Computational Physics – Lecture 5

Andrei Alexandru

February 26, 2014

1 Introduction

This week we will discuss how to solve systems of linear equations. As you know, a system of N linear equations with N unknowns can be written generically as

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N = b_1\,, \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N = b_2\,, \\ &\quad\vdots \\ &a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N = b_N\,. \end{aligned} \tag{1}$$

Equivalently, we can write this system as a matrix equation $Ax = b$, that is

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N} \\ a_{21} & a_{22} & \ldots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}}_{b}\,. \tag{2}$$

A is an $N \times N$ matrix with entries $A_{ij} = a_{ij}$ that are the coefficients of the unknowns. Each equation constitutes a row in this matrix. The unknowns are collected in the column vector x and the right hand sides of the equations are collected in the column vector b.

In the first part of the week, we will focus on writing a routine to solve this type of problems and in the second part we will use this routine to solve a resistor circuit problem.

2 Gauss elimination with partial pivoting

A system of linear equations can be solved using *Gauss elimination*. Before we show how to approach this problem in the general case, we will work out a simple example. Take the system of three equation with three unknowns

$$\begin{aligned} e_1: &\quad x + y + z = 3\,, \\ e_2: &\quad x + 2y + 2z = 5\,, \\ e_3: &\quad 2x + 3y + 4z = 9\,. \end{aligned} \tag{3}$$

To solve this problem we can subtract $1 \times e_1$ from e_2 and $2 \times e_1$ from e_3 to get a system of equivalent equations that do not involve x in e_2' and e_3'. We have

$$\begin{aligned} e_1: &\quad x + y + z = 3\,, \\ e_2': &\quad 0 + y + z = 2\,, \\ e_3': &\quad 0 + y + 2z = 3\,. \end{aligned} \tag{4}$$

We continue to get rid of y from the third equation by subtracting e'_2 out of e'_3. We get

$$\begin{aligned} e_1: &\quad x+y+z=3\,,\\ e'_2: &\quad 0+y+z=2\,,\\ e''_3: &\quad 0+0+z=1\,. \end{aligned} \tag{5}$$

At this point we are ready to solve the system. From e''_3 we determine that $z=1$. Plugging this in e'_2 we find that $y=2-z=1$. Finally using the values of z and y we can determine x_1 from e_1: $x=3-y-z=1$.

The first part of the process is eliminating variable x_i from equations $e_{i+1},\ldots,e_N$, for $i=1,\ldots,N$. At the end of this elimination process, equation e_i only involves variables x_i through x_n. In terms of matrix equation, the equivalent system involves a *triangular* matrix, that is a matrix that has all elements under the diagonal zero:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N}\\ 0 & a'_{22} & \ldots & a'_{2N}\\ \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \ldots & a'_{NN} \end{pmatrix}}_{A'} \underbrace{\begin{pmatrix} x_1\\ x_2\\ \vdots\\ x_N \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} b_1\\ b'_2\\ \vdots\\ b'_N \end{pmatrix}}_{b'}\,. \tag{6}$$

Note that although the matrix equation is different, the solutions for x is the same.

Elimination is an iterative process. Assume we have already performed the elimination up to row $i-1$ and the equivalent equation is given by

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1i-1} & a_{1i} & \ldots & a_{1N}\\ 0 & a'_{22} & \ldots & a'_{2i-1} & a'_{2i} & \ldots & a'_{2N}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \ldots & a'_{i-1i-1} & a'_{i-1i} & \ldots & a'_{i-1N}\\ 0 & 0 & \ldots & 0 & a'_{ii} & \ldots & a'_{iN}\\ 0 & 0 & \ldots & 0 & a'_{i+1i} & \ldots & a'_{i+1N}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \ldots & 0 & a'_{Ni} & \ldots & a'_{NN} \end{pmatrix} \begin{pmatrix} x_1\\ x_2\\ \vdots\\ x_{i-1}\\ x_i\\ x_{i+1}\\ \vdots\\ x_N \end{pmatrix} = \begin{pmatrix} b_1\\ b'_2\\ \vdots\\ b'_{i-1}\\ b'_i\\ b'_{i+1}\\ \vdots\\ b'_N \end{pmatrix}\,. \tag{7}$$

