Manual for linalg.dev: A Node Editor Webapp for Building Linear Algebra Intuition in Three Dimensions

Cristian Perez Jensen
`cristianpjensen@gmail.com`
2022–11–27

Abstract

The intention of linalg.dev is to provide the user with mostly elementary mathematical tools and letting them use these to form vectors, planes, and matrices. Its purpose is to be a tool for students studying linear algebra to get an intuition of the underlying mathematics behind the concepts in linear algebra. By letting the user play with the vector space in three dimensions, the user may be able to get a better understanding of the concepts. It is also a lot more engaging to play with an actual three-dimensional environment than having to draw it out in mere two dimensions. It can also be used by linear algebra educators, because it is easy to download environments and hand them in as part of assignments.

Contents

Quick overview

Diagram of how one would use linalg.dev.

$$\begin{tikzcd}
\text{Define nodes} \arrow[d] \\
\text{Define relations between nodes by adding edges} \arrow[d] \\
\text{Observe vector space as nodes change} \arrow[d] \\
\text{Relate to theory}
\end{tikzcd}$$

1 Introduction

linalg.dev consists of two parts; the node editor and the three dimensional space that contains the vectors and planes defined by the node editor. The purpose of this tool is to build linear algebra intuition for what the various operations in linear algebra do and represent. By building the concepts with only elementary mathematical operators in the form of nodes, students will learn how the concepts work "behind the scenes" and relate it to whatever is happening in the vector space. It is also faster to iterate in linalg.dev than it is to redraw a two-dimensional space with pen and paper while studying certain concepts. Additionally, it can be used by educators for assignments, because environments can be downloaded and uploaded, so it is easy to hand in environments as part of an assignment.

→ Section 3

Many nodes are defined in linalg.dev and can be connected with edges in the node editor. For example, the output of a multiplication node can be connected to the x-axis of a vector. Relations between nodes can be defined in this way and is what makes the tool powerful.

→ Section 4

In the space, all vector and plane nodes are shown in a three-dimensional vector space. As the components of the vectors and planes change, they animate to their new position. Furthermore, it is possible to apply matrix transformations to the vector space. These transformations are animated as well.

2 Menu

The menu consists of three tabs; environments, exercises, and manual. The environments tab is a list of the user's environments. The user can also create new environments by pressing the +-button in the top right corner. Next to that button, there is also an upload button. A JSON file can be uploaded here that contains an environment. Together with the download button that is present for every environment, environments can be easily shared between teacher and student. A teacher could share a student that is useful to their students or the student could share their environment with a teacher as part of an assignment. Furthermore, the exercises tab is a list of exercises that the user could do with linalg.dev. Lastly, the menu also contains a condensed version of this manual.

→ Section 5

3 Nodes and edges

The nodes that are defined by linalg.dev are listed below. Keep in mind that most of the inputs can also be entered manually. The manual inputs accept most math functions and constants, such as π and the cosine function. After entering, the value of the input will be evaluated and the input will be replaced by its numerical value. Furthermore, the inputs and outputs can easily be read from the below overview, because they are denoted by (input,output). `N` denotes a number input, `V` denotes a vector input, and `M` denotes a matrix input.

Scalar (`N,N`): The scalar is the most simple node. It simply outputs a number, but is very useful for some kind of constant variable or for getting an intermediate result, because it also takes a number as an input.

Slider (`NN,N`): The slider node is also very simple. It takes two numbers as inputs that determine between which two numbers the slider should slide. It simply outputs the value of the slider. This node is very useful for some interactivity.

Unary operation (`N,N`): The unary operation node has three modes; square, cube, and square root. It performs that operation on its input and outputs the result.

Binary operation (`NN,N`): The binary operation is similar to the unary operation, but it takes two numbers as input and has the following modes: add, subtract, multiply, divide, and modulo. The output is the result of the operation.

Vector (`NNNV,V`): The vector node takes three numbers as input for the x, y, and z components of the vector respectively. The output is the vector of whatever the components make up. It also takes an origin vector as input, which is from which position the vector should start in the vector space. The vector also includes buttons for customizing its appearance in the vector space. From left to right, the buttons control the color of the vector, whether the vector should be represented as a dot (useful for data points) or a vector, and whether the vector should be hidden or not.

Plane (`VVV,`): The plane node takes three vectors as input. These vectors are the position vector and two direction vectors. The plane is then the plane defined parametrically by all points of the form

$$\vec{p} + \alpha\vec{v}_1 + \beta\vec{v}_2.$$

The plane's color can be customized in the same way as the vector.

Matrix (`VVV,M`): The matrix node takes three vectors as input, which are the column vectors of the matrix. The matrix also includes a transform button that applies the transformation defined by the matrix to the vector space. It outputs a matrix that can be used by other nodes as input.

Norm (`V,N`): The norm node outputs the norm (length) of the input vector.

Transform (`MV,V`): The transform node transforms the input vector by the input matrix and the output vector is the result of the transformation. By default, the resulting vector is hidden in the vector space.

Vector scaling (`VN,V`): The vector scaling node scales the input vector by the input number and the output vector is the result of the scaling. Again, by default, the resulting vector is hidden in the vector space.

Vector components (`V,NNN`): The vector components node outputs the x, y, and z components of the input vector. This is useful for custom operations, such as computing the dot product.

Transpose (`M,M`): The transpose node simply transposes the input matrix.

Matrix multiplication (`MM,M`): The matrix multiplication node simply multiplies the matrices. Let $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ be the first and second input matrix, respectively, then the output matrix is computed as $\boldsymbol{M}_1\boldsymbol{M}_2$.

