

# Pseudocodes L<sup>A</sup>T<sub>E</sub>X

Daniel Hernandez, Samuel Villegas, Jose Blanco, Neller Pellegrino

June 2, 2022

---

**Algorithm 1** Aitken

---

```
1: READ f,g,x0,tolerance,nMax
2:  $x1 \leftarrow g(x0)$ 
3:  $x2 \leftarrow g(x1)$ 
4:  $x3 \leftarrow g(x2)$ 
5:  $xi \leftarrow x1 - ((x2 - x1)^2)/(x3 - 2 * x2 + x1)$ 
6:  $fxi \leftarrow f(xi)$ 
7:  $error \leftarrow tolerance + 1$ 
8:  $counter \leftarrow 0$ 
9: while  $fxi \neq 0$  AND  $error > tolerance$  AND  $counter < nMax$  do
10:    $x3Aux \leftarrow x3$ 
11:    $x2Aux \leftarrow x2$ 
12:    $x3 \leftarrow g(x3)$ 
13:    $x2 \leftarrow x3Aux$ 
14:    $x1 \leftarrow x2Aux$ 
15:    $xiAux \leftarrow xi$ 
16:    $xi \leftarrow x1 - ((x2 - x1)^2)/(x3 - 2 * x2 + x1)$ 
17:    $fxi \leftarrow f(xi)$ 
18:    $error \leftarrow |xi - xiAux|$ 
19:    $counter \leftarrow counter + 1$ 
20: end while
21: if  $fxi = 0$  then
22:   A root has been found and it is xi
23: else if  $error \leq tolerance$  then
24:   one approach has been found and it is xi
25: else
26:   The method fails with the maximum number of iterations given
27: end if
```

---

---

**Algorithm 2** Bisection

---

```
1: READ  $f, left, right, tolerance, niter$ 
2:  $fRight \leftarrow f(right)$ 
3:  $fLeft \leftarrow f(left)$ 
4: if  $fRight = 0$  then
5:   right is a root
6: else if  $fLeft = 0$  then
7:   left is a root
8: else if  $fLeft * fRight < 0$  then
9:    $xmid \leftarrow (left + right)/2$ 
10:   $fXmid \leftarrow f(xmid)$ 
11:   $counter \leftarrow 1$ 
12:   $error \leftarrow tolerance + 1$ 
13:  while  $error > tolerance$  AND  $fXmid \neq 0$  AND  $counter < niter$ 
14:    do
15:      if  $fLeft * fXmid < 0$  then
16:         $right \leftarrow xmid$ 
17:         $fRight \leftarrow fXmid$ 
18:      else
19:         $left \leftarrow xmid$ 
20:         $fLeft \leftarrow fXmid$ 
21:      end if
22:       $xAux \leftarrow xmid$ 
23:       $xmid \leftarrow (right + left)/2$ 
24:       $fXmid \leftarrow f(xmid)$ 
25:       $error \leftarrow |xmid - xAux|$ 
26:       $counter \leftarrow counter + 1$ 
27:    end while
28:    if  $fXmid = 0$  then
29:       $xmid$  is a root
30:    else if  $error < tolerance$  then
31:       $xmid$  is an approximation with tolerance
32:    else
33:      The method fails in  $niter$  iterations
34:    end if
35:  else
36:    Bad range
37:  end if
```

---

---

**Algorithm 3** Crout

---

```
1: READ  $A, B$ 
2:  $n \leftarrow \text{length of } A$ 
3:  $lower \leftarrow$  matrix  $n \times n$  initialized in zeros
4:  $upper \leftarrow$  identity matrix  $n \times n$ 
5: for  $j = 0 \dots n$  do
6:   for  $i = j \dots n$  do
7:      $sum \leftarrow 0$ 
8:     for  $k = 0 \dots j$  do
9:        $sum \leftarrow sum + lower[i][k] * upper[k][j]$ 
10:    end for
11:     $lower[i][j] \leftarrow A[i][j] - sum$ 
12:  end for
13:  for  $i = j + 1 \dots n$  do
14:     $sum \leftarrow 0$ 
15:    for  $k = 0 \dots j$  do
16:       $sum \leftarrow sum + lower[j][k] * upper[k][i]$ 
17:    end for
18:     $upper[j][i] \leftarrow (A[j][i] - sum) / lower[j][j]$ 
19:  end for
20: end for
21:  $Z \leftarrow \text{regressiveSubstitution}(lower|B)$ 
22:  $X \leftarrow \text{progressiveSubstitution}(upper|Z)$ 
```

---

---

**Algorithm 4** Doolittle

---

```
1:  $n \leftarrow \text{length of } A$ 
2:  $lower \leftarrow \text{matrix } nxn \text{ initialized in zeros}$ 
3:  $upper \leftarrow \text{identity matrix } nxn$ 
4: for  $i = 0 \dots n$  do
5:   for  $k = i \dots n$  do
6:      $sum \leftarrow 0$ 
7:     for  $j = 0 \dots i$  do
8:        $sum \leftarrow lower[i][j] * upper[j][k]$ 
9:     end for
10:     $upper[i][k] \leftarrow A[i][k] - sum$ 
11:  end for
12:  for  $k = i \dots n$  do
13:    if  $i = k$  then
14:       $lower[i][i] \leftarrow 1$ 
15:    else
16:       $sum \leftarrow 0$ 
17:      for  $j = 0 \dots i$  do
18:         $sum \leftarrow sum + lower[k][j] * upper[j][i]$ 
19:      end for
20:       $lower[k][i] \leftarrow (A[k][i] - sum) / upper[i][i]$ 
21:    end if
22:  end for
23: end for
24:  $Z \leftarrow \text{regressiveSubstitution}(lower|B)$ 
25:  $X \leftarrow \text{progressiveSubstitution}(upper|Z)$ 
```