To eliminate x_i from equations e'_j for $j>i$ we have to subtract a multiple of equation e_i from the subsequent equations. The multiplication factor is chosen to make the coefficient a'_{ji} vanish after subtraction. For each row $j>i$ we have

$$a''_{jk}=a'_{jk}-\frac{a'_{ji}}{a'_{ii}}a'_{ik} \quad \text{for} \quad k=i,\ldots,N \qquad \text{and} \qquad b''_j=b'_j-\frac{a'_{ji}}{a'_{ii}}b'_i\,. \tag{8}$$

The procedure is repeated for all rows from 1 to $N-1$. Note that there is nothing to do for row N since there are no subsequent equations. At the end of the process the matrix is triangular.

Once we reduced the system to a triangular matrix, we are ready to solve the system using *back substitution*. It is easy to see that we can solve the last equation directly by setting $x_N=b'_N/a'_{NN}$. We can then back track and solve equation e'_{N-1}, using the value for x_N to find x_{N-1}. This process continues until we find x_1. To be precise, we find that the equation e_i corresponding to row i of matrix A',

$$a'_{ii}x_i+a'_{ii+1}x_{i+1}+\ldots+a'_{iN}x_N=b'_i, \tag{9}$$

is solved by

$$x_i=\frac{1}{a'_{ii}}\left(b'_i-\sum_{j=i+1}^{N}a'_{ij}x_j\right)\,. \tag{10}$$

This works because by the time we reach equation e'_i we already know the values of x_j for all $j>i$.

The final algorithmic issue has to do with pivoting. Note that to cancel the elements below diagonal element a'_{ii}, we have to subtract the row i from row j. The row i is multiplied by a'_{ji}/a'_{ii} as can be seen

from Eq. 8. If the diagonal element a_{ii} is zero, or nearly zero, the process will fail or introduce very large numerical errors. To avoid this situation we can use *partial pivoting*, that is we can swap equations. To better understand the procedure, consider this system of equations

$$\begin{aligned} e_1: &\quad x+y+z+w=4\,,\\ e_2: &\quad 2x+2y+3z+3w=10\,,\\ e_3: &\quad 2x+3y+4z+3w=12\,,\\ e_4: &\quad 3x+2y+4z+2w=11\,. \end{aligned} \tag{11}$$

After eliminating x from e_2, e_3, and e_4, we get

$$\begin{aligned} e_1: &\quad x+y+z+w=4\,,\\ e_2': &\quad 0+0+z+w=2\,,\\ e_3': &\quad 0+y+2z+w=4\,.\\ e_4': &\quad 0-y+z-w=-1\,. \end{aligned} \tag{12}$$

If we try to use equation e_2' to cancel y from e_3' we would fail since there is no y in equation e_2'. However, this can be easily fixed by swapping equations e_2' and e_3' to get the following equivalent system of equations

$$\begin{aligned} e_1: &\quad x+y+z+w=4\,,\\ e_2'': &\quad 0+y+2z+w=4\,.\\ e_3'': &\quad 0+0+z+w=2\,,\\ e_4': &\quad 0-y+z-w=-1\,. \end{aligned} \tag{13}$$

Now we can proceed to eliminating y form the equations following e_2'' without any problems.

In general, we always scan the column below a_{ii} to find the largest element in magnitude. If we find this element on row k such that $|a_{ki}| > |a_{ji}|$ for all $j \geq i$, we swap row i and k and then proceed to eliminate x_i from the subsequent equations. It is possible that the largest element below the diagonal is zero, or nearly zero. In this case, this means that we have a singular or nearly singular matrix and the system does not have a well defined solution.

3 Implementation of Gauss elimination

To find the solution of the system $Ax = b$ we will first reduce the system to a triangular matrix and then use back substitution to solve it. The matrix and vectors will be stored as arrays of `doubles`. For a $N \times N$ matrix we need N^2 elements in the array and for the column vectors x and b, N elements for each array. We will store the elements of the matrix in *row major* order, that is A_{ij} will be found at $a[i \times N + j]$, assuming that array `a` stores the matrix. Note that in the C programs the array indices run from 0 to $N-1$.

