# 欧拉伯努利横梁的研究

夏钰棋 陈祺越 陈致真

上海交通大学 致远学院

（2020年6月28日）

## 引言

欧拉伯努利衡量的矩阵模型求解问题，其本质上是一个针对稀疏矩阵求解方式的优化问题。在具体的问题中，我们并不需要进行完整的高斯消去法求解，而是可以根据实际的情况去修改算法的过程，实现问题求解的优化——更精确的结果，更快速的运算。

## 产生稀疏矩阵并且使用优化的高斯消去法求解

1. 产生稀疏矩阵A

采用循环语句输入，观察到对于3到n-2行矩阵具有相同的项

1. 优化高斯消去法求解

高斯消去法采用逐行消去系数成为上三角矩阵，而我们注意到每次只需要对至多三行进行消去第一列的系数即可使得这一列主对角线以下的数都是零，同时每次对其中一行进行消去系数的时候，只需要对至多四列进行操作，而后续的列中的元素都是零，不需要计算。同时，进行消去操作的时候也只需要考虑四行即可。

事实上，我们发现整个矩阵的带宽是7，而当n非常大的时候，进行优化的高斯消元法计算可以大大地减少程序的运算量，由）减少到），完成了一次非常有价值的优化。具体的代码实现在附录给出

## 三、无运动员的情形

由于右侧负载一定（恒为常数），那么对此问题求解就转化为了对稀疏矩阵A求Ax=b的问题。线性方程组的解法主要分为直接求解和迭代求解两种。为了探寻针对较大的稀疏矩阵哪一种算法更加合适，我们同时采用了高斯消元法、贾可比迭代、高斯赛德尔迭代，以及连续过松弛的方法。我们对比了这几种算法的收敛速度和收敛精度，来评价哪种方法更好。

对于直接求解的办法（高斯消去法），最终能够收敛，但是一般的高斯消去法复杂度为,而即使改进过的、针对稀疏矩阵的高斯消元复杂度也有。随着n的不断增大，计算的时间会以平方速度增长，这对处理大数据是很不利的。那么高斯消去法的误差如何？经过计算，误差向量的模为0.0181。这个误差是相当大的。

下面我们来看看利用迭代法处理伯努利横梁的效果。

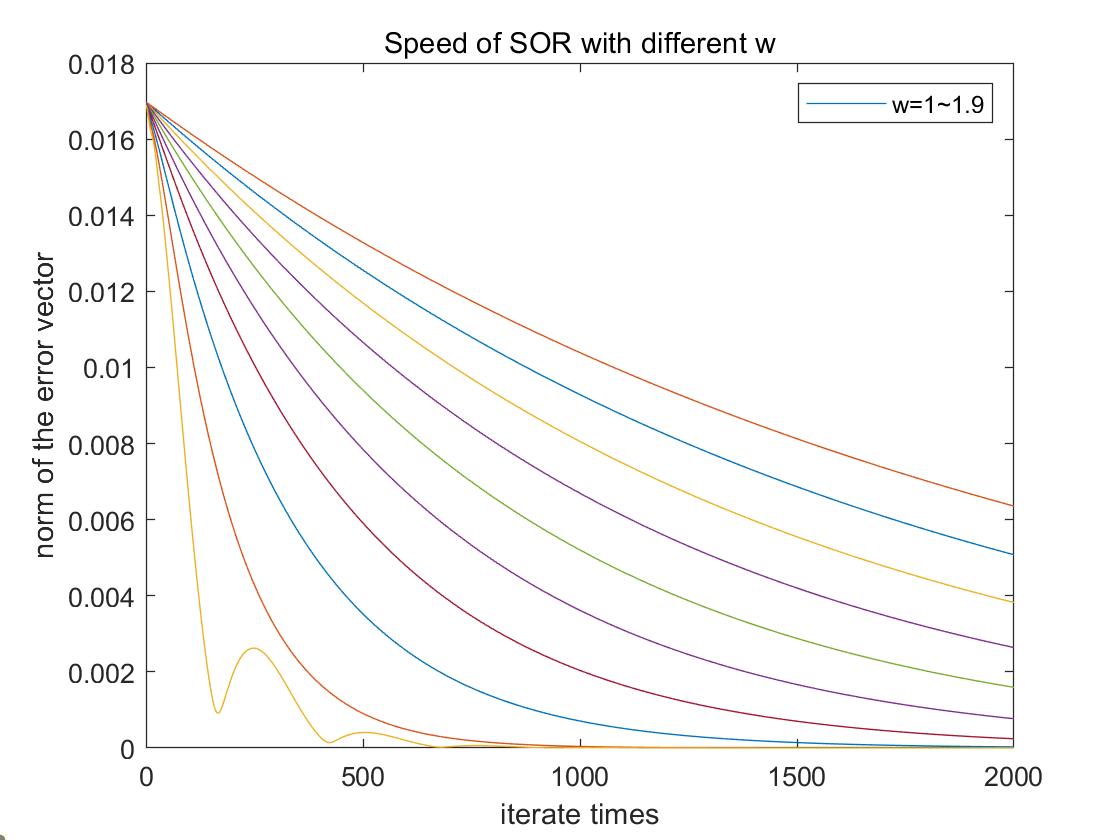
根据贾可比迭代法和高斯赛德尔迭代的方程（如下）我们可以写出迭代程序尝试进行求解。（附录详细展示了代码，这里不展开）