---

---

**Algorithm 5** Incremental Search

---

```
1: READ  $f, x_0, \text{delta}, \text{niter}$ 
2:  $fx_0 \leftarrow f(x_0)$ 
3: if  $fx_0 = 0$  then
4:    $x_0$  is a root
5: else
6:    $x_1 \leftarrow x_0 + \text{delta}$ 
7:    $\text{counter} \leftarrow 1$ 
8:    $fx_1 \leftarrow f(x_1)$ 
9:   while  $fx_0 * fx_1 > 0$  AND  $\text{counter} > \text{niter}$  do
10:     $x_0 \leftarrow x_1$ 
11:     $fx_0 \leftarrow fx_1$ 
12:     $x_1 \leftarrow x_0 + \text{delta}$ 
13:     $fx_1 \leftarrow f(x_1)$ 
14:     $\text{counter} \leftarrow \text{counter} + 1$ 
15:   end while
16:   if  $fx_1 = 0$  then
17:     $x_1$  is a root
18:   else if  $fx_0 * fx_1 < 0$  then
19:    there is a root between  $x_0$  and  $x_1$ 
20:   else
21:    the method fails in niter iterations
22:   end if
23: end if
```

---

---

**Algorithm 6** Muller

---

```
1: READ  $f, x0, tolerance, nMax$ 
2:  $fx0 \leftarrow f(x0)$ 
3:  $fx1 \leftarrow f(x1)$ 
4:  $x1 \leftarrow (x0 + x1)/2$ 
5:  $fx2 \leftarrow f(x2)$ 
6:  $h0 \leftarrow x1 - x0$ 
7:  $h1 \leftarrow x2 - x1$ 
8:  $delta0 \leftarrow (fx1 - fx0)/h0$ 
9:  $delta1 \leftarrow (fx2 - fx1)/h1$ 
10:  $a \leftarrow (delta1 - delta0)/(h1 - h0)$ 
11:  $b \leftarrow a * h1 + delta1$ 
12:  $c \leftarrow fx2$ 
13:  $xi \leftarrow x2 + (-2 * c)/(b + (b/|b|) * \sqrt{b^2 - 4 * a * c})$ 
14:  $fxi \leftarrow f(xi)$ 
15:  $error \leftarrow tolerance + 1$ 
16:  $counter \leftarrow 0$ 
17: while  $fx1 \neq 0$  AND  $error > tolerance$  AND  $counter < nMax$  do
18:    $x2Aux \leftarrow x2$ 
19:    $x1Aux \leftarrow x1$ 
20:    $x2 \leftarrow xi$ 
21:    $x1 \leftarrow x2Aux$ 
22:    $x0 \leftarrow x1Aux$ 
23:    $fx0 \leftarrow f(x0)$ 
24:    $fx1 \leftarrow f(x1)$ 
25:    $fx2 \leftarrow f(x2)$ 
26:    $h0 \leftarrow x1 - x0$ 
27:    $h1 \leftarrow x2 - x1$ 
28:    $delta0 \leftarrow (fx1 - fx0)/h0$ 
29:    $delta1 \leftarrow (fx2 - fx1)/h1$ 
30:    $a \leftarrow (delta1 - delta0)/(h1 - h0)$ 
31:    $b \leftarrow a * h1 + delta1$ 
32:    $c \leftarrow fx2$ 
33:    $xi \leftarrow x2 + (-2 * c)/(b + (b/|b|) * \sqrt{b^2 - 4 * a * c})$ 
34:    $fxi \leftarrow f(xi)$ 
35:    $error \leftarrow tolerance + 1$ 
36:    $counter \leftarrow counter + 1$ 
37: end while
38: if  $fxi = 0$  then
39:   A root has been found and it is  $xi$ 
40: else if  $error \leq tolerance$  then
41:   One approach has been found and it is  $xi$ 
42: else
43:   The method fails with the maximum number of iterations given
44: end if
```

---

---

**Algorithm 7** Multiple root

---

```
1: READ  $f, f1, f2, x0, tolerance, nMax$ 
2:  $xi \leftarrow x0$ 
3:  $fxi \leftarrow f(xi)$ 
4: if  $fx = 0$  then
5:   A root has been found and it is  $xi$ 
6: else
7:    $counter \leftarrow 0$ 
8:    $f1xi \leftarrow f1(xi)$ 
9:    $f2xi \leftarrow f2(xi)$ 
10:   $error \leftarrow tolerance + 1$ 
11:   $det \leftarrow (f1xi^2) - (fxi * f2xi)$ 
12:  while  $fxi \neq 0$  AND  $error > tolerance$  AND  $counter < nMax$  do
13:     $xiAux \leftarrow xi$ 
14:     $xi \leftarrow xi - ((fxi * f1xi) / ((f1xi^2) - (fxi * f2xi)))$ 
15:     $fxi \leftarrow f(xi)$ 
16:     $f1xi \leftarrow f1(xi)$ 
17:     $f2xi \leftarrow f2(xi)$ 
18:     $error \leftarrow |xi - xiAux|$ 
19:     $det \leftarrow (f1xi^2) - (fxi * f2xi)$ 
20:     $counter \leftarrow counter + 1$ 
21:  end while
22:  if  $fxi = 0$  then
23:    A root has been found and it is  $xi$ 
24:  else if  $error \leq tolerance$  then
25:    One approach has been found and it is  $xi$ 
26:  else
27:    The method fails with the maximum number of iterations given
28:  end if
29: end if
```

