
VectorPlot[Evec2[x, y], {x, -1, 1}, {y, -1, 1}, VectorPoints → 10]
```



StreamPlot plots only the directions of the vectors

```
StreamPlot[Evec2[x, y], {x, -1, 1}, {y, -1, 1}]
```



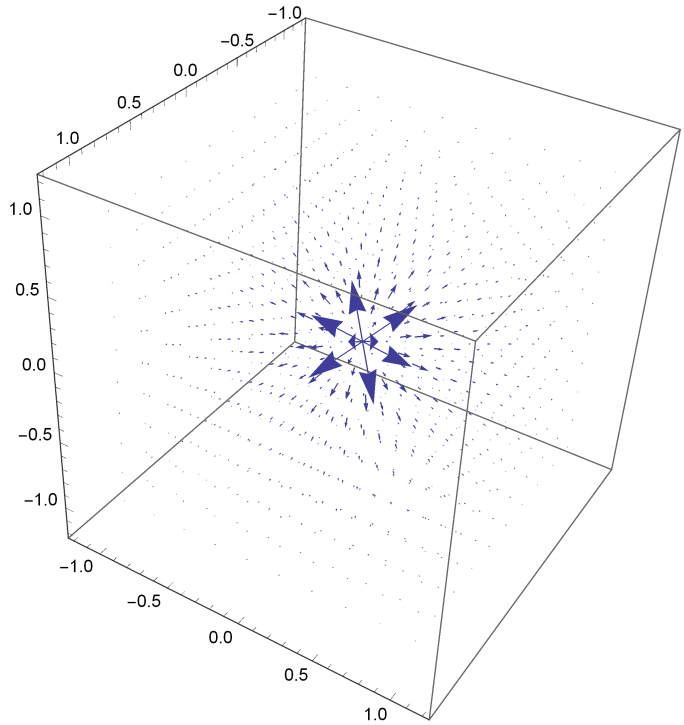
Note that *Mathematica* is not yet developed enough to vector plot functions of a vector argument. For vector plotting, the arguments have to be the vector components.

Exercise: Plot the vector field created by two charges, same and different signs.

Vector plotting is also possible in 3d.

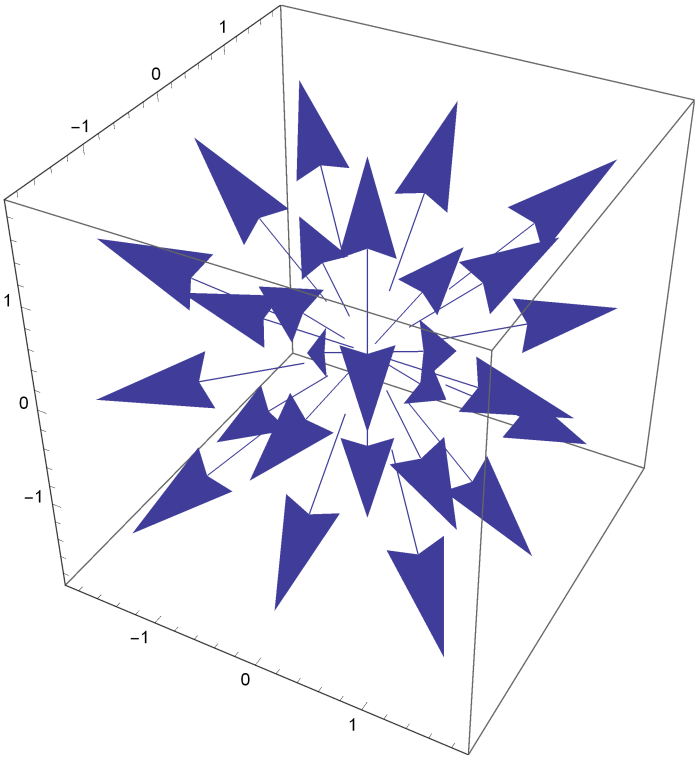
```
Evec[x_, y_, z_] := {x/(x^2 + y^2 + z^2)^(3/2), y/(x^2 + y^2 + z^2)^(3/2), z/(x^2 + y^2 + z^2)^(3/2)}
```

```
VectorPlot3D[Evec[x, y, z], {x, -1, 1}, {y, -1, 1}, {z, -1, 1}, VectorPoints → 10]
```



Unfortunately, StreamPlot3D does not exist. To work around the problem, one can plot **E**[**r**] r^2 that is equivalent to StreamPlot3D

```
VectorPlot3D[Evec[x, y, z] (x^2 + y^2 + z^2), {x, -1, 1}, {y, -1, 1}, {z, -1, 1}, VectorPoints → 3]
```



3d vector plots are not always easy to understand, however.

More general Lists; Matrices

Elements of a List can be Lists. An example considered above is

```
DataList = {{1, 2}, {2, 4}, {3, 5}, {4, 5}, {5, 4}, {6, 3}, {7, 2.5}};
```

Here the elements of DataList are sublists of length 2. Such Lists can be plotted with ListPlot, the first elements of sublists being interpreted as x_i and the second elements as y_i. The elements of such Lists of Lists can be called as follows

```
DataList[[2]]
```

```
{2, 4}
```

```
DataList[[2]][[1]]
DataList[[2, 1]] (* The same *)
```

```
2
```

```
2
```

Lists can have whatever complicated substructure. One can construct Lists of Lists of Lists etc. In all cases elements (parts) of the Lists can be addressed by specifying positions of the parts as above. DataList above is also a matrix that can be output explicitly in the matrix form

