Pseudocodes LaTeX

Daniel Hernandez, Samuel Villegas, Jose Blanco, Neller Pellegrino

June 2, 2022

Algorithm 1 Aitken

```
READ f,g,x0,tolerance,nMax
x1 ← g(x0)
x2 ← g(x1)
x3 ← g(x2)
xi ← x1 − ((x2 − x1)^2)/(x3 − 2 ∗ x2 + x1)
fxi ← f(xi)
error ← tolerance + 1
counter ← 0
while fxi ≠ 0 AND error > tolerance AND counter < nMax do
    x3Aux ← x3
    x2Aux ← x2
    x3 ← g(x3)
    x2 ← x3Aux
    x1 ← x2Aux
    xiAux ← xi
    xi ← x1 − ((x2 − x1)^2)/(x3 − 2 ∗ x2 + x1)
    fxi ← f(xi)
    error ← |xi − xiAux|
    counter ← counter + 1
end while
if fxi = 0 then
    A root has been found and it is xi
else if error ≤ tolerance then
    one approach has been found and it is xi
else
    The method fails with the maximum number of iterations given
end if
```

Algorithm 2 Bisection

READ $f, left, right, tolerance, niter$
$fRight \leftarrow f(right)$
$fLeft \leftarrow f(left)$
if $fRight = 0$ **then**
    right is a root
else if $fLeft = 0$ **then**
    left is a root
else if $fLeft * fRight < 0$ **then**
    $xmid \leftarrow (left * right)/2$
    $fXmid \leftarrow f(xmid)$
    $counter \leftarrow 1$
    $error \leftarrow tolerance + 1$
    while $error > tolerance$ AND $fXmid \neq 0$ AND $counter < niter$ **do**
        if $fLeft * fXmid < 0$ **then**
            $right \leftarrow xmid$
            $fRight \leftarrow fXmid$
        else
            $left \leftarrow xmid$
            $fLeft \leftarrow fXmid$
        end if
        $xAux \leftarrow xmid$
        $xmid \leftarrow (right + left)/2$
        $fXmid \leftarrow f(xmid)$
        $error \leftarrow |xmid - xAux|$
        $counter \leftarrow counter + 1$
    end while
    if $fXmid = 0$ **then**
        xmid is a root
    else if $error < tolerance$ **then**
        xmid is an approximation with tolerance
    else
        The method fails in niter iterations
    end if
else
    Bad range
end if

Algorithm 3 Crout

```
READ A, B
n ← length of A
lower ← matrix nxn initialized in ceros
upper ← identity matrix nxn
for j = 0...n do
    for i = j...n do
        sum ← 0
        for k = 0...j do
            sum ← sum + lower[i][k] * upper[k][j]
        end for
        lower[i][j] ← A[i][j] − sum
    end for
    for i = j + 1...n do
        sum ← 0
        for k = 0...j do
            sum ← sum + lower[j][k] * upper[k][i]
        end for
        upper[j][i] ← (A[j][j] − sum)/lower[j][j]
    end for
end for
Z ← regressiveSubstitution(lower|B)
X ← progressiveSubstitution(upper|Z)
```

Algorithm 4 Doolittle

```
n ← length of A
lower ← matrix nxn initialized in ceros
upper ← identity matrix nxn
for i = 0...n do
    for k = i...n do
        sum ← 0
        for j = 0...i do
            sum ← lower[i][j] * upper[j][k]
        end for
        upper[i][k] ← A[i][k] − sum
    end for
    for k = i...n do
        if i = k then
            lower[i][i] ← 1
        else
            sum ← 0
            for j = 0...i do
                sum ← sum + lower[k][j] * upper[j][i]
            end for
            lower[k][i] ← (A[k][i] − sum)/upper[i][i]
        end if
    end for
end for
Z ← regressiveSubstitution(lower|B)
X ← progressiveSubstitution(upper|Z)
```

