

Iterative Methods for Linear Systems

We consider three iterative methods for solving equations of the form $Ax=b$ where A is $n \times n$ and the equation has a unique solution.

```
> restart;
> with(LinearAlgebra):
> libname:="c:/nalib", libname;
libname := "/nalib", "/Library/Frameworks/Maple.framework/Versions/14/lib",
"/Library/Frameworks/Maple.framework/Versions/14/toolbox/NAG/lib"
> with(numanal);
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,
clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,
falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,
hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,
muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir,
secant, secant_dir, steffensen, steffensen_dir]
```

Jacobi Iterative Method

Our goal here is to solve the linear system $Ax=b$.

```
> A:=Matrix(4,4,[[10,-1,2,0],[-1,11,-1,3],[2,-1,10,-1],[0,3,-1,8]]);
```

$$A := \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}$$

We check to see if there is a solution by taking the determinant of A .

```
> if Determinant(A)=0 then print("There is no unique solution.")
else print("There is a unique solution.") fi;
"There is a unique solution."
```

If there is a solution, we go on to find it. We treat b as a vector.

```
> b:=Vector(4,[6,25,-11,15]);
```

$$b := \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$$

All three methods studied here assume that none of the diagonal elements is 0. If one or more of them are, since there is a unique solution, you can correct that by doing row swaps. You must also rewrite b using a similar adjustment.

We do the iteration using the library procedure **jacobi**, based on equation (7.1) on page 308 of the text. We choose the **0** vector as a first approximation. Let's first check the directions for **jacobi**.

```
> jacobi_dir();  
jacobi returns an approximation to a solution of a vector equation.
```

It uses the built-in Maple library LinearAlgebra.

The arguments for **jacobi** are:

- (1) the coefficient matrix (must be square)
- (2) the right hand side vector
- (3) the initial approximation vector
- (4) tolerance
- (5) maximum number of iterations
- (6) variable for returning the approximate solution

If assigning the result to a variable, have the variable and the 6th argument the same.

If **v** is the variable for returning the approximate solution and has already been given a value, the procedure should be preceded by the statement:

```
v:='v'
```

We execute the procedure.

```
> v:=jacobi(A,b,Vector(4,[0,0,0,0]),10^(-6),100,v);  
n      x  
-      -  
0      [ 0.00000000, 0.00000000, 0.00000000, 0.00000000 ]  
1      [ 0.60000000, 2.27272727, -1.10000000, 1.87500000 ]  
2      [ 1.04727273, 1.71590909, -0.80522727, 0.88522727 ]  
3      [ 0.93263636, 2.05330578, -1.04934091, 1.13088068 ]  
4      [ 1.01519876, 1.95369576, -0.96810863, 0.97384272 ]  
5      [ 0.98899130, 2.01141472, -1.01028590, 1.02135051 ]  
6      [ 1.00319865, 1.99224126, -0.99452174, 0.99443374 ]  
7      [ 0.99812847, 2.00230688, -1.00197223, 1.00359431 ]  
8      [ 1.00062513, 1.99867030, -0.99903558, 0.99888839 ]  
9      [ 0.99967415, 2.00044767, -1.00036916, 1.00061919 ]  
10     [ 1.00011860, 1.99976795, -0.99982814, 0.99978598 ]  
11     [ 0.99994242, 2.00008478, -1.00006833, 1.00010850 ]  
12     [ 1.00002214, 1.99995896, -0.99996916, 0.99995967 ]  
13     [ 0.99998973, 2.00001582, -1.00001256, 1.00001924 ]  
14     [ 1.00000410, 1.99999268, -0.99999444, 0.99999250 ]  
15     [ 0.99999816, 2.00000292, -1.00000230, 1.00000344 ]  
16     [ 1.00000075, 1.99999868, -0.99999899, 0.99999862 ]  
17     [ 0.99999967, 2.00000054, -1.00000042, 1.00000062 ]  
18     [ 1.00000014, 1.99999976, -0.99999982, 0.99999975 ]
```

The solution vector is

```
v := [ 1.000000138 1.999999763 -0.9999998179 0.9999997460 ]
```

