# NUMERICAL LINEAR ALGEBRA WITH MATLAB

This document looks at some important concepts in linear and shows how to solve some basic problems using MATLAB.

## Contents

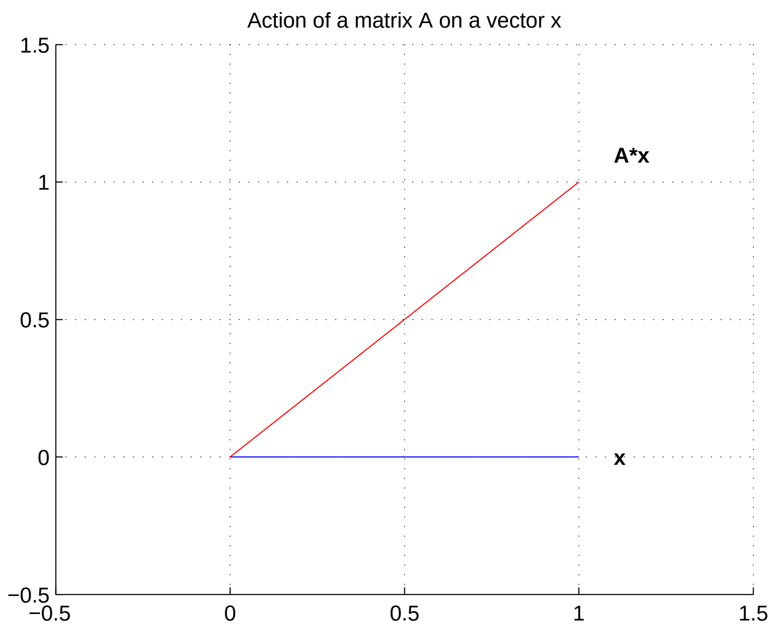
## Matrix-Vector products

Let's think about what a matrix-vector product actually does. It's more interesting that you think! Lets take a simple example. Consider the vector [1;0] multiplied by a matrix [1,1;1,1]

```
figure('color',[1 1 1])
A=ones(2)
x=[1;0]
Ax=A*x
line([0 x(1)],[0 x(2)]), hold on, grid on
line([0,Ax(1)],[0,Ax(2)],'color',[1 0 0])
text(1.1,0,'x','fontweight','bold'), text(1.1,1.1,'A*x','fontweight','bold')
axis([-0.5 1.5 -0.5 1.5])
title('Action of a matrix A on a vector x')
```

```
A =

     1     1
     1     1


x =

     1
     0


Ax =

     1
```