---

---

**Algorithm 8** Newton

---

```
1: READ  $f, fder, tolerance, x0, niter$ 
2:  $fx \leftarrow f(x0)$ 
3:  $dfx \leftarrow fder(x0)$ 
4:  $counter \leftarrow 1$ 
5:  $error \leftarrow tolerance + 1$ 
6: while  $error > tolerance$  AND  $fx \neq 0$  AND  $counter < niter$  do
7:    $x1 \leftarrow x0 - (fx/dfx)$ 
8:    $fx \leftarrow f(x1)$ 
9:    $dfx \leftarrow fder(x1)$ 
10:   $error \leftarrow |x1 - x0|$ 
11:   $x0 \leftarrow x1$ 
12:   $counter \leftarrow counter + 1$ 
13: end while
14: if  $fx = 0$  then  $x0$  is a root
15: else if  $error < tolerance$  then
16:    $x1$  is a root approximation with tolerance  $tolerance$ 
17: elseif method fails in  $niter$  iterations
18: end if
```

---

---

**Algorithm 9** Vandermonde

---

```
1: READ  $x, y$ 
2:  $n \leftarrow findVectorLength(x)$ 
3:  $A \leftarrow generateMatrixofOnes(n)$ 
4: for  $i = 1 \dots n$  do
5:   for  $j = 1 \dots n - 1$  do
6:      $A_{ij} \leftarrow x_i^{n-j}$ 
7:   end for
8: end for
9:  $xSolution \leftarrow solveLinearEquation(A, y)$ 
10:  $xSolution$  is the coefficient vector
```

---



---

**Algorithm 10** gaussianElimination

---

```
1: READ  $A, b$ 
2:  $m \leftarrow concatenateMatrices(A, b)$ 
3: for  $i = 1 \dots n - 1$  do
4:   if  $m_{ii} == 0$  then
5:     Error when executing the method: division by 0
6:     return
7:   end if
8:   for  $j = 1 + 1 \dots n$  do
9:      $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$ 
10:  end for
11: end for
12:  $x \leftarrow backwardSubstitution(m)$ 
```

---

---

**Algorithm 11** partialPivoting

---

```
1: READ  $A, b$ 
2:  $m \leftarrow concatenateMatrices(A, b)$ 
3: for  $i = 1 \dots n - 1$  do
4:    $ChangeRows(m, i)$ 
5:   if  $m_{ii} == 0$  then
6:     Error when executing the method: division by 0
7:     return
8:   end if
9:   for  $j = 1 + 1 \dots n$  do
10:     $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$ 
11:  end for
12: end for
13:  $x \leftarrow backwardSubstitution(m)$ 
```

---

---

**Algorithm 12** totalPivoting

---

```
1: READ  $A, b$ 
2:  $m \leftarrow concatenateMatrices(A, b)$ 
3: for  $i = 1 \dots n - 1$  do
4:    $ChangeRowsAndColumns(m, i)$ 
5:   if  $m_{ii} = 0$  then
6:     Error when executing the method: division by 0
7:     return
8:   end if
9:   for  $j = 1 + 1 \dots n$  do
10:     $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$ 
11:  end for
12: end for
13:  $x \leftarrow backwardSubstitution(m)$ 
```

---

---

**Algorithm 13** simpleLu

---

```
1: READ  $A, b$ 
2:  $[U, L] \leftarrow foundMatrixUandL(A)$ 
3:  $z \leftarrow progressiveSubstitution(L, b)$ 
4:  $x \leftarrow backwardSubstitution(U, z)$ 
5:  $x$  is the vector solution
```

---

---

**Algorithm 14** PivotLu

---

```
1: READ  $A, b$ 
2:  $[U, L, P] \leftarrow foundMatrixUandLandPWithPartialPivoting(A)$ 
3:  $Bn \leftarrow P * b$ 
4:  $z \leftarrow progressiveSubstitution(L, Bn)$ 
5:  $x \leftarrow backwardSubstitution(U, z)$ 
6:  $x$  is the vector solution
```

---

---

**Algorithm 15** Compound Trapeze

---

```
1: READ  $a, b, f, n$ 
2:  $deltaX \leftarrow (b - a)/n$ 
3:  $A \leftarrow 0$ 
4: for  $i = 0 \dots n$  do
5:    $xi \leftarrow a + i * deltaX$ 
6:    $fxi \leftarrow f(xi)$ 
7:   if  $i > 0$  AND  $i < n$  then
8:      $fxi \leftarrow 2 * fxi$ 
9:   end if
10:   $A \leftarrow A + fxi$ 
11: end for
12:  $A \leftarrow A * (deltaX/2)$ 
13:  $A$  is the result of the integral
```