A number of utility routines need to be implemented. To help test and debug our codes, we implement `print_mat` a routine to print a matrix to the terminal.

```
1 void print_mat(double* m, int N)
2 {
3   for(int r=0; r<N; ++r)
4   {
5     for(int c=0; c<N; ++c)
6       printf("% 10.5f ", m[r*N+c]);
7     printf("\n");
8   }
9 }
```

To help pivoting we need to be able to swap individual elements of arrays and whole rows of matrices. We use the following routines

```
void swap(double* a, double* b)
{
  double c=*a;
  *a = *b;
  *b = c;
}

void swap_rows(double* m, int N, int r1, int r2)
{
  for(int c=0; c<N; ++c) swap(m+r1*N+c, m+r2*N+c);
}
```

Note that in the `swap` routine we pass two pointers, `a` and `b`. These point to the addresses in memory where the values that we want swapped are stored. When the routine returns the value `*a` will be stored at `b` and the value `*b` will be stored at `a`. The second routine swaps the entire `r1` row with row `r2` of matrix `m`. It is also important to note that every time we pass a matrix to a routine we need to also pass a parameter indicating its dimension.

The last utility routine that we need is the `find_pivot`, a routine that scans the entries below a_{rr} diagonal element and determines the row that has the largest magnitude entry. The code listing is given below

```
int find_pivot(double* m, int N, int r)
{
  int p = r;
  double val = fabs(m[r*N+r]);
  for(int rp=r+1; rp<N; ++rp)
    if(fabs(m[rp*N+r]) > val)
    {
      p = rp;
      val = fabs(m[rp*N+r]);
    }

  return p;
}
```

The idea is to scan the elements $a_{ii}, a_{ii+1}, a_{ii+2}, \ldots, a_{iN}$ to find the one that has the largest magnitude and its corresponding row. This will be needed to identify the best pivot for elimination. Please take some time to understand how these routines work.

With these routines, we implement the elimination step as follows:

```
void elimination(double* m, int N, double* b)
{
  for(int r=0; r<N-1; ++r)
  {
    int p = find_pivot(m, N, r);
    if(p != r)
    {
      swap_rows(m, N, r, p);
      swap(b+r, b+p);
    }

    double piv = m[r*N+r];
    if(fabs(piv) < 1e-20)
    {
      printf("matrix nearly singular. exiting...\n");
      abort();
    }
```

```
18
19      for(int rp = r+1; rp<N; ++rp)
20      {
21        double fact = -m[rp*N+r]/piv;
22        for(int c=r; c<N; ++c) m[rp*N+c] += fact*m[r*N+c];
23        b[rp] += fact*b[r];
24      }
25    }
26  }
```

In lines 5–10 we identify the maximal pivot and, if this is not on the current row, we swap the rows. On lines 19–24 we perform the elimination following Eq. 8.

The back substitution routine follows Eq. 10:

```
1   void bks(double* m, int N, double* b, double* x)
2   {
3     for(int r=N-1; r > -1; --r)
4     {
5       double temp = b[r];
6       for(int c=r+1; c<N; ++c) temp -= m[r*N+c]*x[c];
7       x[r] = temp/m[r*N+r];
8     }
9   }
10
11  void solve_system(double* m, int N, double* b, double* x)
12  {
13    elimination(m, N, b);
14    bks(m, N, b, x);
15  }
```

To wrap things up we write a `solve_system` routine that runs the elimination and back substitution steps in the right order.

4 Resistor problems

One area where we encounter systems of linear equations in physics is in the study of electrical circuits. Kirchhoff's laws allow us to find the currents and voltages that appear in such a circuit. In this section we will restrict ourself to circuits that involve resistors only. We will be concerned with finding the equivalent resistance of between two points in such a circuit.

For simple resistor circuits the usual rules to compute the equivalent resistance for serial and parallel connections are sufficient. For more complex circuits a more generic approach might be required. In fact, even for simple circuits it may be difficult to determine the equivalent resistance using only the serial and parallel connections rules. For example the equivalent resistance for the circuit in Fig. 1 between vertices V_0 and V_2 cannot be determined by reducing the circuit to serial or parallel arrangements. We will show how to solve this problem by reducing it to a system of linear equations and then we will write a routine to solve generic circuits of this kind.

