Final Report SolverPro

Universidad EAFIT
Computer and Systems Department

Delivery Date of This Report: 20/11/2024

Project Name: SolverPro

Website URL (GitHub repository)

https://github.com/MauricioCa07/SolverPro.git

Team Members

- Mauricio Carrillo Carvajal
- Sebastian Cano
- Manuel Quintero
- Juan Diego Llorente

User's manual

The set of instructions and important information for the user to know and understand how to navigate and use the final product is available in the proper project, section "Tutorials" with the purpose of unification.

Installation/setup guide

Attached on repo's Readme.

Pseudocodes of methods

1. Bisection

Algorithm 1 Bisection Method with Error Handling and Iteration Limit

Input: f, a, b, tol, N_max
Initialize: $fa \leftarrow f(a)$, $pm \leftarrow \frac{a+b}{2}$, $fpm \leftarrow f(pm)$
$E \leftarrow 1000$, $cont \leftarrow 1$
while $E >$ tol **and** $cont <$ N_max **do**
  if $fa \times fpm < 0$ **then**
    $b \leftarrow pm$
  else
    $a \leftarrow pm$
    $fa \leftarrow fpm$ {Update fa when new asigned a}
  end if
  $p_0 \leftarrow pm$
  $pm \leftarrow \frac{a+b}{2}$
  $fpm \leftarrow f(pm)$
  $E \leftarrow |pm - p_0|$
  $cont \leftarrow cont + 1$
end while
Print: "Root found at", pm, "in", $cont$, "iterations with an error of:", E
Return: pm =0

2. False Position

Algorithm 2 False Position Method with Error Handling and Iteration Limit

Input: f, a, b, tol, N_max
Initialize: $fa \leftarrow f(a)$, $fb \leftarrow f(b)$
$pm \leftarrow \frac{fb \cdot a - fa \cdot b}{fb - fa}$
$fpm \leftarrow f(pm)$, $E \leftarrow 1000$, $cont \leftarrow 1$
while $E >$ tol **and** $cont <$ N_max **do**
  if $fa \times fpm < 0$ **then**
    $b \leftarrow pm$
    $fb \leftarrow fpm$
  else
    $a \leftarrow pm$
    $fa \leftarrow fpm$
  end if
  $p_0 \leftarrow pm$
  $pm \leftarrow \frac{fb \cdot a - fa \cdot b}{fb - fa}$
  $fpm \leftarrow f(pm)$
  $E \leftarrow |pm - p_0|$
  $cont \leftarrow cont + 1$
end while
Print: "Root found at", pm, "in", $cont$, "iterations with an error of", E
Return: pm =0

4. Gaussian Elimination

Algorithm 3 Gaussian Elimination with Back Substitution

Input: $matrixA$, $matrixB$, n
Initialize: $M \leftarrow$ new Array(n)
for $i = 0$ **to** $n-1$ **do**
  $M[i] \leftarrow$ new Array$(n+1)$
  for $j = 0$ **to** $n-1$ **do**
    $M[i][j] \leftarrow A[i][j]$
  end for
  $M[i][n] \leftarrow B[i]$
end for
for $k = 0$ **to** $n-2$ **do**
  for $i = k+1$ **to** $n-1$ **do**
    $ratio \leftarrow \frac{M[i][k]}{M[k][k]}$
    for $j = k$ **to** n **do**
      $M[i][j] \leftarrow M[i][j] - ratio \cdot M[k][j]$
    end for
  end for
end for
$X \leftarrow$ new Array(n)
for $i = n-1$ **to** 0 **do**
  $sum \leftarrow 0$
  for $j = i+1$ **to** $n-1$ **do**
    $sum \leftarrow sum + M[i][j] \cdot X[j]$
  end for
  $X[i] \leftarrow \frac{M[i][n] - sum}{M[i][i]}$
end for
Return: X
=0