---

---

**Algorithm 16** Simpson 1/3

---

```
1: READ  $a, b, f, n$ 
2:  $\text{delta}X \leftarrow (b - a)/n$ 
3:  $A \leftarrow 0$ 
4: for  $i = 0 \dots n$  do
5:    $xi \leftarrow a + i * \text{delta}X$ 
6:    $fxi \leftarrow f(xi)$ 
7:   if  $i > 0$  AND  $i < n$  then
8:     if  $i \bmod 2 == 0$  then
9:        $fxi \leftarrow 2 * fxi$ 
10:    else
11:       $fxi \leftarrow 4 * fxi$ 
12:    end if
13:  end if
14:   $A \leftarrow A + fxi$ 
15: end for
16:  $A \leftarrow A * (\text{delta}X/3)$ 
17: A is the result of the integral
```

---

---

**Algorithm 17** Simpson 3/8

---

```
1: READ  $a, b, f, n$ 
2:  $\text{delta}X \leftarrow (b - a)/n$ 
3:  $A \leftarrow 0$ 
4: for  $i = 0 \dots n$  do
5:    $xi \leftarrow a + i * \text{delta}X$ 
6:    $fxi \leftarrow f(xi)$ 
7:   if  $i > 0$  AND  $i < n$  then
8:     if  $i \bmod 3 == 0$  then
9:        $fxi \leftarrow 2 * fxi$ 
10:    else
11:       $fxi \leftarrow 3 * fxi$ 
12:    end if
13:  end if
14:   $A \leftarrow A + fxi$ 
15: end for
16:  $A \leftarrow A * (3 * \text{delta}X/8)$ 
17: A is the result of the integral
```

---

---

**Algorithm 18** Steffensen

---

```
1: READ  $f, x_0, tolerance, nMax$ 
2: for  $i = 1 \dots nMax$  do
3:    $x_1 \leftarrow f(x_0)$ 
4:    $x_2 \leftarrow f(x_1)$ 
5:    $denominator \leftarrow (x_2 - x_1) - (x_1 - x_0)$ 
6:   if  $absoluteValue(denominator) < 10e - 16$  then
7:     Error during method execution: division by 0
8:     return
9:   end if
10:   $xi \leftarrow x_2 - ((x_2 - x_1)^2) / denominator$ 
11:  if  $absoluteValue(xi - x_2) < tolerance$  then
12:    xi is an approximation
13:    return
14:  end if
15:   $x_0 \leftarrow xi$ 
16: end for
17: xi is an approximation
```

---

---

**Algorithm 19** Jacobi

---

```
1: READ  $A, b, x, iter, tol$ 
2: if  $foundDeterminant(A)$  then
3:   The determinant is zero, the problem has no unique solution.
4:   return
5: end if
6:  $d \leftarrow findDiagOfMatrix(A)$ 
7:  $p \leftarrow findUpperTriangular(A)$ 
8:  $o \leftarrow findLowerTriangular(A)$ 
9:  $l \leftarrow d - o$ 
10:  $u \leftarrow d - p$ 
11:  $T \leftarrow findInverseOfMatrix(d) * (l + u)$ 
12:  $re \leftarrow foundSpectralRadius(T)$ 
13: if  $re > 1$  then
14:   Spectral radius greater than 1: the method does not converge.
15:   return
16: end if
17:  $C \leftarrow findInverseOfMatrix(d) * b$ 
18:  $i \leftarrow 0$ 
19:  $err \leftarrow tol + 1$ 
20: while  $err > tol$  and  $i < iter$  do
21:    $xi \leftarrow T * x + C$ 
22:    $err \leftarrow findNormOfVector(xi - x)$ 
23:    $x \leftarrow xi$ 
24:    $i \leftarrow i + 1$ 
25: end while
26: if  $i \geq iter$  then
27:   The method fails with the maximum number of iterations given
28:   return
29: end if
30:  $x$  is an approximation with tolerance
```

---

---

**Algorithm 20** Gauss - Seidel

---

```
1: READ  $A, b, x, iter, tol$ 
2: if  $foundDeterminant(A)$  then
3:   The determinant is zero, the problem has no unique solution.
4:   return
5: end if
6:  $d \leftarrow findDiagOfMatrix(A)$ 
7:  $p \leftarrow findUpperTriangular(A)$ 
8:  $o \leftarrow findLowerTriangular(A)$ 
9:  $l \leftarrow d - o$ 
10:  $u \leftarrow d - p$ 
11:  $T \leftarrow findInverseOfMatrix(d - l) * u$ 
12:  $re \leftarrow foundSpectralRadius(T)$ 
13: if  $re > 1$  then
14:   Spectral radius greater than 1: the method does not converge.
15:   return
16: end if
17:  $C \leftarrow findInverseOfMatrix(d - l) * b$ 
18:  $i \leftarrow 0$ 
19:  $err \leftarrow tol + 1$ 
20: while  $err > tol$  and  $i < iter$  do
21:    $xi \leftarrow T * x + C$ 
22:    $err \leftarrow findNormOfVector(xi - x)$ 
23:    $x \leftarrow xi$ 
24:    $i \leftarrow i + 1$ 
25: end while
26: if  $i \geq iter$  then
27:   The method fails with the maximum number of iterations given
28:   return
29: end if
30:  $x$  is an approximation with tolerance
```