To determine the equivalent resistance between a pair of vertices, V_a and V_b, we will use a virtual ohmmeter: we imagine that we inject an external current I at V_a and extract the same current I at V_b. The equivalent resistance can be determined from the potential drop $R_{\text{eq}} = (V_a - V_b)/I$, where $V_a - V_b$ is the potential drop between the vertices. Note that the potential drop as defined is positive since the current flows from vertices of higher potential to lower potentials. Thus we have to solve this circuit problem in the presence of this external currents.

In general such a circuit has a electrostatic potential for every vertex and a current running through each resistor. The unknowns are the potentials at each vertex V_i, $i = 0, 1, \ldots, N_v - 1$, and the currents running through each resistor I_i, $i = 0, 1, \ldots, N_r - 1$. The total number of unknowns are $N_v + N_r$. In the circuit in

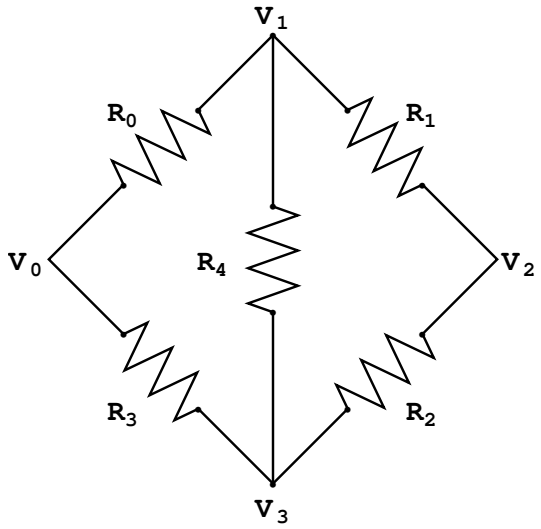

Figure 1: A simple resistor circuit that cannot be solved with serial and parallel connection rules.

Fig. 1 we have 4 vertices and 5 resistors for a total of 9 unknowns. We need to identify this many equations to pin down these values.

A first set of equations relates the currents and the potential differences at the ends of each resistor. For resistor i connected at vertices k and l we have

$$V_k - V_l - I_i R_i = 0\,. \tag{14}$$

Note that the signs of the currents depend on whether they flow in or out of the vertex. Conventionally we chose the currents flowing out to be positive and the ones flowing in to be negative. To determine the direction of the flow we have to arbitrarily fix a direction of the current for each resistor. If we don't guess the actual direction correctly, the only effect is that the solution for that current will be negative. We will index each current with two indices indicating that the corresponding resistor is attached to those vertices and taking the positive flow direction from the first vertex to the second.

For example for our test circuit we will use the following table for currents

Current	In vertex	Out vertex	Resistance
I_0	0	1	R_0
I_1	1	2	R_1
I_2	2	3	R_2
I_3	0	3	R_3
I_4	1	3	R_4

Note that this table completely describes the circuit since it specifies how many resistors we have and how they are connected. In our case, we have

$$\begin{aligned}
e_0: &\quad V_0 - V_1 - I_0 R_0 = 0\,,\\
e_1: &\quad V_1 - V_2 - I_1 R_1 = 0\,,\\
e_2: &\quad V_2 - V_3 - I_2 R_2 = 0\,,\\
e_3: &\quad V_0 - V_3 - I_3 R_3 = 0\,,\\
e_4: &\quad V_1 - V_3 - I_4 R_4 = 0\,.
\end{aligned} \tag{15}$$

A second set of equations is given by the current conservation at each vertex. For every vertex, we have

$$\sum_{j \text{ connected to } i} I_j = -I_{\text{ext}}(i)\,. \tag{16}$$

For our case, the equations for current conservation at vertices are

$$\begin{aligned} e_5: &\quad I_0 + I_3 = I\,, \\ e_6: &\quad -I_0 + I_1 + I_4 = 0\,, \\ e_7: &\quad -I_1 + I_2 = -I\,, \\ e_8: &\quad -I_2 - I_3 - I_4 = 0\,. \end{aligned} \tag{17}$$