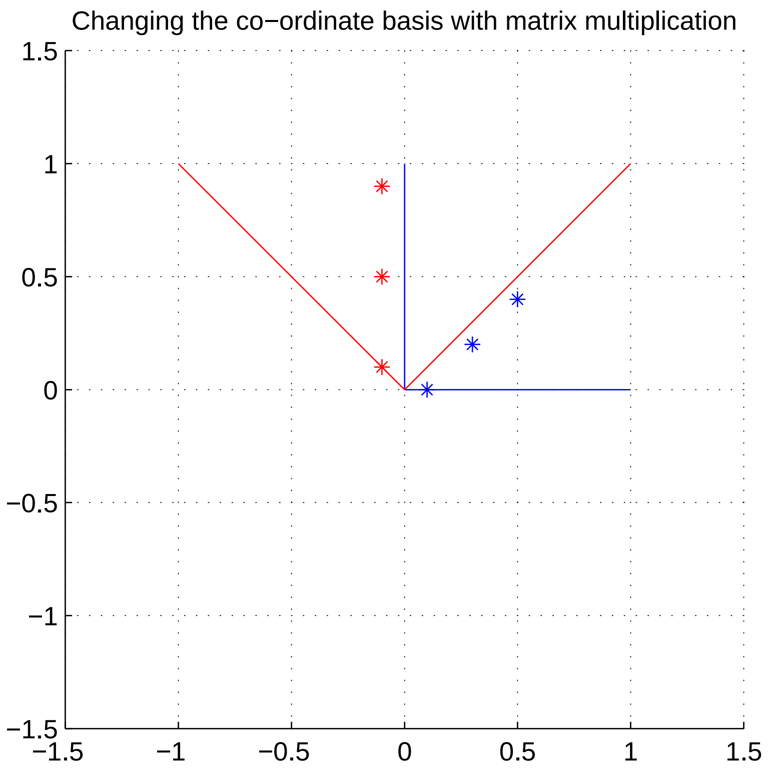
1



We can see that the matrix multiplication resulted in a vector that was rotated by a certain angle and also stretched by a certain factor. This action (stretching and rotating) is extremely useful for switching from one co-ordinate system [x,y] to another [x', y']. Something that you will come across in special relativity and a number of other areas. To visualise what we mean by a co-ordinate transformation, consider the basis vectors x:=[1,0] and y:=[0,1], and a point p:=[0.2,0.1].

```
figure('color',[1 1 1])
xy=[1,0;0,1]
p=[0.5,0.3,0.1;0.4,0.2,0.0]
A=[-1 1;1 1]
xyp=A*xy; % Co-ordinate rotation step
Ap=A*p;
line([0 xy(1,1)],[0 xy(2,1)]), grid on, hold on
line([0,xy(1,2)],[0,xy(2,2)])
plot(p(1,:),p(2,:),'*','markersize',6,'color','b')
line([0, xyp(1,1)],[0 xyp(2,1)],'color',[1 0 0])
line([0, xyp(2,1)],[0,xyp(2,2)],'color',[1 0 0])
plot(Ap(1,:),Ap(2,:),'*','markersize',6,'color','r')
axis([-1.5 1.5 -1.5 1.5]), axis square
title('Changing the co-ordinate basis with matrix multiplication')
```

```
xy =

     1     0
     0     1


p =

    0.5000    0.3000    0.1000
    0.4000    0.2000         0


A =

    -1     1
     1     1
```



## A rogue's gallery of useful matrices

Matrices can be thought of as a 2-D array of data, or a collection of vectors.

There are a number of important matrices that are extremely useful, and worth knowing about. We will use several of these later on in this lecture. Here are a few:

- Identity
- Ones/Zeros
- Upper/Lower Triangular
- Symmetric
- Vandermonde
- Finite Difference
- Rotation

The **Identity Matrix** or **I**

```
I = eye(4)
```

```
I =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

A **matrix of ones or zeros**:

```
O = ones(4)
Z = zeros(4)
```

```
O =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1


Z =

     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

**Upper and lower triangular matrices**. These are particularly important, as we shall see below, and consist of matrices that have zeros below, or above the *main diagonal*

```
UT = triu(randn(4))
LT = tril(randn(4))
```

```
UT =

    1.0933   -1.2141   -0.7697   -1.0891
         0   -1.1135    0.3714    0.0326
         0         0   -0.2256    0.5525
         0         0         0    1.1006


LT =

    1.5442         0         0         0
    0.0859    2.3505         0         0
   -1.4916   -0.6156   -0.7648         0
   -0.7423    0.7481   -1.4023   -0.1961
```

**Symmetric matrices** are matrices that have the same elements above he main diagonal as below the main diagonal:

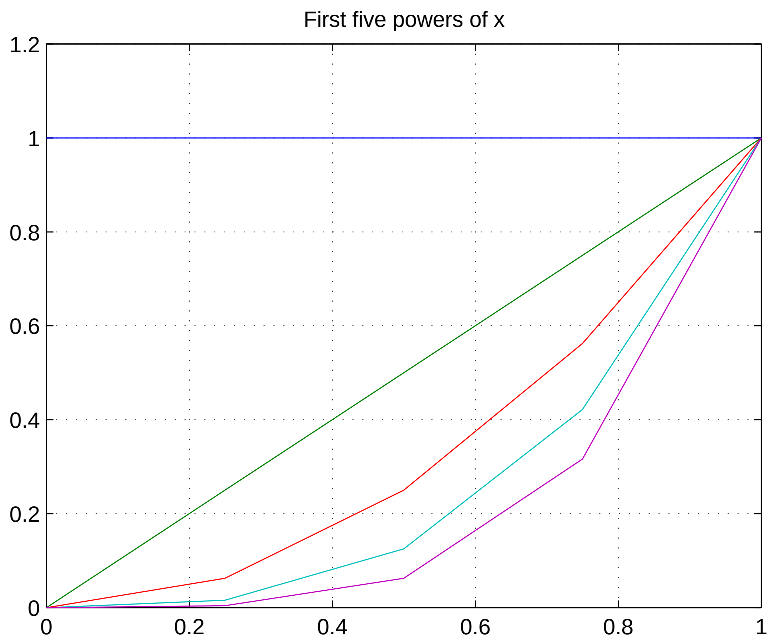
```
A = randn(4);
SYM = A*A'
```

```
SYM =

    3.2298    1.4694   -1.1203    2.6074
    1.4694    8.6143    0.2535    0.6552
   -1.1203    0.2535    2.4990   -0.1348
    2.6074    0.6552   -0.1348    2.6263
```

