Lab IV : Solving a system of linear equations and Kirchoff's laws

Example 1 : Solving a simple system of linear equations If we have a small system of equations, we may easily solve them by direct substitution

$$2x_1 - x_2 = -1$$
$$x_1 + x_2 = 10$$

By summing the two equation, we isolate x_1 and we proceed to find that $x_1 = 1.5$ and $x_2 = 4$. It is a rather standard procedure of linear algebra to cast such equations in the form of matrices

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

where

$$\mathbf{A} = \begin{pmatrix} 2 & -1 \\ 4 & 1 \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} -1 \\ 10 \end{pmatrix} \tag{1}$$

In order to solve the matrix equation above, we invert the matrix $\mathbf{A}$ and multiply both sides of the above matrix equation by its inverse. This operation of course assumes that this inverse exists.

$$\mathbf{A^{-1}A} \cdot \mathbf{x} = \mathbf{A^{-1}b} \Rightarrow \mathbf{x} = \mathbf{A^{-1}b} \tag{2}$$

In `Octave`, the function `inv` inverts nonsingular matrices. Thus, the above system may be solved easily by

```
octave:1> A=[2 -1;4 1]; b=[-1 10]';
octave:2> x=inv(A)*b
x =

  1.5000
  4.0000
```

Example 2 : A more complex system of linear equations This example is not only a demonstration of solving a linear system of equations but also serves as a reminder of tricks that we use to form matrices with certain well-defined patterns. Let's consider the following set of N equations :

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ x_2 + 2x_3 + 3x_4 &= 1 \\ x_3 + 2x_4 + 3x_5 &= 1 \\ &\vdots \\ x_{N-1} + 2x_N + 3x_1 &= 1 \\ x_N + 2x_1 + 3x_2 &= 1 \end{aligned}$$

These equations follow a shifting pattern which wraps around after the Nth variable. The corresponding matrix then looks like

$$C = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & \cdots & 0 & 0 \\ \vdots & & \vdots & & & & \vdots & \\ 3 & 0 & 0 & 0 & 0 & \cdots & 1 & 2 \\ 2 & 3 & 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

We are now going to write a function, `linear_system` that takes in the number N, which is the number of equations (and variables) and returns the solution of the above system. Let's first analyze the matrix above. It's a rather sparse matrix with only a strip of numbers in the middle. The main diagonal is filled with ones, the 1st diagonal is filled with twos and the second with threes. Other than that there are the three numbers on the lower left corner, which is a result of the wrapping around. You'll remember that to construct such a diagonal matrix by placing a given array on its diagonal, we use the `diag` function in `Octave`.

```
octave:1> a=rand(5,1)
a =

  0.10585
  0.75673
  0.80748
  0.28744
  0.70518

octave:2> M=diag(a)
M =

  0.10585  0.00000  0.00000  0.00000  0.00000
  0.00000  0.75673  0.00000  0.00000  0.00000
  0.00000  0.00000  0.80748  0.00000  0.00000
  0.00000  0.00000  0.00000  0.28744  0.00000
  0.00000  0.00000  0.00000  0.00000  0.70518
```

By supplying a second argument to the `diag` function we can shift the diagonal up or down.

```
octave:3> M=diag(a,1)
M =

  0.00000  0.10585  0.00000  0.00000  0.00000  0.00000
  0.00000  0.00000  0.75673  0.00000  0.00000  0.00000
  0.00000  0.00000  0.00000  0.80748  0.00000  0.00000
  0.00000  0.00000  0.00000  0.00000  0.28744  0.00000
  0.00000  0.00000  0.00000  0.00000  0.00000  0.70518
  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000

octave:4> M=diag(a,-1)
M =

  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  0.10585  0.00000  0.00000  0.00000  0.00000  0.00000
  0.00000  0.75673  0.00000  0.00000  0.00000  0.00000
  0.00000  0.00000  0.80748  0.00000  0.00000  0.00000
  0.00000  0.00000  0.00000  0.28744  0.00000  0.00000
  0.00000  0.00000  0.00000  0.00000  0.70518  0.00000

octave:5> M=diag(a,-2)
M =

  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  0.10585  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
  0.00000  0.75673  0.00000  0.00000  0.00000  0.00000  0.00000
  0.00000  0.00000  0.80748  0.00000  0.00000  0.00000  0.00000
  0.00000  0.00000  0.00000  0.28744  0.00000  0.00000  0.00000
  0.00000  0.00000  0.00000  0.00000  0.70518  0.00000  0.00000
```