---

---

**Algorithm 21** SOR

---

```
1: READ  $A, b, x, iter, tol, w$ 
2: if  $foundDeterminant(A)$  then
3:   The determinant is zero, the problem has no unique solution.
4:   return
5: end if
6:  $d \leftarrow findDiagOfMatrix(A)$ 
7:  $p \leftarrow findUpperTriangular(A)$ 
8:  $o \leftarrow findLowerTriangular(A)$ 
9:  $l \leftarrow d - o$ 
10:  $u \leftarrow d - p$ 
11:  $T \leftarrow findInverseOfMatrix(D - w * l) * [(1 - w)d + w * u]$ 
12:  $re \leftarrow foundSpectralRadius(T)$ 
13: if  $re > 1$  then
14:   Spectral radius greater than 1: the method does not converge.
15:   return
16: end if
17:  $C \leftarrow w * findInverseOfMatrix(D - w * l) * b$ 
18:  $i \leftarrow 0$ 
19:  $err \leftarrow tol + 1$ 
20: while  $err > tol$  do
21:    $xi \leftarrow T * x + C$ 
22:    $err \leftarrow findNormOfVector(xi - x)$ 
23:    $x \leftarrow xi$ 
24:    $i \leftarrow i + 1$ 
25: end while
26: if  $i \geq iter$  then
27:   The method fails with the maximum number of iterations given
28:   return
29: end if
30:  $x$  is an approximation with tolerance
```

---

---

**Algorithm 22** Euler

---

```
1: READ  $f, xi, yi, xf, h$ 
2:  $n \leftarrow (xf - xi)/h$ 
3: for  $i = 0; i < n; i++$  do
4:    $y1 \leftarrow f(xi, yi)$ 
5:    $hy1 \leftarrow h * y1$ 
6:    $newarray.push([xi, y1])$ 
7:    $yi \leftarrow yi + hy1$ 
8:    $xi \leftarrow xi + h$ 
9: end for
10: return
```

---

---

**Algorithm 23** Lineal Spline

---

```
1: READ  $x, y$ 
2: create a square matrix of 0s  $x.length$ 
3: create a matrix of 0s  $x.length$  by 1
4:  $m \leftarrow 2 * (n - 1)$ 
5:  $z \leftarrow 0$ 
6: for  $i = 1; i < x.length; i++$  do
7:    $zerosA[i][z] \leftarrow x[i]$ 
8:    $zerosA[i][z + 1] \leftarrow 1$ 
9:    $z \leftarrow z + 2$ 
10:   $zerosB[i][z] \leftarrow y[i]$ 
11: end for
12:  $zerosA[0][0] = x[0]$ 
13:  $zerosA[0][1] = 1$ 
14:  $zerosB[0][0] = y[0]$ 
15: for  $i = 1; i < x.length - 1; i++$  do
16:   $zerosA[x.length - 1 + 1][z] \leftarrow x[i]$ 
17:   $zerosA[x.length - 1 + 1][z + 2] \leftarrow -x[i]$ 
18:   $zerosA[x.length - 1 + 1][z + 1] \leftarrow 1$ 
19:   $zerosA[x.length - 1 + 1][z + 3] \leftarrow -1$ 
20:   $z \leftarrow z + 2$ 
21:   $zerosB[x.length - 1 + 1][0] \leftarrow 0$ 
22: end for
23: return
```

---



---

**Algorithm 24** Quadratic Spline

---

```
1: READ  $x, y$ 
2: create a square matrix of 0s  $x.length$ 
3: create a matrix of 0s  $x.length$  by 1
4:  $m \leftarrow 3 * (n - 1)$ 
5:  $z \leftarrow 0$ 
6: for  $i = 1; i < x.length; i++$  do
7:    $zerosA[i][z] \leftarrow x[i]^2$ 
8:    $zerosA[i][z + 1] \leftarrow x[i]$ 
9:    $zerosA[i][z + 2] \leftarrow 1$ 
10:   $z \leftarrow z + 3$ 
11:   $zerosB[i][0] \leftarrow y[i]$ 
12: end for
13:  $zerosA[0][0] = x[0]^2$ 
14:  $zerosA[0][1] = x[0]^1$ 
15:  $zerosA[0][2] = 1$ 
16:  $zerosB[0][0] = y[0]$ 
17:  $z \leftarrow 0$ 
18: for  $i = 1; i < x.length - 1; i++$  do
19:    $zerosA[x.length - 1 + 1][z] \leftarrow x[i]^2$ 
20:    $zerosA[x.length - 1 + 1][z + 1] \leftarrow x[i]$ 
21:    $zerosA[x.length - 1 + 1][z + 2] \leftarrow 1$ 
22:    $zerosA[x.length - 1 + 1][z + 3] \leftarrow -(x[i]^2)$ 
23:    $zerosA[x.length - 1 + 1][z + 4] \leftarrow -x[i]$ 
24:    $zerosA[x.length - 1 + 1][z + 5] \leftarrow -1$ 
25:    $z \leftarrow z + 3$ 
26:    $zerosB[x.length - 1 + 1][0] \leftarrow 0$ 
27: end for
28:  $z \leftarrow 0$ 
29: for  $i = 2; i < x.length - 1; i++$  do
30:    $zerosA[2 * x.length - 4 + 1][z] \leftarrow x[i - 1] * 2$ 
31:    $zerosA[2 * x.length - 4 + 1][z + 1] \leftarrow 1$ 
32:    $zerosA[2 * x.length - 4 + 1][z + 2] \leftarrow 0$ 
33:    $zerosA[2 * x.length - 4 + 1][z + 3] \leftarrow -(x[i - 1] * 2)$ 
34:    $zerosA[2 * x.length - 4 + 1][z + 4] \leftarrow -1$ 
35:    $zerosA[2 * x.length - 4 + 1][z + 5] \leftarrow 0$ 
36:    $z \leftarrow z + 3$ 
37: end for
38:  $zerosA[m - 1][0] = 2$ 
39:  $zerosB[m - 1][0] = 0$ 
40: return
```