Suppose we set a maximum of 10 iterations.

```
> v:='v';
```

```
v := v
```

```
> v:=jacobi(A,b,Vector(4,[0,0,0,0]),10^(-6),10,v);  
n      x  
-      -  
0      [ 0.00000000, 0.00000000, 0.00000000, 0.00000000 ]  
1      [ 0.60000000, 2.27272727, -1.10000000, 1.87500000 ]  
2      [ 1.04727273, 1.71590909, -0.80522727, 0.88522727 ]  
3      [ 0.93263636, 2.05330578, -1.04934091, 1.13088068 ]
```

```

4 [ 1.01519876, 1.95369576, -0.96810863, 0.97384272]
5 [ 0.98899130, 2.01141472, -1.01028590, 1.02135051]
6 [ 1.00319865, 1.99224126, -0.99452174, 0.99443374]
7 [ 0.99812847, 2.00230688, -1.00197223, 1.00359431]
8 [ 1.00062513, 1.99867030, -0.99903558, 0.99888839]
9 [ 0.99967415, 2.00044767, -1.00036916, 1.00061919]
10 [ 1.00011860, 1.99976795, -0.99982814, 0.99978598]
Maximum Number of Iterations Exceeded.
v:=

```

Gauss-Seidel Iterative Method

The Gauss-Seidel method adjusts the Jacobi method to use the latest information. We use the same system as above. We do the iteration using equation (7.2) on page 309 of the text. We check the directions for **gaussseidel** and then execute the procedure.

```
> gaussseidel_dir();
```

gaussseidel returns an approximation to a solution of a vector equation.

It uses the Maple built-in library LinearAlgebra.

The arguments for gauss-seidel are:
(1)the coefficient matrix (must be square)
(2)the right hand side vector
(3)the initial approximation vector
(4)tolerance
(5)maximum number of iterations
(6)variable for returning the approximate solution

If assigning the result to a variable, have the variable and the 6th argument the same.

If v is the variable for returning the approximate solution and has already been given a value, the procedure should be preceded by the statement:
v:='v'

```
> v:='v';
v:=v
> v:=gaussseidel(A,b,Vector(4,[0,0,0,0]),10^(-6),100,v);
n      x
--      -
0 [ 0.00000000, 0.00000000, 0.00000000, 0.00000000]
1 [ 0.60000000, 2.32727273, -0.98727273, 0.87886364]
2 [ 1.03018182, 2.03693802, -1.01445620, 0.98434122]
3 [ 1.00658504, 2.00355502, -1.00252738, 0.99835095]
4 [ 1.00086098, 2.00029825, -1.00030728, 0.99984975]
5 [ 1.00009128, 2.00002134, -1.00003115, 0.99998810]
6 [ 1.00000836, 2.00000117, -1.00000274, 0.99999922]
7 [ 1.00000067, 2.00000002, -1.00000021, 0.99999996]
8 [ 1.00000004, 2.00000000, -1.00000001, 1.00000000]
```

The solution vector is

$$v := \begin{bmatrix} 1.000000044 & 1.999999995 & -1.000000013 & 1.000000000 \end{bmatrix}$$

We see that the solution converges much more quickly here, but that will not always be the case.

SOR Iterative Method

We do this iteration using the equation at the top of page 313 of the text. We begin with a new system.

```
> A:=Matrix(3,3,[[4,3,0],[3,4,-1],[0,-1,4]]);
```

$$A := \begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}$$

```
> b:=Vector(3,[24,30,-24]);
```

$$b := \begin{bmatrix} 24 \\ 30 \\ -24 \end{bmatrix}$$

```
> SOR_dir();
```

SOR returns an approximation to a solution of a vector equation.

It uses the built-in Maple library LinearAlgebra.

The arguments for SOR are:

- (1) the coefficient matrix (must be square)
- (2) the right hand side vector
- (3) the parameter omega
- (4) the initial approximation vector
- (5) tolerance
- (6) maximum number of iterations
- (7) variable for returning the approximate solution

If assigning the result to a variable, have the variable and the 7th argument the same.