As you can see the `diag` function adjusts the size of the resulting matrix to fit the entire vector on the given diagonal. One should thus make sure to supply a vector with correct length. With this information, it is no quite straightforward to write our function.

```
function X=linear_system(N)

  M=diag([ones(N,1)])+diag([2*ones(N-1,1)],1)+diag([3*ones(N-2,1)],2);
  M(N-1)=3;
  M(N,1:2)=[2 3];

  a=ones(N,1);
  X=inv(M)*a;

endfunction
```

When you run this function, you'll see that the answer is always the same, namely you get a repeating fraction of $0.1666\cdots$ for all the x_n where $n = 1, \cdots, N$. This is due to the fact that the equations are completely symmetric in all the variables. To break the symmetry a little, let's modify our function so that our equations read

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ x_2 + 2x_3 + 3x_4 &= -1 \\ x_3 + 2x_4 + 3x_5 &= 1 \\ x_4 + 2x_5 + 3x_6 &= -1 \\ &\vdots \\ x_{N-1} + 2x_N + 3x_1 &= -1 \\ x_N + 2x_1 + 3x_2 &= 1. \end{aligned}$$

That is to say, the inhomogeneous part is alternating ones and -1's. To do that, we can change the vector `a` in the above function to

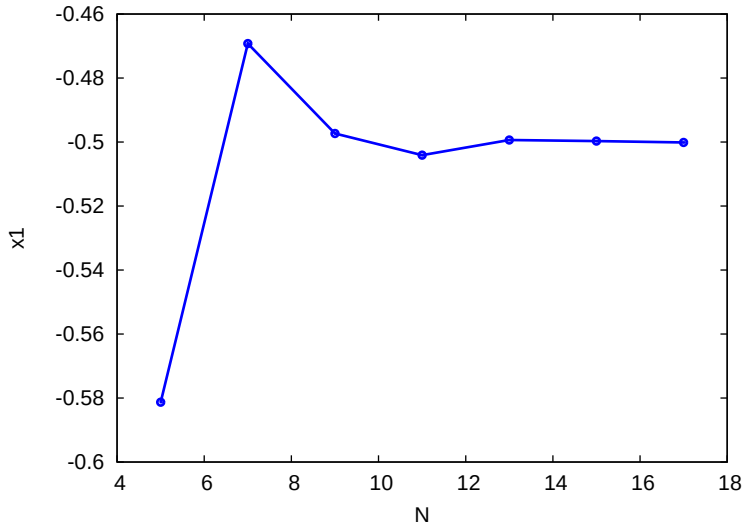
```
  a=ones(N,1);
  a(2:2:N)=-1;
```

while the rest of the function remains the same. The result is somewhat interesting : for even N, we always get an alternating array of 0.5 and -0.5 whereas for odd N, the results are different for different N.

We can now write a loop around the function `linear_system` and call it for a series of odd N and watch the progression of x_1 as a function of N. We can collect the consecutive values in an array and finally plot this array. The smallest meaningful number of variables for this system is 5.

```
octave:1> x1s=[];
octave:2> for N=5:2:17
> x1s=[x1s;linear_system(N)(1)];
> endfor
octave:3> plot([5:2:17],x1s,'b*-')
```

The resulting plot should look like this :



As you can see, as N gets larger x_1 changes value and finally asymptotes to a value near -0.5.