Algorithm 5 Incremental Search

READ $f, x0, delta, niter$
$fx0 \leftarrow f(x0)$
if $fx0 = 0$ **then**
    x0 is a root
else
    $x1 \leftarrow x0 + delta$
    $counter \leftarrow 1$
    $fx1 \leftarrow f(x1)$
    while $fx0 * fx1 > 0$ AND $counter > niter$ **do**
        $x0 \leftarrow x1$
        $fx0 \leftarrow fx1$
        $x1 \leftarrow x0 + delta$
        $fx1 \leftarrow f(x1)$
        $counter \leftarrow counter + 1$
    end while
    if $fx1 = 0$ **then**
        x1 is a root
    else if $fx0 * fx1 < 0$ **then**
        there is a root between x0 and x1
    else
        the method fails in niter iterations
    end if
end if

Algorithm 6 Muller

READ $f, x0, tolerance, nMax$
$fx0 \leftarrow f(x0)$
$fx1 \leftarrow f(x1)$
$x1 \leftarrow (x0 + x1)/2$
$fx2 \leftarrow f(x2)$
$h0 \leftarrow x1 - x0$
$h1 \leftarrow x2 - x1$
$delta0 \leftarrow (fx1 - fx0)/h0$
$delta1 \leftarrow (fx2 - fx1)/h1$
$a \leftarrow (delta1 - delta0)/(h1 - h0)$
$b \leftarrow a * h1 + delta1$
$c \leftarrow fx2$
$xi \leftarrow x2 + (-2 * c)/(b + (b/|b|) * \sqrt{b^2 - 4 * a * c})$
$fxi \leftarrow f(xi)$
$error \leftarrow tolerance + 1$
$counter \leftarrow 0$
while $fx1 \neq 0$ AND $error > tolerance$ AND $counter < nMax$ **do**
    $x2Aux \leftarrow x2$
    $x1Aux \leftarrow x1$
    $x2 \leftarrow xi$
    $x1 \leftarrow x2Aux$
    $x0 \leftarrow x1Aux$
    $fx0 \leftarrow f(x0)$
    $fx1 \leftarrow f(x1)$
    $fx2 \leftarrow f(x2)$
    $h0 \leftarrow x1 - x0$
    $h1 \leftarrow x2 - x1$
    $delta0 \leftarrow (fx1 - fx0)/h0$
    $delta1 \leftarrow (fx2 - fx1)/h1$
    $a \leftarrow (delta1 - delta0)/(h1 - h0)$
    $b \leftarrow a * h1 + delta1$
    $c \leftarrow fx2$
    $xi \leftarrow x2 + (-2 * c)/(b + (b/|b|) * \sqrt{b^2 - 4 * a * c})$
    $fxi \leftarrow f(xi)$
    $error \leftarrow tolerance + 1$
    $counter \leftarrow counter + 1$
end while
if $fxi = 0$ **then**
    A root has been found and it is xi
else if $error \leq tolerance$ **then**
    One approach has been found and it is xi
else
    The method fails with the maximum number of iterations given
end if

Algorithm 7 Multiple root

READ $f, f1, f2, x0, tolerance, nMax$
$xi \leftarrow x0$
$fxi \leftarrow f(xi)$
if $fx = 0$ **then**
    A root has been found and it is xi
else
    $counter \leftarrow 0$
    $f1xi \leftarrow f1(xi)$
    $f2xi \leftarrow f2(xi)$
    $error \leftarrow tolerance + 1$
    $det \leftarrow (f1xi^2) - (fxi * f2xi)$
    while $fxi \neq 0$ AND $error > tolerance$ AND $counter < nMAx$ **do**
        $xiAux \leftarrow xi$
        $xi \leftarrow xi - ((fxi * f1xi)/((f1xi^2) - (fxi * f2xi)))$
        $fxi \leftarrow f(xi)$
        $f1xi \leftarrow f1(xi)$
        $f2xi \leftarrow f2(xi)$
        $error \leftarrow |xi - xiAux|$
        $det \leftarrow (f1xi^2) - (fxi * f2xi)$
        $counter \leftarrow counter + 1$
    end while
    if $fxi = 0$ **then**
        A root has been found and it is xi
    else if $error \leq tolerance$ **then**
        One approach has been found and it is xi
    else
        The method fails with the maximum number of iterations given
    end if
end if