If v is the variable for returning the approximate solution and has already been given a value, the procedure should be preceded by the statement:

```
v:='v'
```

A choice of omega = 1 is the Gauss-Seidel method.

```
> v:='v';
```

```
v:=v
```

```
> v:=SOR(A,b,1,Vector(3,[1,1,1]),10^(-6),100,v);
```

n	x
0	[1.00000000, 1.00000000, 1.00000000]
1	[5.25000000, 3.81250000, -5.04687500]
2	[3.14062500, 3.88281250, -5.02929688]
3	[3.08789062, 3.92675781, -5.01831055]
4	[3.05493164, 3.95422363, -5.01144409]
5	[3.03433228, 3.97138977, -5.00715256]
6	[3.02145767, 3.98211861, -5.00447035]
7	[3.01341104, 3.98882413, -5.00279397]
8	[3.00838190, 3.99301508, -5.00174623]
9	[3.00523869, 3.99563443, -5.00109139]
10	[3.00327418, 3.99727152, -5.00068212]
11	[3.00204636, 3.99829470, -5.00042632]
12	[3.00127898, 3.99893418, -5.00026646]
13	[3.00079936, 3.99933386, -5.00016654]

```

14 [ 3.00049960, 3.99958366, -5.00010408]
15 [ 3.00031225, 3.99973979, -5.00006505]
16 [ 3.00019516, 3.99983737, -5.00004066]
17 [ 3.00012198, 3.99989836, -5.00002541]
18 [ 3.00007624, 3.99993647, -5.00001588]
19 [ 3.00004765, 3.99996029, -5.00000993]
20 [ 3.00002978, 3.99997518, -5.00000620]
21 [ 3.00001861, 3.99998449, -5.00000388]
22 [ 3.00001163, 3.99999031, -5.00000242]
23 [ 3.00000727, 3.99999394, -5.00000152]
24 [ 3.00000454, 3.99999622, -5.00000094]
25 [ 3.00000284, 3.99999764, -5.00000059]
26 [ 3.00000178, 3.99999852, -5.00000037]
27 [ 3.00000111, 3.99999908, -5.00000023]

```

The solution vector is

$$v := \begin{bmatrix} 3.000001110 & 3.999999075 & -5.000000230 \end{bmatrix}$$

A choice of ω such that $1 < \omega < 2$ will often accelerate the convergence. We try $\omega = 1.25$ here.

```
> v:='v';
```

```
v := v
```

```
> v:=SOR(A,b,1.25,Vector(3,[1,1,1]),10^(-6),100,v);
```

```

n      x
--      -
0 [ 1.00000000, 1.00000000, 1.00000000]
1 [ 6.31250000, 3.51953125, -6.65014648]
2 [ 2.62231445, 3.95852661, -4.60042381]
3 [ 3.13330269, 4.01026464, -5.09668635]
4 [ 2.95705123, 4.00748383, -4.97348972]
5 [ 3.00372110, 4.00292497, -5.00571352]
6 [ 2.99632756, 4.00092620, -4.99828218]
7 [ 3.00004980, 4.00025858, -5.00034865]
8 [ 2.99974513, 4.00006534, -4.99989242]
9 [ 3.00000246, 4.00001498, -5.00002222]
10 [ 2.99998534, 4.00000306, -4.99999349]
11 [ 3.00000080, 4.00000052, -5.00000146]
12 [ 2.99999931, 4.00000006, -4.99999961]
13 [ 3.00000012, 4.00000000, -5.00000010]

```

The solution vector is

$$v := \begin{bmatrix} 3.000000120 & 3.999999996 & -5.000000096 \end{bmatrix}$$

Now we will be using the [LinearSolve](#) command with the methods **jacobi**, **gaussseidel**, and **SOR(ω)** from the [NumericalAnalysis](#) package.

```
> with(Student[NumericalAnalysis]);
```

```

[AbsoluteError, AdamsBashforth, AdamsBashforthMoulton, AdamsMoulton, AdaptiveQuadrature,
AddPoint, ApproximateExactUpperBound, ApproximateValue, BackSubstitution, BasisFunctions,
Bisection, CubicSpline, DataPoints, Distance, DividedDifferenceTable, Draw, Euler, EulerTutor,
ExactValue, FalsePosition, FixedPointIteration, ForwardSubstitution, Function,
InitialValueProblem, InitialValueProblemTutor, Interpolant, InterpolantRemainderTerm,
IsConvergent, IsMatrixShape, IterativeApproximate, IterativeFormula, IterativeFormulaTutor,
LeadingPrincipalSubmatrix, LinearSolve, LinearSystem, MatrixConvergence,

