Solving fun engineering problems using iterative methods

Quasar Chunawala*
github.com/quantophile/math
(Dated: May 10, 2020)

These code snippets demonstrate how iterative methods are used to numerically solve problems in engineering and science.

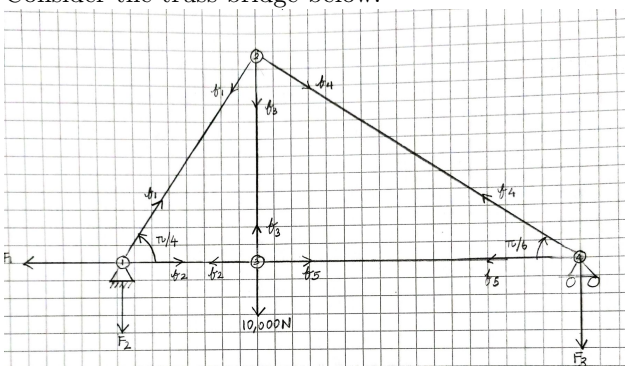
I. INTRODUCTION.

The beautiful gleaming Sydney Harbor bridge is made up of arch trusses.

Trusses are lightweight structures capable of carrying heavy loads. In bridge design, the individual members of the truss are connected with rotatable pin joints that permit forces to be transferred from one member of the truss to another. This distributes the load evenly.

Finite Element Analysis(FEA) methods are used by structural engineers to analyse and model truss bridges. However, we can do with a elementary free body diagram for the truss and work out the tension/compression forces on the bridge using Newton's first law of motion.

Consider the truss bridge below:



The truss is held stationary at the lower-left endpoint 1, it is permitted to move horizontally at the lower right endpoint 4 and has pin joints at 1,2,3,4. A load of $10,000$ Newtons is placed at joint 3. The resulting internal forces are given by f_1, f_2, f_3, f_4, f_5. When positive these forces indicate on the truss elements and when negative, compression. The stationary support member could have an external reaction force with both a horizontal and vertical component, but the movable support member only has a vertical force component.

If the truss is in static equilibrium, the forces at each joint must add to the zero vector, so the sum of the horizontal and vertical components at each joint are 0. This produces the system of linear equations shown in the table below. An 8×8 matrix describing this system has 47 zero entries and only 17 nonzero entries. Matrices with large percentage of zero entries are often solved using iterative rather than direct techniques.

Joint	Horizontal force component
1	$-F_1 + (1/\sqrt{2})f_1 + f_2 = 0$
2	$-(1/\sqrt{2})f_1 + (\sqrt{3}/2)f_4 = 0$
3	$-f_2 + f_5 = 0$
4	$-(\sqrt{3}/2)f_4 - f_5 = 0$

Joint	Vertical force component
1	$(1/\sqrt{2})f_1 - F_2 = 0$
2	$-(1/\sqrt{2})f_1 - f_3 - (1/2)f_4 = 0$
3	$f_3 - 10000 = 0$
4	$(1/2)f_4 - F_3 = 0$

In different applications, it happens that the system matrix has the special structure that many of its entries are zero. Systems of this type frequently arise in circuit analysis, physical problems, spline interpolation etcetera.

A matrix that has many zero entries is called *sparse*. Iterative techniques are efficient in terms of both computer storage and running time for sparse matrices.

II. JACOBI ITERATION

Consider a 3×3 system :

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \tag{1}$$

One iterative technique to solve the linear system is obtained by solving the first equation for x_1, second equa-

* quasar.chunawala@gmail.com

tion for x_2 and the third equation x_3 like this :

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}[0 \cdot x_1 - a_{12}x_2 - a_{13}x_3 + b_1] \\ x_2 &= \frac{1}{a_{22}}[-a_{21}x_1 - 0x_2 - a_{23}x_3 + b_2] \\ x_3 &= \frac{1}{a_{22}}[-a_{31}x_1 - a_{32}x_2 - 0x_3 + b_3] \end{aligned} \tag{2}$$

That's it! Simply solve the i-th equation in $Ax = b$ for x_i.

For each $k \geq 1$, generate the components $x_i^{(k)}$ of $x^{(k)}$ from the components of $x^{(k-1)}$ by :

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\sum_{j=1 j \neq n}^{n} (-a_{ij} x_j^{(k-1)}) + b_i \right] \tag{3}$$

The successive iterates $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ converge to the true solution vector x.