I would like to draw your attention to a construct that I used in the above example, which is

```
linear_system(N)(1).
```

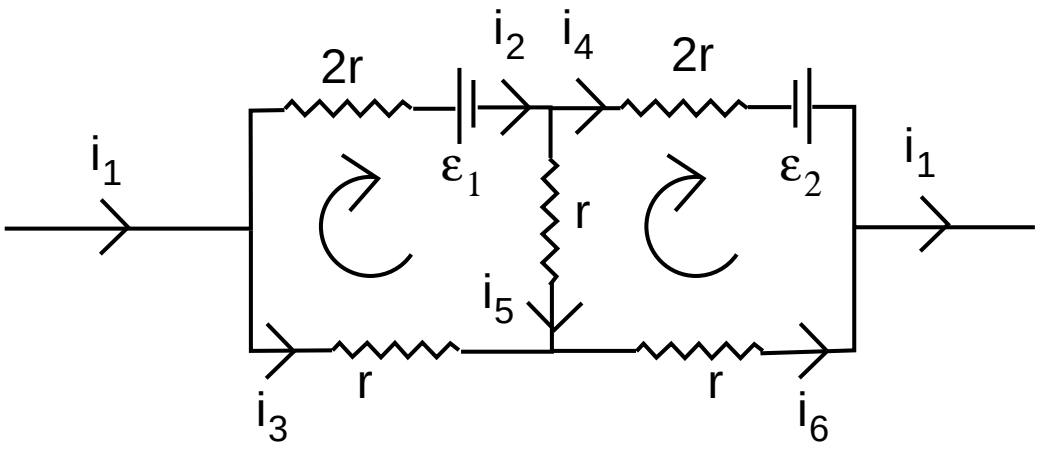
Because the function `linear_system` returns an array, you can access the elements of the resulting array directly from the function name without resorting to an intermediate array variable. **Example 3 : Kirchoff's rules** When we are faced with a simple circuit, we usually solve it by geometric considerations. If, however, the circuit that we need to solve is too complicated, we might not be able to visually simplify it. In that case, it's safest to resort to Kirchoff's rules. Kirchoff's rules state the following :

1. The sum of potential drops around a closed loop is always zero.
2. Due to charge conservation, the sum of all currents entering a junction and leaving a junction are equal.

In practice, we follow a set of rules of thumb to conduct calculations with Kirchoff's rules :

1. We mark all currents in all branches. The directions may be arbitrary.
2. We identify loops and give them directions. Directions are chosen completely arbitrarily.
3. We identify the junctions and apply the second rule.
4. We identify inequivalent loops and write the first rule. In doing this, we
   - take as negative the potential change across each resistor if we are going in the current and the loop are going in the same direction (positive if opposite).
   - take as negative the potential charge across each battery if we are going from the positive plate to the negative plate.
5. After writing as many inequivalent equations as there are unknown currents, we solve the resulting system of equations.

Let's write down the Kirchoff rules for the following circuit.



$$
\begin{aligned}
i_2 + i_3 &= i_1 \\
i_3 + i_5 &= i_6 \\
i_4 + i_6 &= i_1 \\
i_4 + i_5 &= i_2 \\
-2i_2r - \epsilon_1 - i_5r + i_3r &= 0 \\
-2i_4r + \epsilon_2 + i_6r + i_5r &= 0
\end{aligned}
$$

We see that for some values of r, ϵ_1 and ϵ_2, some current values are zero and some others are negative. A zero current simply means that there is no charge transfer across that branch whereas a negative current implies that an incorrect direction has been assigned to the current in that branch.

```
function currents=kirchoff1(ep1,ep2,r)

  A=[ 1  -1  -1   0   0   0;
      0   0   1   0   1  -1;
     -1   0   0   1   0   1;
      0   1   0  -1  -1   0;
      0 -2*r  r   0  -r   0;
      0   0   0 -2*r  r   r];


  E=[0 0 0 0 ep1 -ep2]';

  currents=inv(A)*E;

endfunction
```