The **Vandermonde matrix** is a matrix whose columns are polynomials in $x$. This is useful in curve fitting and interpolation, for example, if I wanted to find the $3^{rd}$ order interpolating polynomial for some data [x,y], then I would use the following matrix:

```
figure('color',[1 1 1])
x=linspace(0,1,5)';
```

```
IM = [x.^0, x.^1, x.^2, x.^3, x.^4]
plot(x,IM), axis([0 1 0 1.2]), grid on
title('First five powers of x')

IM =

    1.0000         0         0         0         0
    1.0000    0.2500    0.0625    0.0156    0.0039
    1.0000    0.5000    0.2500    0.1250    0.0625
    1.0000    0.7500    0.5625    0.4219    0.3164
    1.0000    1.0000    1.0000    1.0000    1.0000
```



Generally for interpolation, it's better to use a matrix whose columns are **orthogonal** polynomials, such as Legendre or Chebyshev, but this is beyond the remit of this course.

**Finite difference matrices** are extremely useful. They are matrices that approximate the derivative of a function on a grid. One way to calculate them is to fit an interpolating polynomial through the function, and then differentiate the polynomial to get what is called a finite difference stencil, while the other is to use a Taylor expansion to approximate the derivatives.

As an illustration, we will use the Taylor expansion.

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{\Delta x^2 f''(x)}{2!} + h\text{igher order terms}$$

Here the dash represents a derivative with respect to x. If we ignore terms above the first derivative, we see we can say that

$$f'(x) \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$$

This is called the **forward difference** approximation. If we apply this formula to itself, we come up with an approximation for the second derivative of a function:

$$f''(x) \approx \frac{1}{\Delta x}\left(\frac{f(x+\Delta x+\Delta x)-f(x+\Delta x)-f(x+\Delta x)+f(x)}{\Delta x}\right)$$

or that, finally:

$$f''(x) \approx \frac{1}{\Delta x^2}\left(f(x+2\Delta x)-2f(x+\Delta x)+f(x)\right)$$

How to turn this into a matrix, however? If we imagine a function expressed on a grid of points $x_1, x_2, \cdots, x_N$ and replace the function evaluation $f(x+\Delta x)$ with function evaluations at the grid points, then the second derivative can be expressed as the following matrix:

```
N=5;
I=ones(N,1);
D2 = spdiags([I, -2*I, I],[-1:1],N,N);
full(D2)
```

```
ans =

    -2     1     0     0     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

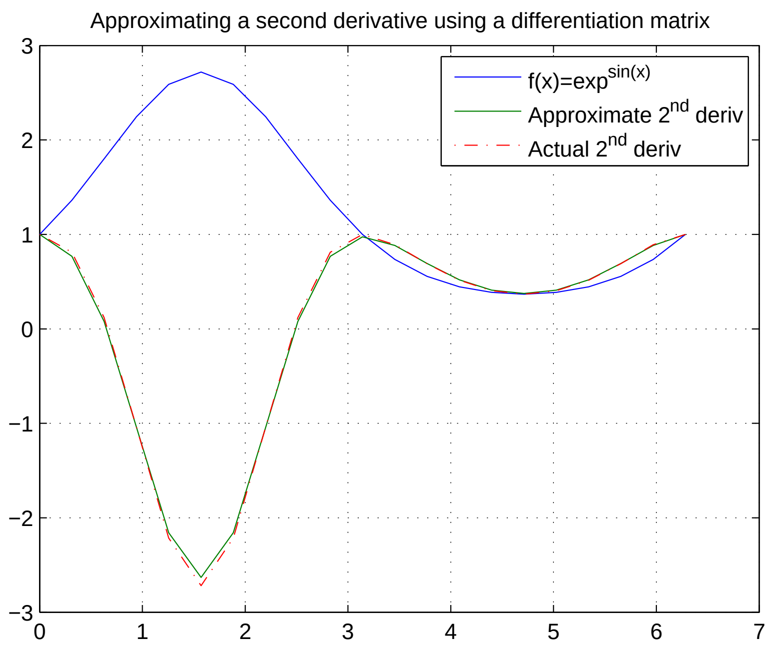
Here we use the construct spdiags to tell MATLAB that the matrix is sparse, i.e. is composed of mainly zero elements. In fact, this particular type of sparsity pattern is called **tridiagonal**. Consider the second row of this matrix multiplied by a vector which is the function evaluated on the grid of points $x_i$, $[f(x_0), f(x_1), f(x_2), \cdots, f(x_N)]^T$. We have the following result:

$$\begin{pmatrix} 1 & -2 & 1 & 0 & \cdots & 0 \end{pmatrix} \times \begin{pmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} = f(x_0) - 2 \times f(x_1) + f(x_2)$$

That is, the result of multiplying the second row of the differentiation matrix by a vector representing our function on the grid, gives an approximation to the derivative of our function on the grid. As an example, let's calculate the second derivative of $f(x) = \exp^{\sin(x)}$ on a grid of 21 points:

```
N=21; x=linspace(0,2*pi,N)'; % set up the grid
I=ones(N,1); dx=x(2)-x(1);
D2 = spdiags([I, -2*I, I],[-1:1],N,N)/dx^2; %2nd derivative matrix
f=@(x)(exp(sin(x))) % function
d2 = D2*f(x);       % numerical derivative
d2(1)=1; d2(N)=1;   % fix boundary points
figure('color',[1 1 1])
plot(x,f(x),x,d2), grid on, hold on
d2f=@(x)(cos(x).^2.*exp(sin(x))-sin(x).*exp(sin(x))) % actual second derivative
plot(x,d2f(x),'r-.')
title('Approximating a second derivative using a differentiation matrix');
legend('f(x)=exp^{sin(x)}','Approximate 2^{nd} deriv','Actual 2^{nd} deriv');
hold off
```

```
f =

    @(x)(exp(sin(x)))


d2f =

    @(x)(cos(x).^2.*exp(sin(x))-sin(x).*exp(sin(x)))
```



## Systems of linear equations

At its heart, the subject of linear algebra is concerned with how to solve the seemingly simple equation:

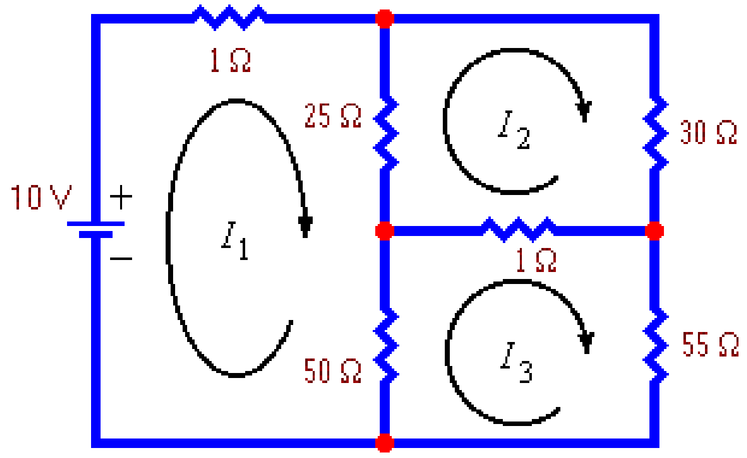
$\mathbf{A}x = b$, where $\mathbf{A}$ is a matrix and $x$ and $b$ are vectors.

This might seem surprising, but this problem is right at the heart of almost all problems in scientific computing, from differential equations to eigenvalue problems to interpolation and curve fitting.

Of course, for a single scalar variable, x, we know what that means: $a \times x = y$ from which we can deduce that $x = y/a$, for non-zero $a$.

The question is how do we solve such a problem when we have more than one unknown variable, or, equivalently, when x is a vector? To investigate this question, let's consider the following electrical circuit:

```
[c,m]=imread('circuit2.gif','gif');

figure('color',[1 1 1]),
image(c), colormap(m), axis equal, axis off
```



We want to calculate the currents $I_1, I_2, I_3$

Using Kirchoff's voltage laws we get the following three equations for the three unknown currents:

$$\left\{\begin{array}{rcl} I_1 + 25(I_1 - I_2) + 50(I_1 - I_3) & = & 10 \\ 25(I_2 - I_1) + 30I_2 + I_2 - I_3 & = & 0 \\ 50(I_3 - I_1) + I_3 - I_2 + 55I_3 & = & 0 \end{array}\right.$$

which, with a little re-arranging becomes:

$$\left\{\begin{array}{rcl} 76I_1 - 25I_2 - 50I_3 & = & 10 \\ -25I_1 + 56I_2 - I_3 & = & 0 \\ -50I_1 - I_2 + 106I_3 & = & 0 \end{array}\right.$$

Finally, we can gather all the coefficients into a matrix, and the unknowns into a vector, to write the single matrix linear equation:

$$\begin{pmatrix} 76 & -25 & -50 \\ -25 & 56 & -1 \\ -50 & -1 & 106 \end{pmatrix} \times \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 0 \\ 0 \end{pmatrix}$$

In this way we have got the system of equations into the form that we described above, namely: $\mathbf{A}x = b$

Of course, for a small system (3 by 3 in this case) we can solve this by hand. For a larger system, however, we need a computer. The basic approach to solving such a system is an extension of that used by hand, namely we add and subtract multiples of one row to one another until we finally end up with an upper triangular matrix. This action of adding and subtrating multiples of one row to another is known as a basic row operation. Let us illustrate the process with MATLAB. Before we begin, we augment the right hand side vector to be an extra column in our matrix **A**, so it is now three rows by four columns.

```
A=[76 -25 -50 10;-25 56 -1 0; -50 -1 106 0]
rrefexample(A)


A =

    76   -25   -50    10
   -25    56    -1     0
   -50    -1   106     0

Column: 1
Divide row 1 by 76
Subract -25 times row 1 from row 2
Subract -50 times row 1 from row 3

A =

    1.0000   -0.3289   -0.6579    0.1316
         0   47.7763  -17.4474    3.2895
         0  -17.4474   73.1053    6.5789

Column: 2
Divide row 2 by 47.7763
Subract -17.4474 times row 2 from row 3

A =

    1.0000   -0.3289   -0.6579    0.1316
         0    1.0000   -0.3652    0.0689
         0         0   66.7337    7.7802

Column: 3
Divide row 3 by 66.7337

A =

    1.0000   -0.3289   -0.6579    0.1316
         0    1.0000   -0.3652    0.0689
         0         0    1.0000    0.1166

Matrix A is now upper triangular

ans =
    1.0000   -0.3289   -0.6579
         0    1.0000   -0.3652
         0         0    1.0000
```

What is the advantage of reducing **A** to be upper triangular? Starting from the last row, we see that we now have an equation that depends only on $I_3$, that is $I_3 = 0.117$. The row above depends on $I_3$ and $I_2$ only, and we now know what $I_3$ is! In this way, we work our way up through the rows substituting as we go, and in each row there is only a single unknown variable. This process is called *back substitution*. This entire process - that of reducing the matrix to a triangular form and solving via back substitution - is the way most computer packages solve these linear systems. In MATLAB, the shorthand way of solving such systems is with the backslash character "\"