III. JACOBI ITERATION - NAIVE C++ IMPLEMENTATION

I wrote a naive solver in C++ that implements the Jacobi iterative method. Here's the source code:

```
#include <eigen-3.3.7/Eigen/Dense>
#include <iostream>

using namespace Eigen;

double infty_norm(VectorXd x) {
    int n = x.size();
    double max_elem = 0.0;
    double abs_x_i;
    for (int i = 0; i < n; ++i) {
        abs_x_i = abs(x(i));
        if (abs_x_i > max_elem)
            max_elem = abs_x_i;
    }
    return max_elem;
}

void jacobi_solve(MatrixXd& A, VectorXd& x,
    VectorXd& b)
{
    // Naive implementation of Jacobi iteration

    int k = 1;
    int N = 200;
    VectorXd x_0;
    int n = A.rows();
    double tolerance = 1e-4;

    while (k < N)
    {
        x_0 = x;

        for (int i = 0; i < n; ++i)
        {
            double a_ii_inv = 1.0 / A(i, i);
            double sum = 0.0;
            for (int j = 0; j < n; ++j) {
                if (j != i)
                    sum += -A(i, j) * x_0(j);
            }
            x(i) = a_ii_inv * (sum + b(i));
        }

        if ((infty_norm(x - x_0) / infty_norm(x))
            < tolerance)
            break;

        std::cout << "k = " << k << std::endl;
        std::cout << "\n"<< x << std::endl;

        k +=1;
    }
}

int main()
{
    MatrixXd A(4, 4);
    VectorXd b(4);
    VectorXd x(4);

    A << 10,    -1,     2,      0,
        -1,     11,     -1,     3,
         2,     -1,     10,     -1,
         0,     3,      -1,     8;

    b << 6, 25, -11, 15;

    x << 0, 0, 0, 0;

    jacobi_solve(A, x, b);

    std::cout << x << std::endl;

    return 0;
}
```

In the above code, I stop iterating once

$$\frac{||x^{(k)} - x^{(k-1)}||_\infty}{||x^{(k)}||_\infty} < TOL \tag{4}$$

IV. TRUSS BRIDGES.

The linear system in the motivating example can be written in matrix form as -

$$Ax = b$$

where

$$\mathrm{A} = \begin{bmatrix} -1 & 0 & 0 & \frac{1}{\sqrt{2}} & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & -1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{\sqrt{3}}{2} & -1 \end{bmatrix}$$

$x = [F_1, F_2, F_3, f_1, f_2, f_3, f_4, f_5]^t$

$b = [0, 0, 0, 0, 0, 10^4, 0, 0]^t$

The iterative solver produces the solution vector x

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} = \begin{bmatrix} 3.3473 \\ -6341.42 \\ -3658.58 \\ -8968.21 \\ 6336.85 \\ 10000 \\ -7322.44 \\ 6336.85 \end{bmatrix}$$

V. RANDOM WALK ON A STRAIGHT LINE

Suppose that a drunkard on a street(modelled as a particle on the real line $\mathbb{R}$) can be at any one of the $(n+1)$ equally spaced points. When he is at location x_i, he is equally likely to move to any other location. Consider the probabilities $\{P_i\}_{i=0}^{n}$ that starting at location x_i he will reach the left endpoint x_0 before reaching the right endpoint x_n. Clearly, $P_0 = 1$ and $P_n = 0$. Since the particle can move from x_i only to x_{i-1} or x_{i+1} and does so with probability $\frac{1}{2}$ for each of these locations,

$$P_i = \frac{1}{2}P_{i-1} + \frac{1}{2}P_{i+1}$$

In simple words, the probability of hitting the left endpoint before the right endpoint starting at node x_i = one-half the probability of doing the same from node x_{i-1} plus one-half the chance of doing the same thing from node x_{i+1}.

We can easily show that $Ax = b$, where the coefficient matrix A is,

$$\mathrm{A} = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & \dots & 0 & 0 & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 & -\frac{1}{2} & 1 \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & -\frac{1}{2} \end{bmatrix}$$

$x = (P_1, P_2, \dots, P_{n-1})$ and $b = (\frac{1}{2}, 0, 0, \dots)$

Clearly, $P_1 = \frac{1}{2}P_0 + \frac{1}{2}P_2$, so $P_1 - \frac{1}{2}P_2 = \frac{1}{2}$. Moreover, $P_{n-1} = \frac{1}{2}P_{n-2} + \frac{1}{2}P_n$. But, $P_n = 0$, so $P_{n-1} - \frac{1}{2}P_n = 0$. These two equations correspond to the first and last rows of A. So, the matrix equation checks out.

I solved this $(n-1) \times (n-1)$ system for three cases - $n = 10$, $n = 50$, and $n = 100$.

The probability distribution generated by solver for $n = 10$,

```
x(0) = 0.897328
x(1) = 0.795167
x(2) = 0.693006
x(3) = 0.59218
x(4) = 0.491354
x(5) = 0.39218
x(6) = 0.293006
x(7) = 0.195167
x(8) = 0.0973284
```