

Solving Systems of Linear Equations Numerically

Introduction

Systems of linear equations of the form

$$a_{11}x + a_{12}y = c_1$$

$$a_{21}x + a_{22}y = c_2$$

are straightforward to solve algebraically. However systems of linear equations of the form

$$a_{11}x + a_{12}y + a_{13}z + a_{14}w = c_1$$

$$a_{21}x + a_{22}y + a_{23}z + a_{24}w = c_2$$

$$a_{31}x + a_{32}y + a_{33}z + a_{34}w = c_3$$

$$a_{41}x + a_{42}y + a_{43}z + a_{44}w = c_4$$

are not so quite straightforward to solve. We will now investigate two iterative methods for solving sets of linear equations.

The Jacobi Method

This method quite simply involves rearranging each equation to make each variable a function of the other variables. Then make an initial guess for each solution and iterate. This is best described in an example.

Example 9A

Perform two iterations of the Jacobi method on the system of equations

$$2x + y = 1$$

$$x + 2y = 1$$

with starting values $x^{(0)} = 0$ and $y^{(0)} = 0$.

You will notice that the starting values are written in the form $x^{(0)}$ instead of x_0 . This notation is adopted because a large system of equations has many variables and we would run out of letters after 26 (not including possible Greek and Russian characters). So the variables in large systems of equations are denoted as x_0, x_1, x_2, \dots , which could cause confusion with the iterates, hence the new notation for the iterates! So an expression such as $x_3^{(5)}$ represents the fifth iterate of the third variable.

- (1) Rearrange each equation in turn so that we have an iterative equation for each variable.

$$x = \frac{1}{2}(1 - y) \Rightarrow x^{(n+1)} = \frac{1}{2}(1 - y^{(n)})$$

$$y = \frac{1}{2}(1 - x) \Rightarrow y^{(n+1)} = \frac{1}{2}(1 - x^{(n)})$$

- (2) Starting with the initial guesses $x^{(0)} = 0$ and $y^{(0)} = 0$, find the values of $x^{(1)}$ and $y^{(1)}$.

$$x^{(1)} = \frac{1}{2}(1-0) = \frac{1}{2}$$

$$y^{(1)} = \frac{1}{2}(1-0) = \frac{1}{2}$$

- (3) Use the values for $x^{(1)}$ and $y^{(1)}$ to find $x^{(2)}$ and $y^{(2)}$.

$$x^{(2)} = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$$

$$y^{(2)} = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$$

To obtain the solutions we continue to iterate until convergence occurs (if convergence should occur!). We will now use DERIVE to perform the calculations.

DERIVE ACTIVITY 9A

Solve the above equations using the Jacobi method, using 10 iterations.

Author

#1: $2x + y = 1$

soLve, #1, variable x

Author

#3: $x + 2y = 1$

soLve, #3, variable y

Author

#5: $[RHS(\#2), RHS(\#4)]$

Simplify #5

Author

#7: $ITERATES(\#6, [x, y], [0, 0], 10)$

approX #6 to give

```
#7: ITERATES [[ [ [ 1 - y, 1 - x ], [x, y], [0, 0], 10 ] ]
      [
        0          0
        0.5        0.5
        0.25       0.25
        0.375      0.375
        0.3125     0.3125
#8:    0.34375     0.34375
        0.328125  0.328125
        0.3359375 0.3359375
        0.33203125 0.33203125
        0.333984375 0.333984375
        0.3330078125 0.3330078125 ] ]
```

```
COMMAND: Author Build Calculus Declare Expand Factor Help Jump solve Manage
          Options Plot Quit Remove Simplify Transfer Unremove move Window approx
Compute time: 0.0 seconds
User                               Free:100% Ins           Derive Algebra
```

The Jacobi method in action

The analytical answers are in fact $x = \frac{1}{3}$, $y = \frac{1}{3}$ and as you can see the iterations do appear to be converging towards $\frac{1}{3}$. In fact, 24 iterations produce 10 significant figure correspondence with the analytical solution. This particular method is slow to converge and so is not generally used, however it is relatively easy to program. However, there is a problem with this method! Lets change the order of the simultaneous equations in the previous example, i.e.