---

---

**Algorithm 25** Cubic Spline

---

```
1: READ  $x, y$ 
2: create a square matrix of 0s  $x.length$ 
3: create a matrix of 0s  $x.length$  by 1
4:  $m \leftarrow 4 * (n - 1)$ 
5:  $z \leftarrow 0$ 
6: for  $i = 1; i < x.length; i++$  do
7:    $zerosA[i][z] \leftarrow x[i]^3$ 
8:    $zerosA[i][z + 1] \leftarrow x[i]^2$ 
9:    $zerosA[i][z + 2] \leftarrow x[i]$ 
10:   $zerosA[i][z + 3] \leftarrow 1$ 
11:   $z \leftarrow z + 4$ 
12:   $zerosB[i][0] \leftarrow y[i]$ 
13: end for
14:  $zerosA[0][0] = x[0]^3$ 
15:  $zerosA[0][1] = x[0]^2$ 
16:  $zerosA[0][2] = x[0]$ 
17:  $zerosA[0][3] = 1$ 
18:  $zerosB[0][0] = y[0]$ 
19:  $z \leftarrow 0$ 
20: for  $i = 2; i < x.length; i++$  do
21:   $zerosA[2 * x.length - 2 + i][z] \leftarrow (x[i - 1]^3)$ 
22:   $zerosA[2 * x.length - 2 + i][z + 1] \leftarrow x[i - 1]^2$ 
23:   $zerosA[2 * x.length - 2 + i][z + 2] \leftarrow x[i - 1]$ 
24:   $zerosA[2 * x.length - 2 + i][z + 3] \leftarrow 1$ 
25:   $zerosA[2 * x.length - 2 + i][z + 4] \leftarrow -(x[i - 1]^3)$ 
26:   $zerosA[2 * x.length - 2 + i][z + 5] \leftarrow -(x[i - 1]^2)$ 
27:   $zerosA[2 * x.length - 2 + i][z + 6] \leftarrow -x[i - 1]$ 
28:   $zerosA[2 * x.length - 2 + i][z + 7] \leftarrow -1$ 
29:   $z \leftarrow z + 4$ 
30:   $zerosB[x.length - 1 + i][0] \leftarrow 0$ 
31: end for
32:  $z \leftarrow 0$ 
33: for  $i = 2; i < x.length; i++$  do
34:   $zerosA[2 * x.length - 4 + i][z] \leftarrow (x[i - 1]^2) * 3$ 
35:   $zerosA[2 * x.length - 4 + i][z + 1] \leftarrow x[i - 1] * 2$ 
36:   $zerosA[2 * x.length - 4 + i][z + 2] \leftarrow 1$ 
37:   $zerosA[2 * x.length - 4 + i][z + 3] \leftarrow 0$ 
38:   $zerosA[2 * x.length - 4 + i][z + 4] \leftarrow -((x[i - 1]^2) * 3)$ 
39:   $zerosA[2 * x.length - 4 + i][z + 5] \leftarrow -(x[i - 1] * 2)$ 
40:   $zerosA[2 * x.length - 4 + i][z + 6] \leftarrow -1$ 
41:   $zerosA[2 * x.length - 4 + i][z + 7] \leftarrow 0$ 
42:   $z \leftarrow z + 4$ 
43:   $zerosB[2 * x.length - 3 + i][0] \leftarrow 0$ 
44: end for
```

---

---

```

45:  $z \leftarrow 0$ 
46: for  $i = 2; i < x.length; i++$  do
47:    $zerosA[3 * x.length - 6 + i][z] \leftarrow x[i - 1] * 6$ 
48:    $zerosA[3 * x.length - 6 + i][z + 1] \leftarrow 2$ 
49:    $zerosA[3 * x.length - 6 + i][z + 2] \leftarrow 0$ 
50:    $zerosA[3 * x.length - 6 + i][z + 3] \leftarrow 0$ 
51:    $zerosA[3 * x.length - 6 + i][z + 4] \leftarrow -(x[i - 1] * 6)$ 
52:    $zerosA[3 * x.length - 6 + i][z + 5] \leftarrow -2$ 
53:    $zerosA[3 * x.length - 6 + i][z + 6] \leftarrow 0$ 
54:    $zerosA[3 * x.length - 6 + i][z + 7] \leftarrow 0$ 
55:    $z \leftarrow z + 4$ 
56:    $zerosB[2 * x.length - 2 + i][0] \leftarrow 0$ 
57: end for
58:  $zerosA[m - 2][0] = x[0] * 6$ 
59:  $zerosA[m - 2][0] = 2$ 
60:  $zerosA[m - 1][0] = x[x.length - 1] * 6$ 
61:  $zerosA[m - 1][0] = 2$ 
62:  $zerosB[m - 1][0] = 0$ 
63:  $zerosB[m - 2][0] = 0$ 
64: return