)

然而，经过程序实践我们发现，针对10阶的稀疏矩阵，贾可比迭代法并不收敛而高斯赛德尔方法收敛到了近似解。为什么呢？这是因为迭代是否收敛与前矩阵的谱半径有关。贾可比迭代法中 矩阵的谱半径约等于1.7（大于1），因此不收敛。而高斯赛德尔方法中我们可以把方程改写成 ，此时矩阵的谱半径决定能否收敛。而此时谱半径小于1，因此高斯赛德尔能够收敛到近似解。

那么高斯赛德尔迭代的速度如何？我们先来看每一次迭代的复杂度。如果处理阶数为A的矩阵，根据迭代公式，每一步高斯赛德尔迭代需要加法乘法共次。而与高斯消去法相比， 肯定会更好。当阶数逐渐增大，迭代的速度要优于高斯消去法。

最后我们来看过松弛算法（SOR）。其中w为松弛参数，针对不同的问题，我们可以调整松弛参数来调整收敛的速度。下图是在不同的松弛参数下迭代误差随迭代次数变化的示意图。不同颜色的曲线代表着不同的松弛参数下的曲线。



图表 1 误差二范数随迭代次数的变化曲线

这张图展示了w从1到1.9的误差二范数随迭代次数增加的变化曲线。w=1到w=1.9依次从上向下排列（当w大于等于2，就会产生剧烈抖动并且发散）。从图中我们可以直观的看到，随着w的逐渐增大，收敛速度越来越快。当w=1.8时2000次迭代后误差模已经小到-15次方量级。而当w=1时，过松弛方法就会退化成为高斯赛德尔迭代。

因此，无论是在迭代精度还是在计算速度上，面对高阶稀疏矩阵，迭代的算法要优于高斯消去法，而过松弛的迭代方法在几种迭代算法中更胜一筹。但是需要寻找合适的松弛参数才能获得更好的收敛速度。

四、求出何时的误差最小

执行附录中程序，即可输出的误差，列表如下：

|  |  |
| --- | --- |
|  | 绝对误差 |
| 1 | 1.3761561334924e-14 |
| 2 | 1.47331799538186e-13 |
| 3 | 6.66359328826971e-13 |
| 4 | 8.60994955886873e-12 |
| 5 | 1.72876280998824e-10 |
| 6 | 8.31952883320608e-10 |
| 7 | 2.2284891572702e-08 |
| 8 | 1.03881457609589e-07 |
| 9 | 1.66558323965414e-06 |
| 10 | 4.36949342139947e-05 |
| 11 | 3.18512163212936e-05 |

我们看到，随着的增加，绝对误差也相应地增加。在数值微分当中我们知道总误差关于微分步长有个极小值，若微分步长太小舍入误差会成为主要误差来源，使得总误差增大。但是在这里，我们没有看到极小值的出现，反而我们看到当从0.1变成0.05时，绝对误差放大了10倍，绝对误差随着步长减小而单调增加了。数值微分的那个现象，以及对应理由在这里不成立，然而，当我们将目光转向矩阵的条件数时，一切便自明了。

|  |  |
| --- | --- |
|  | 条件数 |
| 1 | 530302.499998998 |
| 2 | 8449279.99985676 |
| 3 | 134821259.980659 |
| 4 | 2153877317.45886 |
| 5 | 34434645510.4922 |
| 6 | 550730030029.797 |
| 7 | 8809861144987.75 |
| 8 | 140942568348747 |
| 9 | 2.25488634226904e+15 |
| 10 | 3.60725725331037e+16 |
| 11 | 5.77022291760934e+17 |