Algorithm 8 Newton

READ $f, fder, tolerance, x0, niter$
$fx \leftarrow f(x0)$
$dfx \leftarrow fder(x0)$
$counter \leftarrow 1$
$error \leftarrow tolerance + 1$
while $error > tolerance$ AND $fx \neq 0$ AND $counter < niter$ **do**
    $x1 \leftarrow x0 - (fx/dfx)$
    $fx \leftarrow f(x1)$
    $dfx \leftarrow fder(x1)$
    $error \leftarrow |x1 - x0|$
    $x0 \leftarrow x1$
    $counter \leftarrow counter + 1$
end while
if $fx = 0$ **then** $x0$ is a root
else if $error < tolerance$ **then**
    $x1$ is a root approximation with tolerance $tolerance$
elseth method fails in $niter$ iterations
end if

Algorithm 9 Vandermonde

READ x, y
$n \leftarrow findVectorLength(x)$
$A \leftarrow generateMatrixofOnes(n)$
for $i = 1...n$ **do**
    for $j = 1...n - 1$ **do**
        $A_{ij} \leftarrow x_i^{n-j}$
    end for
end for
$xSolution \leftarrow solveLinearEquation(A, y)$
xSolution is the coefficient vector

Algorithm 10 gaussianElimination

READ A, b
$m \leftarrow concatenateMatrices(A, b)$
for $i = 1...n-1$ **do**
    if $m_{ii} == 0$ **then**
        Error when executing the method: division by 0
        return
    end if
    for $j = 1 + 1...n$ **do**
        $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$
    end for
end for
$x \leftarrow backwardSubstitution(m)$

Algorithm 11 partialPivoting

READ A, b
$m \leftarrow concatenateMatrices(A, b)$
for $i = 1...n-1$ **do**
    $ChangeRows(m, i)$
    if $m_{ii} == 0$ **then**
        Error when executing the method: division by 0
        return
    end if
    for $j = 1 + 1...n$ **do**
        $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$
    end for
end for
$x \leftarrow backwardSubstitution(m)$

Algorithm 12 totalPivoting

READ A, b
$m \leftarrow concatenateMatrices(A, b)$
for $i = 1...n-1$ **do**
    $ChangeRowsAndColumns(m, i)$
    if $m_{ii} = 0$ **then**
        Error when executing the method: division by 0
        return
    end if
    for $j = 1 + 1...n$ **do**
        $m_j \leftarrow m_j - (m_i * (m_{ji}/m_{ii}))$
    end for
end for
$x \leftarrow backwardSubstitution(m)$

Algorithm 13 simpleLu

READ A, b
$[U, L] \leftarrow foundMatrixUandL(A)$
$z \leftarrow progressiveSubstitution(L, b)$
$x \leftarrow backwardSubstitution(U, z)$
x is the vector solution

Algorithm 14 PivotLu

READ A, b
$[U, L, P] \leftarrow foundMatrixUandLandPWithPartialPivoting(A)$
$Bn \leftarrow P * b$
$z \leftarrow progressiveSubstitution(L, Bn)$
$x \leftarrow backwardSubstitution(U, z)$
x is the vector solution

Algorithm 15 Compound Trapeze

READ a, b, f, n
$deltaX \leftarrow (b - a)/n$
$A \leftarrow 0$
for $i = 0...n$ **do**
    $xi \leftarrow a + i * deltaX$
    $fxi \leftarrow f(xi)$
    if $i > 0$ AND $i < n$ **then**
        $fxi \leftarrow 2 * fxi$
    end if
    $A \leftarrow A + fxi$
end for
$A \leftarrow A * (deltaX/2)$
A is the result of the integral