4. Partial Pivoting

Algorithm 4 Gaussian Elimination with Partial Pivoting

Input: A, b, n {A is the coefficient matrix, b is the constant vector, n is the number of variables}
Combine A and b into an augmented matrix M
for $k = 1$ **to** $n - 1$ **do**
  Initialize: max $\leftarrow |M[k,k]|$, maxRow $\leftarrow k$
  for $i = k + 1$ **to** n **do**
    if $|M[i,k]| >$ max **then**
      max $\leftarrow |M[i,k]|$
      maxRow $\leftarrow i$
    end if
  end for
  if max ≈ 0 **then**
    Throw error: "Singular matrix"
  end if
  Swap: $M[k,*]$ **with** $M[\text{maxRow},*]$ {Swap rows for partial pivoting}
  for $i = k + 1$ **to** n **do**
    factor $\leftarrow M[i,k]/M[k,k]$
    for $j = k$ **to** $n + 1$ **do**
      $M[i,j] \leftarrow M[i,j] - \text{factor} \cdot M[k,j]$ {Elimination step}
    end for
  end for
end for
Initialize: $x \leftarrow$ new vector of size n
for $i = n$ **to** 1 **(descending) do**
  sum $\leftarrow 0$
  for $j = i + 1$ **to** n **do**
    sum $\leftarrow \text{sum} + M[i,j] \cdot x[j]$
  end for
  $x[i] \leftarrow (M[i,n+1] - \text{sum})/M[i,i]$ {Back-substitution}
end for
Return: x =0

5. Total Pivoting

Algorithm 5 Gaussian Elimination with Total Pivoting

Input: A, b, n $\{A$ es la matriz de coeficientes, b es el vector de constantes, n es el número de variables$\}$
Combine A y b en una matriz aumentada M
for $k = 1$ **to** $n - 1$ **do**
  Initialize: $\max \leftarrow 0$
  for $i = k$ **to** n **do**
    for $j = k$ **to** n **do**
      if $|M[i, j]| > \max$ **then**
        $\max \leftarrow |M[i, j]|$
        $\text{maxRow} \leftarrow i$
        $\text{maxCol} \leftarrow j$
      end if
    end for
  end for
  if $\max \approx 0$ **then**
    Throw error: "Singular matrix"
  end if
  Swap: $M[k, *]$ **with** $M[\text{maxRow}, *]$ $\{$Intercambiar filas$\}$
  Swap: $M[*, k]$ **with** $M[*, \text{maxCol}]$ $\{$Intercambiar columnas$\}$
  Record the column swap for reordering the final solution
  for $i = k + 1$ **to** n **do**
    $\text{factor} \leftarrow M[i, k]/M[k, k]$
    for $j = k$ **to** $n + 1$ **do**
      $M[i, j] \leftarrow M[i, j] - \text{factor} \cdot M[k, j]$ $\{$Paso de eliminación$\}$
    end for
  end for
end for
Initialize: $x \leftarrow$ new vector of size n
for $i = n$ **to** 1 **(descending) do**
  $\text{sum} \leftarrow 0$
  for $j = i + 1$ **to** n **do**
    $\text{sum} \leftarrow \text{sum} + M[i, j] \cdot x[j]$
  end for
  $x[i] \leftarrow (M[i, n + 1] - \text{sum})/M[i, i]$ $\{$Sustitución hacia atrás$\}$
end for
Reorder x according to the recorded column swaps $\{$Reordenar según los intercambios de columnas$\}$
Return: x =0

6. Newton's Method

Algorithm 6 Newton's Method with Error Handling

Entrada: f, f', x_0, tol, max_iter
Inicializar: $x \leftarrow x_0$, iter $\leftarrow 0$, error $\leftarrow 1$
Verificar: Si $f(x_0)$ o $f'(x_0)$ no están definidos en el dominio, lanzar error
Verificar: Si $f'(x_0) = 0$, lanzar error (no se puede dividir por 0)
while error ¿ tol **y** iter ¡ max_iter **do**
  $f(x) \leftarrow$ evaluar f en x
  $f'(x) \leftarrow$ evaluar f' en x
  $x_{\text{next}} \leftarrow x - \frac{f(x)}{f'(x)}$
  $error \leftarrow |x_{\text{next}} - x|$
  if $error < tol$ **then**
    Imprimir: "Solución encontrada en la iteración", iter
    Retornar: x_{next}
  else if $f(x_{\text{next}}) = 0$ **then**
    Imprimir: "Raíz exacta encontrada en la iteración", iter
    Retornar: x_{next}
  else if iter $=$ max_iter **then**
    Imprimir: "No se encontró solución en el número máximo de iteraciones"
    Retornar: None
  end if
  $x \leftarrow x_{\text{next}}$
  iter $\leftarrow$ iter $+ 1$
end while
Imprimir: "It was impossible to find the root within", max_iter, "iterations"
Retornar: None =0