$$x + 2y = 1$$

$$2x + y = 1$$

and rearrange as before to obtain two iterative schemes

$$x = 1 - 2y \Rightarrow x^{(n+1)} = 1 - 2y^{(n)}$$

$$y = 1 - 2x \Rightarrow y^{(n+1)} = 1 - 2x^{(n)}$$

Repeating the Jacobi method with these rearrangements;

Author

```
#9: 1 - 2y
```

```
#10: 1 - 2x
```

```
#11 ITERATES([#9,#10],[x,y],[0,0],10)
```

approx #11 gives us

#10: $1 - 2 \cdot x$

#11: ITERATES([1 - 2·y, 1 - 2·x], [x, y], [0, 0], 10)

```
#12:
      0      0
      1      1
     -1     -1
      3      3
     -5     -5
     11     11
    -21    -21
     43     43
    -85    -85
    171    171
   -341   -341
```

COMMAND: **Author** Build Calculus Declare Expand Factor Help Jump solve Manage
Options Plot Quit Remove Simplify Transfer Unremove move Window approx
Compute time: 0.0 seconds
User Free:100% Ins Derive Algebra
The Jacobi method diverging

Here we can see that this particular iterative scheme diverges.

If one looks at the iterative schemes it is clear that the schemes will diverge as the previous iterate is being multiplied by -2 and then 1 is added. Multiplying any number by a number out of the range [-1,1] will cause growth, hence we would want any iterative scheme to have multipliers in the region [-1,1]. You will notice that this is the case in the first example, the multipliers are each -0.5 we now describe an efficient method for arranging the equations so that we can obtain convergence.

Stopping Divergence.

The previous activity shows that the order in which the equations are placed is very important if you wish to avoid convergence. Writing the linear equations in matrix form helps shed some light onto how we can avoid convergence.

The previous examples in matrix form are

$$\begin{aligned} 2x + y = 1 \\ x + 2y = 1 \end{aligned} \Rightarrow \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{aligned} x + 2y = 1 \\ 2x + y = 1 \end{aligned} \Rightarrow \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The diagonal elements (down left to right) in the matrix of the first set of equations are larger than their off diagonal partners, the opposite is true for the second set of equations. When the diagonal elements dominate (the absolute values are greater) the

off diagonal elements, we have diagonal dominance. Let us look at a 3x3 case to see why we need diagonal dominance,

$$a_1x + b_1y + c_1z = d_1$$

$$a_2x + b_2y + c_2z = d_2$$

$$a_3x + b_3y + c_3z = d_3$$

rearranged gives

$$x = d_1 - \frac{b_1}{a_1}y - \frac{c_1}{a_1}z$$

$$y = d_2 - \frac{a_2}{b_2}x - \frac{c_2}{b_2}z$$

$$z = d_3 - \frac{a_3}{c_3}x - \frac{b_3}{c_3}y$$

The diagonal terms are a_1, b_2 & c_3 , in order that each multiplier lies in the range $[-1,1]$ then $|a_1| > |b_1|$ and $|c_1|, |b_2| > |a_2|$ and $|c_2|, |c_3| > |a_3|$ and $|b_3|$. In other words, the modulus of the diagonal terms must be greater than the modulus of the off diagonal terms.

In fact the story is a little more complicated that this, to ensure convergence the condition

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \text{ for all } i$$

and in at least one case

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|$$

must apply.

If these conditions are not met, the iterative equations **may** still converge. If these conditions are met the iterative equations will **definitely** converge.

Exercise

Rearrange the following set of linear equations so that we have diagonal dominance and use the Jacobi method to find the solutions to 2 decimal places, starting with $[0,0,0]$

$$x + y + 3z = 2$$

$$2x - y + z = 4$$

$$-x + 5y - 2z = 4$$

The Gauss Seidel Method

This method is a more efficient method than the Jacobi method, and it is very similar to the Jacobi method. It differs from the Jacobi method in that it always uses the most up to date iterate for any particular variable. Let's use the first example as a demonstration.

$$2x + y = 1$$

$$x + 2y = 1$$

with starting values $x^{(0)} = 0$ and $y^{(0)} = 0$.