Algorithm 16 Simpson 1/3

```
READ a, b, f, n
deltaX ← (b − a)/n
A ← 0
for i = 0...n do
    xi ← a + i ∗ deltaX
    fxi ← f(xi)
    if i > 0 AND i < n then
        if i module 2 == 0 then
            fxi ← 2 ∗ fxi
        else
            fxi ← 4 ∗ fxi
        end if
    end if
    A ← A + fxi
end for
A ← A ∗ (deltaX/3)
A is the result of the integral
```

Algorithm 17 Simpson 3/8

```
READ a, b, f, n
deltaX ← (b − a)/n
A ← 0
for i = 0...n do
    xi ← a + i ∗ deltaX
    fxi ← f(xi)
    if i > 0 AND i < n then
        if i module 3 == 0 then
            fxi ← 2 ∗ fxi
        else
            fxi ← 3 ∗ fxi
        end if
    end if
    A ← A + fxi
end for
A ← A ∗ (3 ∗ deltaX/8)
A is the result of the integral
```

Algorithm 18 Steffensen

READ $f, x0, tolerance, nMax$
for $i = 1...nMax$ **do**
    $x1 \leftarrow f(x0)$
    $x2 \leftarrow f(x1)$
    $denominator \leftarrow (x2 - x1) - (x1 - x0)$
    if $absoluteValue(denominator) < 10e - 16$ **then**
        Error during method execution: division by 0
        return
    end if
    $xi \leftarrow x2 - ((x2 - x1)^2)/denominator$
    if $absoluteValue(xi - x2) < tolerance$ **then**
        xi is an approximation
        return
    end if
    $x0 \leftarrow xi$
end for
xi is an approximation

Algorithm 19 Jacobi

```
READ A, b, x, iter, tol
if foundDeterminant(A) then
    The determinant is zero, the problem has no unique solution.
    return
end if
d ← findDiagOfMatrix(A)
p ← findUpperTriangular(A)
o ← findLowerTriangular(A)
l ← d − o
u ← d − p
T ← findInverseOfMatrix(d) ∗ (l + u)
re ← foundSpectralRadius(T)
if re > 1 then
    Spectral radius greater than 1: the method does not converge.
    return
end if
C ← findInverseOfMatrix(d) ∗ b
i ← 0
err ← tol + 1
while err > toli < iter do
    xi ← T ∗ x + C
    err ← findNormOfVector(xi − x)
    x ← xi
    i ← i + 1
end while
if i >= iter then
    The method fails with the maximum number of iterations given
    return
end if
x is an approximation with tolerance
```

Algorithm 20 Gauss - Seidel

```
READ A, b, x, iter, tol
if foundDeterminant(A) then
    The determinant is zero, the problem has no unique solution.
    return
end if
d ← findDiagOfMatrix(A)
p ← findUpperTriangular(A)
o ← findLowerTriangular(A)
l ← d − o
u ← d − p
T ← findInverseOfMatrix(d − l) ∗ u
re ← foundSpectralRadius(T)
if re > 1 then
    Spectral radius greater than 1: the method does not converge.
    return
end if
C ← findInverseOfMatrix(d − l) ∗ b
i ← 0
err ← tol + 1
while err > toli < iter do
    xi ← T ∗ x + C
    err ← findNormOfVector(xi − x)
    x ← xi
    i ← i + 1
end while
if i >= iter then
    The method fails with the maximum number of iterations given
    return
end if
x is an approximation with tolerance
```