可以发现，矩阵的条件数随着单调增加，且增加速度约为当增加1时，条件数变为10倍，这也可以解释为什么绝对误差看上去也是当增加1时，变为10倍。

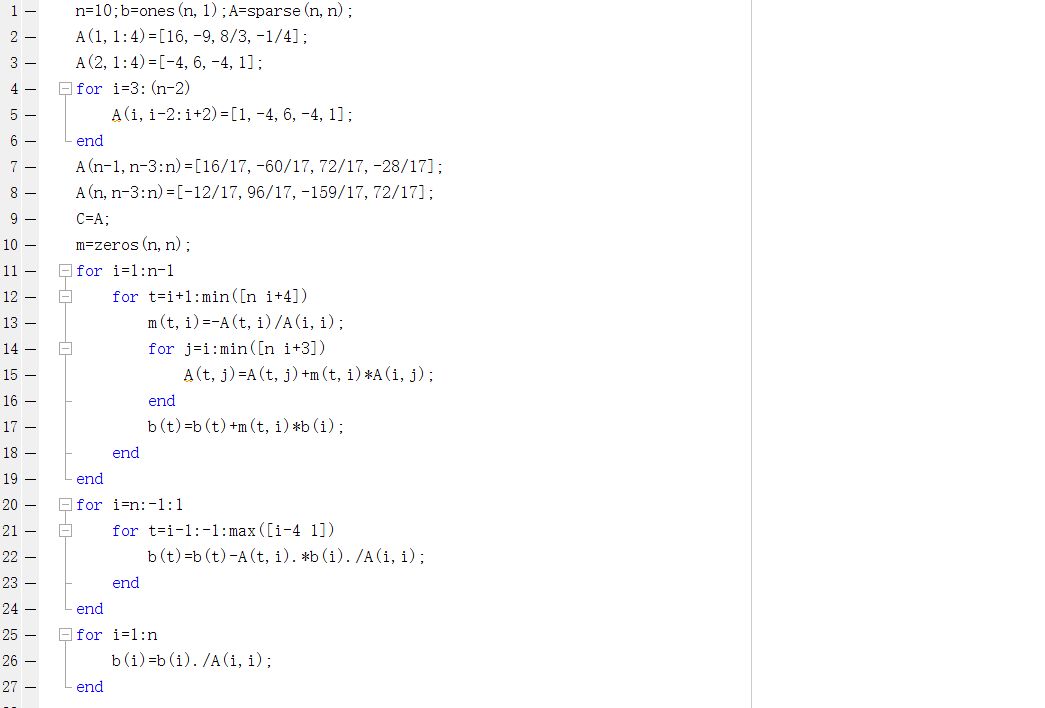
五、跳水员在板上

我们从上一部分知道当，即时误差最小

由题意我们知道，需要对区间中的力作出调整。用下述代码可以画出加上跳水员之后的木板形状。

从中我们可以看到，自由端被下压了0.247728119657767米，偏离度约为10%

六、附录



四中所用代码

function A=discrete(n)

A=sparse(n,n);

A(1,1:4)=[16,-9,8/3,-1/4];

A(2,1:4)=[-4,6,-4,1];

for i=3:(n-2)

A(i,i-2:i+2)=[1,-4,6,-4,1];

end

A(n-1,n-3:n)=[16/17,-60/17,72/17,-28/17];

A(n,n-3:n)=[-12/17,96/17,-156/17,72/17];

function e=erff(n)

A=discrete(n);

L=2;w=0.3;d=0.03;

E=1.3e10;I=w\*d^3/12;

g=9.81;

f=-480\*w\*d\*g;

h=L/n;

B(1:n)=h^4\*f/(E\*I);B=B';

y=A\B;

z=zeros(n,1);

for i=1:n

x=i\*h;

z(i)=(f/(24\*E\*I))\*x^2\*(x^2-4\*L^x+6\*L^2);

end

e=abs(y(n)-z(n));

五中所用代码

function e=erff(n)

A=discrete(n);

L=2;w=0.3;d=0.03;

E=1.3e10;I=w\*d^3/12;

g=9.81;

f=-480\*w\*d\*g;

h=L/n;

B=zeros(n,1);

for i=1:n

if i\*h<1.8

B(i)=h^4\*f/(E\*I);

else

B(i)=h^4\*(f-g\*70/0.2)/(E\*I);

end

end

y=A\B;

z=zeros(n,1);

for i=1:n

z(i)=i\*h;

end

e=abs(y(n));

plot(z,y)