```
MatrixForm[DataList]
```

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 5 \\ 4 & 5 \\ 5 & 4 \\ 6 & 3 \\ 7 & 2.5 \end{pmatrix}$$

Lists, as matrices, can be transposed

```
Transpose[DataList]
MatrixForm[Transpose[DataList]]
```

```
{{1, 2, 3, 4, 5, 6, 7}, {2, 4, 5, 5, 4, 3, 2.5}}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 5 & 5 & 4 & 3 & 2.5 \end{pmatrix}$$

Especially important are square matrices

```
Matrix = {{1, 2}, {2, 4}};
MatrixForm[Matrix]
```

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

Lists can be generated by several commands, the most general being Table

In[1]:=
```
MyList = Table[{i, i^2}, {i, 1, 10}]
MatrixForm[Transpose[MyList]]
```

Out[1]=
```
{{1, 1}, {2, 4}, {3, 9}, {4, 16}, {5, 25}, {6, 36}, {7, 49}, {8, 64}, {9, 81}, {10, 100}}
```

Out[2]//MatrixForm=

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 & 100 \end{pmatrix}$$

or

In[8]:=
```
MyList3 = Table[A_{i,j,k}, {i, 1, 3}, {j, 1, 3}, {k, 1, 3}]
```

Out[8]=
```
{{{A1,1,1, A1,1,2, A1,1,3}, {A1,2,1, A1,2,2, A1,2,3}, {A1,3,1, A1,3,2, A1,3,3}},
 {{A2,1,1, A2,1,2, A2,1,3}, {A2,2,1, A2,2,2, A2,2,3}, {A2,3,1, A2,3,2, A2,3,3}},
 {{A3,1,1, A3,1,2, A3,1,3}, {A3,2,1, A3,2,2, A3,2,3}, {A3,3,1, A3,3,2, A3,3,3}}}
```

This list is an example of a tensor, a structure containing elements labeled by several indices.

The elements of this list can be obtained as

In[9]:=
```
MyList3[[1, 2, 3]]
```

Out[9]= $A_{1,2,3}$

etc.

In the above definition, the first index *i* is the outermost index. One can obtain all elements with a given *i* simply as.

In[11]:=
```
MyList3[[1]]
MyList3[[2]]
```

Out[11]= $\{\{A_{1,1,1}, A_{1,1,2}, A_{1,1,3}\}, \{A_{1,2,1}, A_{1,2,2}, A_{1,2,3}\}, \{A_{1,3,1}, A_{1,3,2}, A_{1,3,3}\}\}$

Out[12]= $\{\{A_{2,1,1}, A_{2,1,2}, A_{2,1,3}\}, \{A_{2,2,1}, A_{2,2,2}, A_{2,2,3}\}, \{A_{2,3,1}, A_{2,3,2}, A_{2,3,3}\}\}$

etc.

Elements with a given k can be obtained with the help of All symbol as

In[13]:=
```
MyList3[[All, All, 1]]
MyList3[[All, All, 2]]
```

Out[13]= $\{\{A_{1,1,1}, A_{1,2,1}, A_{1,3,1}\}, \{A_{2,1,1}, A_{2,2,1}, A_{2,3,1}\}, \{A_{3,1,1}, A_{3,2,1}, A_{3,3,1}\}\}$

Out[14]= $\{\{A_{1,1,2}, A_{1,2,2}, A_{1,3,2}\}, \{A_{2,1,2}, A_{2,2,2}, A_{2,3,2}\}, \{A_{3,1,2}, A_{3,2,2}, A_{3,3,2}\}\}$

etc.

Lists can be stripped of internal brackets by the command Flatten and its variants

In[15]:=
```
Flatten[MyList3]
```

Out[15]= $\{A_{1,1,1}, A_{1,1,2}, A_{1,1,3}, A_{1,2,1}, A_{1,2,2}, A_{1,2,3}, A_{1,3,1}, A_{1,3,2}, A_{1,3,3}, A_{2,1,1}, A_{2,1,2}, A_{2,1,3}, A_{2,2,1}, A_{2,2,2}, A_{2,2,3}, A_{2,3,1}, A_{2,3,2}, A_{2,3,3}, A_{3,1,1}, A_{3,1,2}, A_{3,1,3}, A_{3,2,1}, A_{3,2,2}, A_{3,2,3}, A_{3,3,1}, A_{3,3,2}, A_{3,3,3}\}$

In[18]:=
```
Flatten[MyList3, 1]
```

Out[18]= $\{\{A_{1,1,1}, A_{1,1,2}, A_{1,1,3}\}, \{A_{1,2,1}, A_{1,2,2}, A_{1,2,3}\}, \{A_{1,3,1}, A_{1,3,2}, A_{1,3,3}\}, \{A_{2,1,1}, A_{2,1,2}, A_{2,1,3}\}, \{A_{2,2,1}, A_{2,2,2}, A_{2,2,3}\}, \{A_{2,3,1}, A_{2,3,2}, A_{2,3,3}\}, \{A_{3,1,1}, A_{3,1,2}, A_{3,1,3}\}, \{A_{3,2,1}, A_{3,2,2}, A_{3,2,3}\}, \{A_{3,3,1}, A_{3,3,2}, A_{3,3,3}\}\}$