```
A=[76 -25 -50;-25 56 -1; -50 -1 106]
b =[10;0;0]
I= A\b
```

```
A =

    76   -25   -50
   -25    56    -1
   -50    -1   106


b =

    10
     0
     0


I =

    0.2449
    0.1114
    0.1166
```

In the example above, we have performed a special case of what is a more general idea - that of transforming a matrix into a special form, so that the equations can be solved more easily. In general, the idea of transforming a matrix through a series of elementary row operations is a powerful one. May special types of *matrix factorisations* exist, and two of the most important examples are:

**QR decomposition** In this case, a general matrix is written as the product of two matrices, one orthogonal and one upper triangular: $\mathbf{A} = \mathbf{QR}$ As the inverse of an orthogonal matrix is equal to its transpose, we find the solution to the general system $\mathbf{A}x = b$ can be calculated as:

$\mathbf{A}x = b$
$\mathbf{QR}x = b$
$\mathbf{Q}(\mathbf{R}x) = b$
$\mathbf{R}x = \mathbf{Q^T} \times b$
$x = \mathbf{R} \setminus \mathbf{Q^T} \times b$

**LU Decomposition** In this case the matrix is decomposed into two, a lower and an upper triangular matrix. As both the matrices are triangular, the back (and forward) substitution process is very quick:

$\mathbf{A}x = b$
$\mathbf{LU}x = b$
$\mathbf{L}(\mathbf{U}x) = b$
$\mathbf{L}x = \mathbf{U} \setminus b$
$x = \mathbf{L} \setminus (\mathbf{U} \setminus b)$

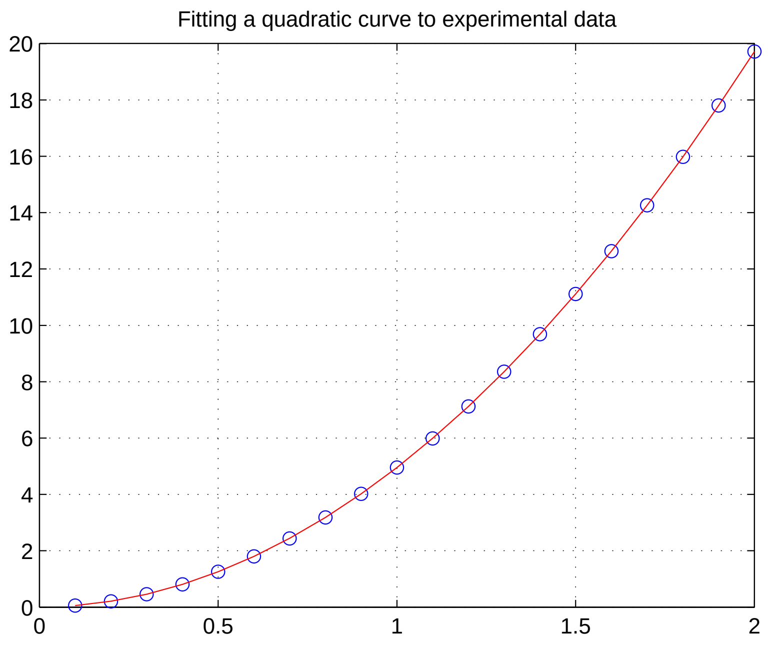
As another example of using MATLAB to solve linear systems, let us perform a least squares fit on some experimental data. We have run an experiment to measure the distance fallen by an object in a given time. We know from the laws of motion that the distance is related to the initial velocity and position, and the acceleration through a quadratic relationship:

$x(t) = x(0) + v(0)t + \frac{1}{2}gt^2$

We want to fit a quadratic line of best fit through the data, and hence work out a rough estimate for $g$ the acceleration due to gravity.

```
load fall_data
t=data(:,1);
dist=data(:,2);
I=ones(size(t));
R=[I,t,t.^2];  % This is a VANDERMONDE matrix (see above)
coeffs = R\dist
g=2*coeffs(3)  % approximation for g (coeff is g/2)
plot(t,dist,'o',t,R*coeffs,'r'), grid on
title('Fitting a quadratic curve to experimental data')
```

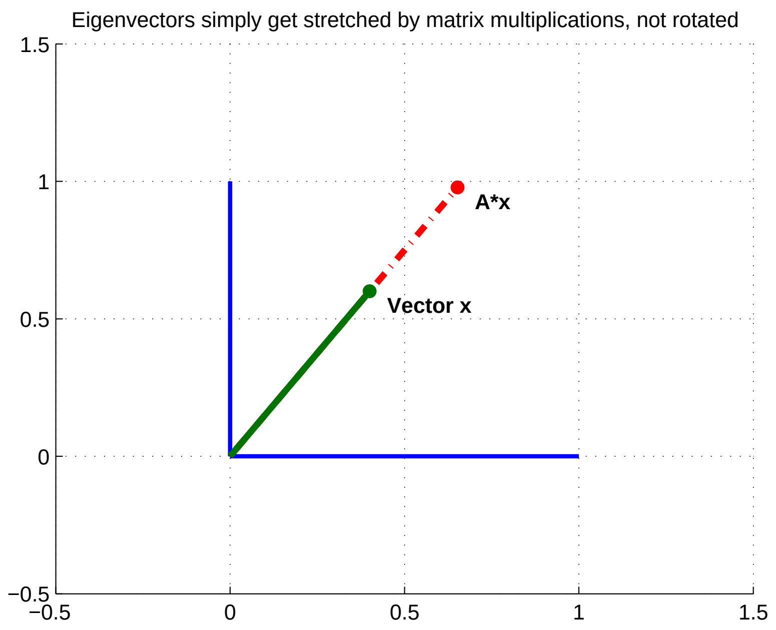
```
coeffs =