Eigenvalues (`M,NNN`): The eigenvalues node outputs the eigenvalues of the input matrix, in descending order.

Eigenvectors (`M,VVV`): The eigenvectors node outputs the eigenvectors of the input matrix, in the same order as the eigenvalues. The eigenvectors can be hidden from the vector space, but by default they are not and have a purple color.

Nodes can be added to the node editor by selecting the node in the toolbar and clicking in the node editor where you want the node to be placed. Then, edges between nodes can be added by clicking an output and then dragging the edge to a connector that accepts the same type. The edges can be removed by clicking on the input connector of the edge. The nodes can be removed by clicking on the cross in the top right corner of the node. Per node, there are also more options that are described in the above list.

4 Vector space

→ Section 3

The vector space visualizes the vectors and the planes defined by the node editor. Furthermore, each vector and plane has customization options that changes how they are visualized. There are three global options that also changes how the space looks and behaves. The first option makes all vectors visualized as spheres, which is useful if the vectors represent data points. The next option shows an RGB cube wireframe around the space. This is useful for getting a better overview of how transformations affect the space. The last option toggles whether the entire space (including vectors and planes) should be transformed by matrices or only the vectors and planes. By transforming only the vectors and planes, the transformation is more clear with respect to the previous space.

5 Sample exercises

Basics: Basic exercises are meant for understanding the most basic linear algebra operations for vectors. Operations such as addition, subtraction, and the dot product are covered in these exercises.

1. Dot product. Given two vectors, the student must compute their dot product. The student is provided with two vector nodes and the result must be a scalar node containing the dot product between the two vectors. After finishing the exercise, the student can play around with different vectors and see how the vectors in the vector space relate to the dot product.
2. Vector addition. Given two vectors, the student must compute their addition. This exercise is very elementary, but it is very useful for the student to see how the vector space relates the three vectors with each other. Especially if one of the vectors is the origin of the other.
3. Vector distance. Given two vectors, the student must compute their distance. This exercise is a little harder, because it requires the insight that the distance between two vectors is the length of their subtraction.
4. Cosine angle. Given two vectors, the student must compute the cosine of their angle.
5. Unit vector. Given a vector, the student must compute its unit vector.

Advanced: Advanced exercises require some more insight than the basic exercises and also some more specific concepts can be introduced here.

1. Perpendicular vector. Given a vector, the student must compute any perpendicular vector. This exercise requires the insight that

perpendicular vectors have a dot product equal to zero, so they must solve the following equation for y

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ y \end{bmatrix} = 0.$$

2. Cross product. Given two vectors, the student must compute their cross product. This leads to an interesting theory question as to what the cross product represents.
3. Normal vector. Given one (normal) vector, the student must find the plane defined by it. This is done by using the previous two exercises, because the plane is defined by two vectors that are perpendicular to the normal vector.
4. Singular value decomposition. Given a matrix, the user must find its singular value decomposition. This is possible because of the eigenvectors and eigenvalues nodes. This leads to interesting visualizations as to how the matrices transform the vector space.
5. Principal component analysis. Given four vectors that are treated as data points, the student must compute the matrix that transforms them to two dimensions while losing the least amount of information. Many students just treat PCA as a scikit-learn function they can use, but this exercise will give them some knowledge as to what happens under the hood.
6. Principal component analysis, continued. Given the answer to the previous exercise, add a data point. This is an easier version of the previous exercise, but is still useful.

Theory visualized: linalg.dev can also be used for visualizing theory and answering questions about them.

1. Eigenpairs. The user is given an environment with a matrix and its eigenvectors and eigenvalues. Then, the student must play around with different matrices and see how the eigenvectors are affected when the transformations are applied. This should make it clear that the eigenvectors are only scaled by its eigenvalue when its matrix is applied.

Find the mistake: Another interesting type of exercise could be that a user is given a faulty reconstruction in linalg.dev of some concept. Then, the user must figure out what is going wrong and fix it.

6 Keyboard shortcuts

Lastly, there are some keyboard shortcuts that make it easy to quickly select a node type. These shortcuts are also shown next to each node in the toolbar.

⇧+V	Vector.
⇧+P	Plane.
⇧+M	Matrix.
⇧+S	Scalar.
⇧+L	Slider.
⇧+U	Unary operation.
⇧+B	Binary operation.
⇧+N	Norm.
⇧+R	Transform.
⇧+O	Vector scaling.
⇧+C	Vector components.
⇧+T	Transpose.
⇧+A	Matrix multiplication.
⇧+E	Eigenvalues.
⇧+I	Eigenvectors.

7 Adding new nodes

`https://reactjs.org/`

This section is intended for maintainers of the project and requires knowledge of how the React framework works. To create a new node type, the following steps must be done:

1. Add the type declaration of the node in `src/nodes/types.ts`. The inputs are defined by the fields and the outputs should be defined in the `output` field.
2. Make a new file with the node's name in `src/nodes`.
3. Build the React component in that file with the building blocks defined by `src/nodes/Node`.
4. Write the function for how the output should be computed from the input using the `useOutput` React hook.
5. Add how the node's data should be initialized in `src/nodes/nodeObjects.ts`.

Now that the node is fully defined, it must be added as a tool, such that it can be actually used:

1. Add the tool that represents the node to the `Tool` enum in `src/stores/editor.ts`.
2. Add the case for the tool in the switch case at line 98 in `src/Editor.tsx`.

3. Add the tool to one of the dropdowns in the toolbar in `src/Toolbar.tsx`.