Note that the external current is zero for all vertices except e_0 where we inject a current I and at e_2 where the external current I flows out. There seem to be 4 independent equation (or in the general case N_v such equations). However, not all of them are independent. To verify that sum all equations together: the total sum is zero both on the right hand side and on the left hand side of the equation. On the left hand side it is because each current runs between two vertices, flowing out of one with positive sign and flowing into the other one with negative sign. On the right hand side you get zero because the net external current is zero. This means that one equation is superfluous. We eliminate the last equation from our system of equations, e_8.[1]

We are almost ready: we have 9 unknowns and 8 equations. The mismatch has to do with the fact that there is no physical principle that can fix the value of the potentials, they are only defined up to an additive constant. To fix the values, we can choose an arbitrary vertex as the "ground", the point where the potential is zero. We will choose the last potential $V_{N_v-1} = 0$. We can think of this as an extra equation added to our system or eliminate V_{N_v-1} as an unknown from our equations. We will use the second option.

For our circuit, the final system of equations is $e_0, \ldots, e_7$ which can be expressed as a matrix equation as follows:

$$\begin{pmatrix} -R_0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & -R_1 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & -R_2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -R_3 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -R_4 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \\ I_4 \\ V_0 \\ V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ I \\ 0 \\ -I \end{pmatrix}. \tag{18}$$

Note that in the above system V_3 was replaced with zero everywhere. This system can be solved to get

$$\begin{aligned} V_0 &= I\frac{R_3[R_1R_4 + R_0(R_3+R_4)]}{(R_1+R_2)(R_0+R_3) + (R_0+R_1+R_2+R_3)R_4}\,, \\ V_2 &= I\frac{R_2[R_0(R_1+R_4) + R_1(R_3+R_4)]}{(R_1+R_2)(R_0+R_3) + (R_0+R_1+R_2+R_3)R_4}\,. \end{aligned} \tag{19}$$

The equivalent resistance is then

$$R_{\text{eq}} = \frac{V_0 - V_2}{I} = \frac{R_0R_1R_2 + R_0R_1R_3 + R_0R_2R_3 + R_1R_2R_3 + (R_0+R_1)(R_2+R_3)R_4}{(R_1+R_2)(R_0+R_3) + (R_0+R_1+R_2+R_3)R_4}\,. \tag{20}$$

5 Implementation of the resistor problem

For our C code we will solve the generic problem for a resistor circuit when the resistors values are given numerically. We will reduce the problem to a matrix problem $Mx = b$. We will follow the same steps as in the previous section to set up the matrix M and left hand side vector b.

We will define a routine that computes the equivalent resistance for a circuit. The circuit will be specified using a list of resistors. For each resistor we will need to know the connected vertices and the resistance. The current will be assumed to flow from the first to the second vertex. To store all this information compactly we will define a `struct resistor` as follows

[1] Note that only one equation is dependent. If there are more than one dependent equations, that means that the circuit is not connected, that is, there are more than two pieces of circuit completely detached from each other.

```
1  struct resistor
2  {
3    int v1, v2;
4    double R;
5  };
```

The routine we will write will take as arguments the number of vertices `nv`, the number of resistors `nr`, a list of resistors that describes the circuit `rl`, and the two vertices that serve as leads for our "ohmmeter", `beg` and `end`. We show the code for the entire routine and then comment each section

```
1  double compute_resistance(int nv, int nr, struct resistor* lr,
2      int beg, int end)
3  {
4    int neq = nr+nv-1;
5
6    double *m = malloc(neq*neq*sizeof(double));
7    double *b = malloc(neq*sizeof(double));
8    double *x = malloc(neq*sizeof(double));
9
10   for(int i=0; i< neq*neq; ++i) m[i] = 0;
11
12   for(int i=0; i<nr; ++i)
13   {
14     int v = lr[i].v1;
15     if(v < nv-1) m[(nr+v)*neq + i] = +1;
16     v = lr[i].v2;
17     if(v < nv-1) m[(nr+v)*neq + i] = -1;
18   }
19
20   for(int i=0; i<nv-1; ++i) b[nr+i] = 0;
21   if(beg < nv-1) b[nr+beg] = +1;
22   if(end < nv-1) b[nr+end] = -1;
23
24   for(int i=0; i<nr; ++i)
25   {
26     m[i*neq + i] = -lr[i].R;
27     int v = lr[i].v1;
28     if(v < nv-1) m[i*neq + nr + v] = +1;
29     v = lr[i].v2;
30     if(v < nv-1) m[i*neq + nr + v] = -1;
31     b[i] = 0;
32   }
33
34   solve_system(m, neq, b, x);
35   double vbeg = (beg<nv-1)? x[nr+beg] : 0;
36   double vend = (end<nv-1)? x[nr+end] : 0;
37
38   free(b);
39   free(x);
40   free(m);
41
42   return vbeg-vend;
43 }
```