The Jacobi iterative schemes are

$$x = \frac{1}{2}(1 - y) \Rightarrow x^{(n+1)} = \frac{1}{2}(1 - y^{(n)})$$

$$y = \frac{1}{2}(1 - x) \Rightarrow y^{(n+1)} = \frac{1}{2}(1 - x^{(n)})$$

but when we calculate $y^{(n+1)}$ we use the $x^{(n)}$ iterate in calculating it. However, the iterate $x^{(n+1)}$ has already been calculated, so why not use this as it is likely to be more accurate than $x^{(n)}$. The iterative scheme using the Gauss Seidel method will be

$$x^{(n+1)} = \frac{1}{2}(1 - y^{(n)})$$

$$y^{(n+1)} = \frac{1}{2}(1 - x^{(n+1)})$$

Lets try it!

Starting with the initial guesses $x^{(0)} = 0$ and $y^{(0)} = 0$, find the values of $x^{(1)}$ and $y^{(1)}$.

$$x^{(1)} = \frac{1}{2}(1 - y^{(0)}) = \frac{1}{2}(1 - 0) = \frac{1}{2}$$

$$y^{(1)} = \frac{1}{2}(1 - x^{(1)}) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$$

Use the values for $x^{(1)}$ and $y^{(1)}$ to find $x^{(2)}$ and $y^{(2)}$.

$$x^{(2)} = \frac{1}{2}\left(1 - \frac{1}{4}\right) = \frac{3}{8} = 0.375$$

$$y^{(2)} = \frac{1}{2}\left(1 - \frac{3}{8}\right) = \frac{5}{16} = 0.3125$$

The analytical answers are $x = 0.333333$ and $y = 0.333333$ after two iterations of the Jacobi method we have approximate solution 0.25 and 0.25 and after two iteration of the Gauss Seidel method we have approximate solutions 0.375 and 0.3125. As we can see a more accurate approximation from the Gauss Seidel method for the same number of iteration.

Up until the time of writing these notes, I have not found a straightforward way to program DERIVE to perform the Gauss Seidel method on any set of linear equations. N.B. it is very easy to program a spreadsheet to do this!!

However, below is a program that will perform the Gauss Seidel method for a 3x3 system of equations.

DERIVE ACTIVITY 9B

Author

```
#1: LIM(f,[x,y,z],[a,b,c])
#2: LIM(g,[x,y,z],[#1,b,c])
#3: LIM(h,[x,y,z],[#1,#2,c])
#4: [#1,#2,#3]
#5: [x0:=,y0:=,z0:=]
#6: ITERATES(#4,[a,b,c],[x0,y0,z0],n)
#7: G_SEID_AUX(f,g,h,x0,y0,z0,n):=#6
#8: SOLVE(p,x)SUB1
#9: SOLVE(q,y)SUB1
#10: SOLVE(r,z)SUB1
#11: G_SEID_AUX(RHS(#8),RHS(#9),RHS(#10),x0,y0,z0,n)
#12: G_SEID(p,q,r,x,y,z,x0,y0,z0,n):=#11
```

We will now use the function G_SEID() of the set of equations

$$2x + y + z = 4$$

$$x + 2y + z = 4$$

$$x + y + 3z = 5$$

starting with [0,0,0] and iterating 10 times.

We have diagonal dominance and we can see from inspection that the solutions are $x=1$, $y=1$ and $z=1$.

```
#13: G_SEID(2x + y + z = 4, x + 2y + z = 4, x + y + 3z = 5, x, y, z, 0, 0, 0, 10)
      apprx #13 to give
```

#13: G_SEID(2-x + y + z = 4, x + 2·y + z = 4, x + y + 3·z = 5, x, y, z, 0, 0, 0,

0	0	0
2	1	0.6666666666
1.1666666666	1.0833333333	0.9166666666
1	1.0416666666	0.9861111111
0.9861111111	1.0138888888	1
#14: 0.9930555555	1.0034722222	1.001157407
0.9976851851	1.000578703	1.000578703
0.9994212962	1	1.000192901
0.9999035493	0.9999517746	1.000048225
1	0.9999758873	1.000008037
1.000008037	0.9999919624	1

COMMAND: **Author** Build Calculus Declare Expand Factor Help Jump solve Manage
 Options Plot Quit Remove Simplify Transfer Unremove move Window approx
 Enter option
 User Free:72% Ins Derive Algebra

The G_SEID() function at work

Exercise

- Solve the following system of linear equations using the Gauss Seidel Method

$$-x + 3y + 2z = 5$$

$$3x - 0.5y + 2z = 2$$

$$-x - y + 5z = 3$$

- Write a DERIVE function that will solve 4x4 sets of linear equations.