Algorithm 21 SOR

READ $A, b, x, iter, tol, w$
if $foundDeterminant(A)$ **then**
    The determinant is zero, the problem has no unique solution.
    return
end if
$d \leftarrow findDiagOfMatrix(A)$
$p \leftarrow findUpperTriangular(A)$
$o \leftarrow findLowerTriangular(A)$
$l \leftarrow d - o$
$u \leftarrow d - p$
$T \leftarrow findInverseOfMatriz(D - w * l) * [(1 - w)d + w * u]$
$re \leftarrow foundSpectralRadius(T)$
if $re > 1$ **then**
    Spectral radius greater than 1: the method does not converge.
    return
end if
$C \leftarrow w * findInverseOfMatrix(D - w * l) * b$
$i \leftarrow 0$
$err \leftarrow tol + 1$
while $err > toli < iter$ **do**
    $xi \leftarrow T * x + C$
    $err \leftarrow findNormOfVector(xi - x)$
    $x \leftarrow xi$
    $i \leftarrow i + 1$
end while
if $i >= iter$ **then**
    The method fails with the maximum number of iterations given
    return
end if
x is an approximation with tolerance

Algorithm 22 Euler

READ f, xi, yi, xf, h
$n \leftarrow (xf - xi)/h$
for $i = 0; i < n; i + +$ **do**
    $y1 \leftarrow f(xi, yi)$
    $hy1 \leftarrow h * y1$
    $newarray.push([xi, y1])$
    $yi \leftarrow yi + hy1$
    $xi \leftarrow xi + h$
end for
return

Algorithm 23 Lineal Spline

READ x, y
create a square matrix of 0s x.length
create a matrix of 0s x.length by 1
$m \leftarrow 2 * (n - 1)$
$z \leftarrow 0$
for $i = 1; i < x.length; i++$ **do**
    $zerosA[i][z] \leftarrow x[i]$
    $zerosA[i][z + 1] \leftarrow 1$
    $z \leftarrow z + 2$
    $zerosB[i][z] \leftarrow y[i]$
end for
$zerosA[0][0] = x[0]$
$zerosA[0][1] = 1$
$zerosB[0][0] = y[0]$
for $i = 1; i < x.length - 1; i++$ **do**
    $zerosA[x.length - 1 + 1][z] \leftarrow x[i]$
    $zerosA[x.length - 1 + 1][z + 2] \leftarrow -x[i]$
    $zerosA[x.length - 1 + 1][z + 1] \leftarrow 1$
    $zerosA[x.length - 1 + 1][z + 3] \leftarrow -1$
    $z \leftarrow z + 2$
    $zerosB[x.length - 1 + 1][0] \leftarrow 0$
end for
return

Algorithm 24 Cuadratic Spline

```
READ x, y
create a square matrix of 0s x.length
create a matrix of 0s x.length by 1
m ← 3 ∗ (n − 1)
z ← 0
for i = 1; i < x.length; i + + do
    zerosA[i][z] ← x[i]^2
    zerosA[i][z + 1] ← x[i]
    zerosA[i][z + 2] ← 1
    z ← z + 3
    zerosB[i][0] ← y[i]
end for
zerosA[0][0] = x[0]^2
zerosA[0][1] = x[0]^1
zerosA[0][2] = 1
zerosB[0][0] = y[0]
z ← 0
for i = 1; i < x.length − 1; i + + do
    zerosA[x.length − 1 + 1][z] ← x[i]^2
    zerosA[x.length − 1 + 1][z + 1] ← x[i]
    zerosA[x.length − 1 + 1][z + 2] ← 1
    zerosA[x.length − 1 + 1][z + 3] ← −(x[i]^2)
    zerosA[x.length − 1 + 1][z + 4] ← −x[i]
    zerosA[x.length − 1 + 1][z + 5] ← −1
    z ← z + 3
    zerosB[x.length − 1 + 1][0] ← 0
end for
z ← 0
for i = 2; i < x.length − 1; i + + do
    zerosA[2 ∗ x.length − 4 + 1][z] ← x[i − 1] ∗ 2
    zerosA[2 ∗ x.length − 4 + 1][z + 1] ← 1
    zerosA[2 ∗ x.length − 4 + 1][z + 2] ← 0
    zerosA[2 ∗ x.length − 4 + 1][z + 3] ← −(x[i − 1] ∗ 2)
    zerosA[2 ∗ x.length − 4 + 1][z + 4] ← −1
    zerosA[2 ∗ x.length − 4 + 1][z + 5] ← 0
    z ← z + 3
end for
zerosA[m − 1][0] = 2
zerosB[m − 1][0] = 0
return
```