In lines 4–10, we determine the total number of equations, allocate space for our matrix and column vectors and set all the entries in the matrix to zero.

The `for` loop in lines 12–18 implement the current equations in a roundabout way. If you look at the current equations in the previous section e_5, e_6, e_7, e_8 you notice that all the currents appear twice since they are attached to two vertices. Thus we can take each resistor and determine the two equations in which it appears. For a resistor attached to vertices v and v', they will appear in equations e_{nr+v} and $e_{nr+v'}$. For the first equation the coefficient is $+1$ since the current flows out of that vertex. For the second equation the coefficient is -1 for similar reasons. The reason we vertex v corresponds to equation e_{nr+v} is because the vertex equations follow after nr current equations. Finally, recall we decided that equation $e_{nr+nv-1}$ is not independent from the other $nv-1$ vertex equations and we decided to eliminate it from our system. This is why on line 15 and 17 we check whether the equation that we target is not the one eliminated before adding the entries to the matrix m. Otherwise we would write past the allocated storage.

On lines 20–22 we write the right hand side of the vertex equation. The external current is zero for all vertices except for `beg` and `end`. As before we check that neither of this vertices belongs to the equation we eliminated.

On lines 24–32 we write the `nr` resistor equations. On each of these rows we have three entries: two due to the potentials at the end of the resistor and one due to the current running through it. For resistor i we have equations e_i, thus we focus on the row i of matrix `m`. The current entry belongs to column i since the current appears on row i in the x column vector (see Eq. 18). The coefficient for the current is `-lr[i].R`. For the potentials the corresponding columns are `nr+lr[i].v1` and `nr+lr[i].v2` with coefficients $+1$ and -1 as we can see from Eq. 14. For potentials we have to make sure we don't include V_{N_v-1} since this is set to zero. This is the reason why on lines 28 and 30 we perform the tests before adding these entries to the matrix.

Finally on lines 34–36 we solve the linear system to find all the unknowns collected in the column vector x. Then we identify the entries in `x` where the potentials for vertices `beg` and `end` are stored. Note that we check whether any of these vertices are $N_v - 1$ since the potential there is zero and it isn't store in the solution vector `x`.

We return the difference $V_{\text{beg}} - V_{\text{end}}$ as the equivalent resistance since the current running through this vertices is set to 1. Before returning we free the allocated memory for the matrix and the column vectors.

Before we conclude, we show a simple main routine where the above routine is used to compute the equivalent resistance for the circuit shown in Fig. 1. Note that in our code we have to specify the resistances numerically. We choose $R_0 = 1$, $R_1 = 2$, $R_2 = 3$, $R_3 = 4$, and $R_4 = 5$. The equivalent resistance between vertices 0 and 2 is given by the formal in Eq. 20. For this case we find $R_{\text{eq}} = 31/15$. Our `main routine` is given below

```
1  int main()
2  {
3    int nv = 4;
4    int nr = 5;
5    struct resistor lr[] = { 0, 1, 1,  1, 2, 2,  2, 3, 3,  0, 3, 4,  1,
         3, 5};
6
7    double eqr = compute_resistance(nv, nr, lr, 0, 2);
8    printf("equiv resistance: %e\n", eqr);
9
10   return 0;
11 }
```

The output of our code is `equiv resistance:  2.066667e+00` which agrees very well with the analytic result.