```

---



---

**Algorithm 26** Lagrange

---

```

1: READ  $x, y$ 
2:  $yp \leftarrow 0$ 
3:  $p \leftarrow 0$ 
4: for  $i = 1; i < x.length; i++$  do
5:    $p \leftarrow 1$ 
6:   for  $j = 1; j < x.length; j++$  do
7:     if  $i \neq j$  then
8:        $p \leftarrow p * (xp - x[j]) / (x[i] - x[j])$ 
9:     end if
10:  end for
11:   $yp \leftarrow yp + p * y[i]$ 
12: end for
13: return

```

---

---

**Algorithm 27** Divided Differences

---

```
1: READ  $x, y$ 
2: create a square matrix of 0s  $x.length$ 
3:  $w \leftarrow 0$ 
4: for  $i = 1; i < x.length; i++$  do
5:    $z \leftarrow i$ 
6:   while  $y.length > z$  do
7:      $aux1 \leftarrow (zerosarray[z][w] - zerosarray[z-1][w]) / (x[z] - x[z-i])$ 
8:      $zerosarray[z][i] = aux1$ 
9:      $z \leftarrow z + 1$ 
10:  end while
11:   $w \leftarrow z + 1$ 
12: end for
13: return
```

---

---

**Algorithm 28** Fixed Point

---

```
1: READ  $f, g, x0, tol, iter$ 
2: for  $i = 0; i < tol; i++$  do
3:    $x1 \leftarrow x0 \text{evaluated on } g$ 
4:   if  $x1 == x0$  evaluated on  $f$  OR  $abs(x1 - x0) < tol$  then
5:     Break
6:   end if
7:    $x0 \leftarrow x0 - x1$ 
8: end for
9: return
```

---

---

**Algorithm 29** Secant

---

```
1: READ  $f, x_0, x_1, tol, iter$ 
2:  $y_0 \leftarrow x_0$  evaluated on  $f$ 
3: if  $y_0 == 0$  then
4:   return
5: else
6:    $y_1 \leftarrow x_1$  evaluated on  $f$ 
7:    $counter \leftarrow 0$ 
8:    $error \leftarrow tol + 1$ 
9:    $density \leftarrow y_1 - y_0$ 
10:  while  $error > tol$   $y_1 \neq 0$   $counter < iter$  do
11:     $x_2 \leftarrow x_1 - ((y_1 * (x_1 - x_0))/density)$ 
12:     $error \leftarrow abs((x_2 - x_1)/x_2)$ 
13:     $x_0 \leftarrow x_1$ 
14:     $y_0 \leftarrow y_1$ 
15:     $x_1 \leftarrow x_2$ 
16:     $y_1 \leftarrow x_1$  evaluated on  $f$ 
17:     $density \leftarrow y_1 - y_0$ 
18:     $counter \leftarrow counter + 1$ 
19:  end while
20:  if  $error < tol$  then
21:    return root
22:  end if
```

---

---

**Algorithm 30** Trisection

---

```
1: READ  $f, left, right, tolerance, niter$ 
2:  $fRight \leftarrow f(right)$ 
3:  $fLeft \leftarrow f(left)$ 
4:  $counter \leftarrow 0$ 
5: if  $fRight == 0$  then
6:   right is a root
7: else if  $fLeft == 0$  then
8:   left is a root
9: else if  $fLeft * fRight < 0$  then
10:   $xmid1 \leftarrow left + (right - left)/3$ 
11:   $xmid2 \leftarrow right - (right - left)/3$ 
12:   $fXmid1 \leftarrow f(xmid1)$ 
13:   $fXmid2 \leftarrow f(xmid2)$ 
14:   $counter \leftarrow 1$ 
15:   $error1 \leftarrow tolerance + 1$ 
16:   $error2 \leftarrow tolerance + 1$ 
17:  while  $error1 > tolerance$  AND  $error2 > tolerance$  AND
     $fXmid1! = 0$  AND  $fXmid2! = 0$  AND  $counter < niter$  do
18:    if  $fLeft * fXmid1 < 0$  then
19:       $right \leftarrow xmid1$ 
20:       $fRight \leftarrow fXmid1$ 
21:    else if  $fXmid1 * fXmid2 < 0$  then
22:       $left \leftarrow xmid1$ 
23:       $fLeft \leftarrow fXmid1$ 
24:       $right \leftarrow xmid2$ 
25:       $fRight \leftarrow fXmid2$ 
26:    else
27:       $left \leftarrow xmid2$ 
28:       $fLeft \leftarrow fXmid2$ 
29:    end if
30:     $xAux1 \leftarrow xmid1$ 
31:     $xAux2 \leftarrow xmid2$ 
32:     $xmid1 \leftarrow left + (right - left)/3$ 
33:     $fXmid1 \leftarrow f(xmid1)$ 
34:     $xmid2 \leftarrow right - (right - left)/3$ 
35:     $fXmid2 \leftarrow f(xmid2)$ 
36:     $error1 \leftarrow absoluteValue(xmid1 - xAux1)$ 
37:     $error2 \leftarrow absoluteValue(xmid2 - xAux2)$ 
38:     $counter \leftarrow counter + 1$ 
39:  end while
```