Algorithm 25 Cubic Spline

READ x, y
create a square matrix of 0s x.length
create a matrix of 0s x.length by 1
$m \leftarrow 4 * (n - 1)$
$z \leftarrow 0$
for $i = 1; i < x.length; i++$ **do**
  $zerosA[i][z] \leftarrow x[i]^3$
  $zerosA[i][z+1] \leftarrow x[i]^2$
  $zerosA[i][z+2] \leftarrow x[i]$
  $zerosA[i][z+3] \leftarrow 1$
  $z \leftarrow z + 4$
  $zerosB[i][0] \leftarrow y[i]$
end for
$zerosA[0][0] = x[0]^3$
$zerosA[0][1] = x[0]^2$
$zerosA[0][2] = x[0]$
$zerosA[0][3] = 1$
$zerosB[0][0] = y[0]$
$z \leftarrow 0$
for $i = 2; i < x.length; i++$ **do**
  $zerosA[2 * x.length - 2 + i][z] \leftarrow (x[i-1]^3)$
  $zerosA[2 * x.length - 2 + i][z+1] \leftarrow x[i-1]^2$
  $zerosA[2 * x.length - 2 + i][z+2] \leftarrow x[i-1]$
  $zerosA[2 * x.length - 2 + i][z+3] \leftarrow 1$
  $zerosA[2 * x.length - 2 + i][z+4] \leftarrow -(x[i-1]^3)$
  $zerosA[2 * x.length - 2 + i][z+5] \leftarrow -(x[i-1]^2)$
  $zerosA[2 * x.length - 2 + i][z+6] \leftarrow -x[i-1]$
  $zerosA[2 * x.length - 2 + i][z+7] \leftarrow -1$
  $z \leftarrow z + 4$
  $zerosB[x.length - 1 + i][0] \leftarrow 0$
end for
$z \leftarrow 0$
for $i = 2; i < x.length; i++$ **do**
  $zerosA[2 * x.length - 4 + i][z] \leftarrow (x[i-1]^2) * 3$
  $zerosA[2 * x.length - 4 + i][z+1] \leftarrow x[i-1] * 2$
  $zerosA[2 * x.length - 4 + i][z+2] \leftarrow 1$
  $zerosA[2 * x.length - 4 + i][z+3] \leftarrow 0$
  $zerosA[2 * x.length - 4 + i][z+4] \leftarrow -((x[i-1]^2) * 3)$
  $zerosA[2 * x.length - 4 + i][z+5] \leftarrow -(x[i-1] * 2)$
  $zerosA[2 * x.length - 4 + i][z+6] \leftarrow -1$
  $zerosA[2 * x.length - 4 + i][z+7] \leftarrow 0$
  $z \leftarrow z + 4$
  $zerosB[2 * x.length - 3 + i][0] \leftarrow 0$
end for

```
z ← 0
for i = 2; i < x.length; i++ do
    zerosA[3 * x.length − 6 + i][z] ← x[i − 1] * 6
    zerosA[3 * x.length − 6 + i][z + 1] ← 2
    zerosA[3 * x.length − 6 + i][z + 2] ← 0
    zerosA[3 * x.length − 6 + i][z + 3] ← 0
    zerosA[3 * x.length − 6 + i][z + 4] ← −(x[i − 1] * 6)
    zerosA[3 * x.length − 6 + i][z + 5] ← −2
    zerosA[3 * x.length − 6 + i][z + 6] ← 0
    zerosA[3 * x.length − 6 + i][z + 7] ← 0
    z ← z + 4
    zerosB[2 * x.length − 2 + i][0] ← 0
end for
zerosA[m − 2][0] = x[0] * 6
zerosA[m − 2][0] = 2
zerosA[m − 1][0] = x[x.length − 1] * 6
zerosA[m − 1][0] = 2
zerosB[m − 1][0] = 0
zerosB[m − 2][0] = 0
return
```