```

MatrixDecomposition, MatrixDecompositionTutor, ModifiedNewton, NevilleTable, Newton, NumberOfSignificantDigits, PolynomialInterpolation, Quadrature, RateOfConvergence, RelativeError, RemainderTerm, Roots, RungeKutta, Secant, SpectralRadius, Steffensen, Taylor, TaylorPolynomial, UpperBoundOfRemainderTerm, VectorLimit]

We will first apply the **Jacobi method** to solving $Ax=b$ where A and B are as below.

```
> A:=Matrix(3,3,[[16,3,0],[3,4,-1],[0,-1,16]]);
```

$$A := \begin{bmatrix} 16 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 16 \end{bmatrix}$$

```
> b:=Vector(3,[24.,30,-24]);
```

$$b := \begin{bmatrix} 24. \\ 30 \\ -24 \end{bmatrix}$$

I need to list at least one number in floating point form to get floating point results.

```
> x1:=LinearSolve(A,b,method=jacobi,initialapprox=Vector(3,[[0],[0],[0]]),tolerance=10^(-6),maxiterations=100);
```

$$x1 := \begin{bmatrix} 0.1666665012 \\ 7.111108585 \\ -1.055555500 \end{bmatrix}$$

Note that the default for **initialapprox** is the zero vector, for **tolerance** is $\frac{1}{10000}$, and for **maxiterations** is 20. We now use the **Gauss-Seidel** method.

```
> x2:=LinearSolve(A,b,method=gaussseidel,tolerance=10^(-6));
```

$$x2 := \begin{bmatrix} 0.1666669806 \\ 7.111110850 \\ -1.055555572 \end{bmatrix}$$

Next we want to use the **SOR** method. We first check to see if we can find an optimal ω . We use the command [IsMatrixShape](#) to determine if the matrix A is **positive definite** and **tridiagonal**.

```
> IsMatrixShape(A,'positivedefinite');
```

true

```
> IsMatrixShape(A,'tridiagonal');
```

true

We do an **LDU decomposition**.

```
> LL,DD,UU:=MatrixDecomposition(A,method=LDU);
```

$$LL, DD, UU := \begin{bmatrix} 1 & 0 & 0 \\ \frac{3}{16} & 1 & 0 \\ 0 & -\frac{16}{55} & 1 \end{bmatrix}, \begin{bmatrix} 16 & 0 & 0 \\ 0 & \frac{55}{16} & 0 \\ 0 & 0 & \frac{864}{55} \end{bmatrix}, \begin{bmatrix} 1 & \frac{3}{16} & 0 \\ 0 & 1 & -\frac{16}{55} \\ 0 & 0 & 1 \end{bmatrix}$$

We find the matrix T_j and its spectral radius.

```
> Tj:=MatrixInverse(DD).(LL+UU);
```

$$T_j := \begin{bmatrix} \frac{1}{8} & \frac{3}{256} & 0 \\ \frac{3}{55} & \frac{32}{55} & -\frac{256}{3025} \\ 0 & -\frac{1}{54} & \frac{55}{432} \end{bmatrix}$$

```
> s:=SpectralRadius(Tj);
```

$$s := 0.5866150145$$

We find the optimal ω by using the **SOR Method Convergence Theorem**.

```
> omega:=2/(1+sqrt(1-s^2));
```

$$\omega := 1.105054229$$

We solve the system using **SOR** with this ω .

```
> x3:=LinearSolve(A,b,method=SOR(omega),tolerance=10^(-6));
```

Error, invalid input: SOR expects its 1st argument, A, to be of type Matrix, but received 1.105054229

OK, the problem is that, in Maple 14 at least, that SOR is taken to be the command from the **numanal** package, so we need to reset the variable.

```
> SOR:='SOR';
```

$$SOR := SOR$$

```
> x3:=LinearSolve(A,b,method=SOR(omega),tolerance=10^(-6));
```

$$x3 := \begin{bmatrix} 0.1666667490 \\ 7.111111008 \\ -1.055555537 \end{bmatrix}$$