---

---

```

40: if  $fX_{mid1} == 0$  then
41:   xmid1 is a root
42: else if  $fX_{mid2} == 0$  then
43:   xmid2 is a root
44: else if  $error1 < tolerance$  then
45:   xmid1 is an approximation with tolerance tolerance
46: else if  $error2 < tolerance$  then
47:   xmid2 is an approximation with tolerance tolerance
48: else
49:   The method fails in niter iterations
50: end if
51:
52: return =0

```

---

---

**Algorithm 31** False position

---

```
1: READ  $f, xi, xs, tolerance, nMax$ 
2:  $f_{xi} \leftarrow f(xi)$ 
3:  $f_{xs} \leftarrow f(xs)$ 
4: if  $f_{xi} == 0$  then
5:    $xi$  is a root
6: else if  $f_{xs} == 0$  then
7:    $xs$  is a root
8: else if  $f_{xi} < f_{xs}$  then
9:    $xm \leftarrow (xi) - ((f_{xi} * (xi - xs)) / (f_{xi} - f_{xs}))$ 
10:   $f_{xm} \leftarrow f(xm)$ 
11:   $error \leftarrow tolerance + 1$ 
12:   $counter \leftarrow 1$ 
13:  while  $f_{xm} \neq 0$   $error > tolerance$   $counter \leq nMax$  do
14:    if  $f_{xi} * f_{xm} < 0$  then
15:       $xs \leftarrow xm$ 
16:       $f_{xs} \leftarrow f_{xm}$ 
17:    else
18:       $xi \leftarrow xm$ 
19:       $f_{xi} \leftarrow f_{xm}$ 
20:    end if
21:     $x_{Aux} \leftarrow xm$ 
22:     $xm \leftarrow (xi) - ((f_{xi} * (xi - xs)) / (f_{xi} - f_{xs}))$ 
23:     $f_{xm} \leftarrow f(xm)$ 
24:     $error \leftarrow (absoluteValue(xm - x_{Aux}) / xm)$ 
25:     $counter \leftarrow counter + 1$ 
26:  end while
27:  if  $f_{xm} == 0$  then
28:     $xm$  is a root
29:  else if  $error \leq tolerance$  then
30:     $xm$  is an approximation to a root with a tolerance  $tolerance$ 
31:  else
32:    Failure in  $nMax$  iterations
33:  end if
34: end if
```

---



---

**Algorithm 32** Heun

---

```
1: READ  $f, x, y, h, n$ 
2:  $counter \leftarrow 0$ 
3: while  $counter \leq n$  do
4:    $k1 \leftarrow f(x, y)$ 
5:    $k2 \leftarrow f(x + h, y + (k1 * h))$ 
6:    $y \leftarrow y + ((k1 * k2) * (h/2))$ 
7:    $x \leftarrow x + h$ 
8:    $counter \leftarrow counter + 1$ 
9: end while
```

---

---

**Algorithm 33** Cholesky

---

```
1: READ  $A, B$ 
2:  $n \leftarrow \text{lenth of } A$ 
3:  $lower \leftarrow$  matrix nxn initialized in ceros
4: for  $j = 0 \dots n$  do
5:   for  $i = 0 \dots j - 1$  do
6:      $lower[i][j] \leftarrow (A[i][j] - \sum_{k=0}^{i-1} lower[k][i] * lower[k][j]) / lower[i][i]$ 
7:   end for
8:    $lower[i][j] \leftarrow \sqrt{A[j][j] - \sum_{k=0}^{i-1} lower[k][j]^2}$ 
9: end for
10:  $uppper \leftarrow lower^T$ 
11:  $Z \leftarrow \text{regressiveSubstitution}(lower|B)$ 
12:  $X \leftarrow \text{progressiveSubstitution}(upper|Z)$ 
```

---

---

**Algorithm 34** Tridiagonal

---

```
1: READ  $A, B, C, D$ 
2:  $N \leftarrow \text{length of } D$ 
3:  $C[0] \leftarrow C[0]/B[0]$ 
4:  $D[0] \leftarrow D[0]/B[0]$ 
5: for  $i = 1 \dots N$  do
6:    $aux \leftarrow B[i] - (A[i] * C[i - 1])$ 
7:    $C[i] \leftarrow C[i]/aux$ 
8:    $D[i] \leftarrow (D[i] - A[i] * D[i - 1])/aux$ 
9: end for
10:  $x \leftarrow$  vector of length N
11: for  $i = 0 \dots N$  do
12:    $x[i] \leftarrow 0$ 
13: end for
14:  $x[n - 1] \leftarrow D[N - 1]$ 
15: for  $i = 0 \dots N - 1$  do
16:    $x[i][i] \leftarrow C[i] * x[i + 1]$ 
17: end for
18: return  $x$ 
```

---