Algorithm 26 Lagrange

```
READ x, y
yp ← 0
p ← 0
for i = 1; i < x.length; i++ do
    p ← 1
    for j = 1; i < x.length; j++ do
        if i! = j then
            p ← p * (xp − x[j])/(x[i] − x[j])
        end if
    end for
    yp ← yp + p * y[i]
end for
return
```

Algorithm 27 Divided Diferences

READ x, y
create a square matrix of 0s x.length
$w \leftarrow 0$
for $i = 1; i < x.length; i++$ **do**
    $z \leftarrow i$
    while $y.length > z$ **do**
        $aux1 \leftarrow (zerosarray[z][w] - zerosarray[z-1][w])/(x[z] - x[z-i])$
        $zerosarray[z][i] = aux1$
        $z \leftarrow z + 1$
    end while
    $w \leftarrow z + 1$
end for
return

Algorithm 28 Fixed Point

READ $f, g, x0, tol, iter$
for $i = 0; i < tol; i++$ **do**
    $x1 \leftarrow x0evaluatedong$
    if x1 == x0 evaluated on f OR abs(x1-x0)<tol **then**
        Break
    end if
    $x0 \leftarrow x0 - x1$
end for
return

Algorithm 29 Secant

```
READ f, x0, x1, tol, iter
y0 ← x0 evaluated on f
if y0 == 0 then
    return
else
    y1 ← x1 evaluated on f
    counter ← 0
    error ← tol + 1
    density ← y1 − y0
    while error > tol  y1! = 0  counter < iter do
        x2 ← x1 − ((y1 ∗ (x1 − x0))/density)
        error ← abs((x2 − x1)/x2)
        x0 ← x1
        y0 ← y1
        x1 ← x2
        y1 ← x1 evaluated on f
        density ← y1 − y0
        counter ← counter + 1
    end while
    if error < tol then
        return root
    end if
```

Algorithm 30 Trisection

READ $f, left, right, tolerance, niter$
$fRight \leftarrow f(right)$
$fLeft \leftarrow f(left)$
$counter \leftarrow 0$
if $fRight == 0$ **then**
    right is a root
else if $fLeft == 0$ **then**
    left is a root
else if $fLeft * fRigth < 0$ **then**
    $xmid1 \leftarrow left + (right - left)/3$
    $xmid2 \leftarrow right - (right - left)/3$
    $fXmid1 \leftarrow f(xmid1)$
    $fXmid2 \leftarrow f(xmid2)$
    $counter \leftarrow 1$
    $error1 \leftarrow tolerance + 1$
    $error2 \leftarrow tolerance + 1$
    while $error1 > tolerance$ AND $error2 > tolerance$ AND $fXmid1 != 0$ AND $fXmid2 != 0$ AND $counter < niter$ **do**
        if $fLeft * fXmid1 < 0$ **then**
            $right \leftarrow xmid1$
            $fRight \leftarrow fXmid1$
        else if $fXmid1 * fXmid2 < 0$ **then**
            $left \leftarrow xmid1$
            $fLeft \leftarrow fXmid1$
            $right \leftarrow xmid2$
            $fRight \leftarrow fXmid2$
        else
            $left \leftarrow xmid2$
            $fLeft \leftarrow fXmid2$
        end if
        $xAux1 \leftarrow xmid1$
        $xAux2 \leftarrow xmid2$
        $xmid1 \leftarrow left + (right - left)/3$
        $fXmid1 \leftarrow f(xmid1)$
        $xmid2 \leftarrow right - (right - left)/3$
        $fXmid2 \leftarrow f(xmid2)$
        $error1 \leftarrow absoluteValue(xmid1 - xAux1)$
        $error2 \leftarrow absoluteValue(xmid2 - xAux2)$
        $counter \leftarrow counter + 1$
    end while

```
if fXmid1 == 0 then
    xmid1 is a root
else if fXmid2 == 0 then
    xmid2 is a root
else if error1 < tolerance then
    xmid1 is an approximation with tolerance tolerance
else if error2 < tolerance then
    xmid2 is an approximation with tolerance tolerance
else
    The method fails in niter iterations
end if

return =0
```