    0.0030
    0.0493
    4.9041


g =

    9.8081
```



## Eigenvectors and Eigenvalues of a matrix

As we have seen, the action of a mtrix on a vector is one of rotation and scaling. There are certain vectors, however, which in some sense *resonate* with some fundamental property of the matrix, and do not get rotated. These special vectors (or directions in N-Dimensional space) are called **eigenvectors** and the scaling factor by which they are stretched are called the **eigenvalues**. Mathematically what we are saying is: $\mathbf{Ax} = \lambda\mathbf{x}$ Or visually we are saying:

```
evecPlot
```



Essentially, what we are saying is that (square) matrices have certain “resonant modes” or “preferred” co-ordinate directions, which we call *eigenvectors* of the matrix. The eigenvectors are orthogonal to one another, and together form an orthogonal basis. If an $N \times N$ matrix has N distinct eigenvalues, and correspondingly N orthogonal eigenvectors, then it is said to have **Rank** N, or to be **full rank**.

To make all this more concrete, let’s consider an example from the mathematical theory of waves and vibration. In one dimension, the wave equation can be written as the following *partial differential equation* (don’t worry about what this means, exactly, you’ll learn more about all this later).

$$\frac{\partial^2 \Psi}{\partial t^2} = c\frac{\partial^2 \Psi}{\partial x^2}$$

We solve such a system by a process known as *separation of the variables*, and this involves writing the solution to the above equation as a product of two parts, one purely dependent on space and the other on time:

$$\Psi\left(x, t\right) = \Theta\left(x\right)\Phi\left(t\right)$$

By substituting this form of the solution back into the differential equation, we get:

$$\ddot{\Phi}(t)\Theta(x) = c\Phi(t)\Theta''(x)$$

or that

$$\frac{\ddot{\Phi}(t)}{\Phi(t)} = c\frac{\Theta''(x)}{\Theta(x)} = -k^2$$

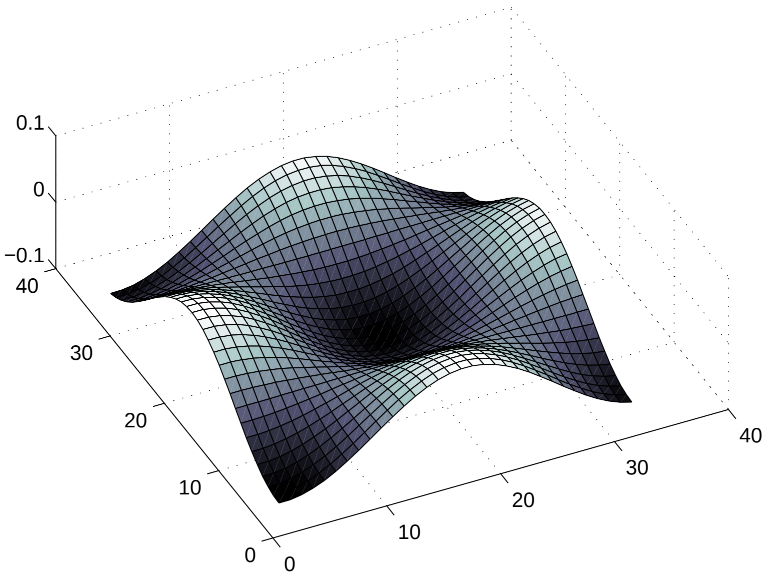
The only way that the left hand side, which is completely independent of $x$, can be equal to the right hand side, which is completely independent of $t$, is if both sides are equal to a constant, $-k^2$. We are interested in time independent solutions (standing waves) for this problem, so we can consider only the spatial parts of the equation, which can be written as:

$$\frac{\partial^2\Theta(x)}{\partial x^2} = -k^2\Theta(x)$$

This is an eigenvalue equation, exactly as we had above, and the solutions we are looking for are called **eigenfunctions**. This particular equation is important in physics and is known as the *Helmholtz equation.*

Using the second derivative differentiation matrix we described above (actually a more accurate version of it) and extending the notion to two dimensions, we transform a differential equation into a matrix eigenvalue equation. We can then use MATLAB to compute the eigenvalues and eigenvectors numerically to investigate the standing wave patterns, for example:

```
chladni
```


0.1
0
−0.1
40
30
20
10
0
0
10
20
30
40