Algorithm 31 False position

```
READ f, xi, xs, tolerance, nMax
fxi ← f(xi)
fxs ← f(xs)
if fxi == 0 then
    xi is a root
else if fxs == 0 then
    xs is a root
else if fxi < fxs then
    xm ← (xi) − ((fxi ∗ (xi − xs))/(fxi − fxs))
    fxm ← f(xm)
    error ← tolerance + 1
    counter ← 1
    while fxm! == 0error > tolerancecounter <= nMax do
        if fxi ∗ fxm < 0 then
            xs ← xm
            fxs ← fxm
        else
            xi ← xm
            fxi ← fxm
        end if
        xAux ← xm
        xm ← (xi) − ((fxi ∗ (xi − xs))/(fxi − fxs))
        fxm ← f(xm)
        error ← (absoluteValue(xm − xAux)/xm)
        counter ← counter + 1
    end while
    if fxm == 0 then
        xm is a root
    else if error <= tolerance then
        xm is an approximation to a root with a tolerance tolerance
    else
        Failure in nMax iterations
    end if
end if
```

Algorithm 32 Heun

READ f, x, y, h, n
$counter \leftarrow 0$
while $counter \leq n$ **do**
    $k1 \leftarrow f(x, y)$
    $k2 \leftarrow f(x + h, y + (k1 * h))$
    $y \leftarrow y + ((k1 * k2) * (h/2))$
    $x \leftarrow x + h$
    $counter \leftarrow counter + 1$
end while

Algorithm 33 Cholesky

READ A, B
$n \leftarrow$ lenth of A
$lower \leftarrow$ matrix nxn initialized in ceros
for $j = 0...n$ **do**
    for $i = 0...j - 1$ **do**
        $lower[i][j] \leftarrow (A[i][j] - \sum_{k=0}^{i-1} lower[k][i] * lower[k][j])/lower[i][i]$
    end for
    $lower[i][j] \leftarrow \sqrt{A[j][j] - \sum_{k=0}^{i-1} lower[k][j]^2}$
end for
$uppper \leftarrow lower^T$
$Z \leftarrow regressiveSubstitution(lower|B)$
$X \leftarrow progressiveSubstitution(upper|Z)$

Algorithm 34 Tridiagonal

READ A, B, C, D
$N \leftarrow$ length of D
$C[0] \leftarrow C[0]/B[0]$
$D[0] \leftarrow D[0]/B[0]$
for $i = 1...N$ **do**
    $aux \leftarrow B[i] - (A[i] * C[i - 1])$
    $C[i] \leftarrow C[i]/aux$
    $D[i] \leftarrow (D[i] - A[i] * D[i - 1])/aux$
end for
$x \leftarrow$ vector of length N
for $i = 0...N$ **do**
    $x[i] \leftarrow 0$
end for
$x[n - 1] \leftarrow D[N - 1]$
for $i = 0...N - 1$ **do**
    $x[i][i] - C[i] * x[i + 1]$
end for
$returnx$