7. Fixed Point

Algorithm 7 Fixed Point Method with Error Handling and Iteration Limit

Entrada: f, g, x_0, tol, max_iter
Inicializar: $x \leftarrow x_0$, iter $\leftarrow 0$, error $\leftarrow 1$
Verificar: Si $f(x_0)$ o $g(x_0)$ no están definidos en el dominio, lanzar error
Verificar: Si $f \equiv g$, lanzar error (no pueden ser iguales)
while error ¿ tol **y** iter ¡ max_iter **do**
  $x_{\text{next}} \leftarrow g(x)$
  $error \leftarrow |x_{\text{next}} - x|$
  if $error < tol$ **then**
    Imprimir: "Solución encontrada en la iteración", iter
    Retornar: x_{next}
  else if $f(x_{\text{next}}) = 0$ **then**
    Imprimir: "Raíz exacta encontrada en la iteración", iter
    Retornar: x_{next}
  else if iter = max_iter **then**
    Imprimir: "No se encontró solución en el número máximo de iteraciones"
    Retornar: None
  end if
  $x \leftarrow x_{\text{next}}$
  iter $\leftarrow$ iter + 1
end while
Imprimir: "It was impossible to find the root within", max_iter, "iterations"
Retornar: None =0

8. Incremental searches

Algorithm 8 Incremental Search Method with Iteration Limit

Input: f, x_0, h, N_max
Initialize: $x_{\text{inf}} \leftarrow x_0$, $x_{\text{sup}} \leftarrow x_{\text{inf}} + h$
$y_{\text{inf}} \leftarrow f(x_{\text{inf}})$, $y_{\text{sup}} \leftarrow f(x_{\text{sup}})$
for $i = 1$ **until** N_max **do**
  if $y_{\text{inf}} \times y_{\text{sup}} \leq 0$ **then**
    Print: "Root at the interval between", x_{inf}, "y", x_{sup}, "in iteration", i
    Return: x_{inf}, x_{sup}
  end if
  $x_{\text{inf}} \leftarrow x_{\text{sup}}$
  $y_{\text{inf}} \leftarrow y_{\text{sup}}$
  $x_{\text{sup}} \leftarrow x_{\text{inf}} + h$
  $y_{\text{sup}} \leftarrow f(x_{\text{sup}})$
end for
if $i > N\_max$ **then**
  Print: "Could not find sign switch on maximum iterations"
  Return: None
end if=0

9. Multiple Roots Method (Euler-Chebyshev Method)

Algorithm 9 Multiple Roots Method

Input: f, f', f'', x_0, ϵ, N_max
Initialize: $n \leftarrow 0$
if $|f(x_0)| < \epsilon$ **then**
  Return: x_0 {x_0 is already a root}
end if
while $n <$ N_max **do**
  $f_x \leftarrow f(x_0)$
  $f'_x \leftarrow f'(x_0)$
  $f''_x \leftarrow f''(x_0)$
  $x_1 \leftarrow x_0 - \frac{f_x \cdot f'_x}{(f'_x)^2 - f_x \cdot f''_x}$ {Update x using the Euler-Chebyshev formula}
  if $|x_1 - x_0| < \epsilon$ **or** $|f(x_1)| < \epsilon$ **then**
    Return: x_1 {Convergence achieved}
  end if
  $x_0 \leftarrow x_1$
  $n \leftarrow n + 1$
end while
Print: "Maximum iterations reached without convergence"
Return: None =0

10. Crout's method

Algorithm 10 Crout's Method for LU Factorization

Input: Matrix A, vector b
Initialize: $n \leftarrow$ rows of A
$L \leftarrow I_n$, $U \leftarrow I_n$ {Identity matrices of size n}
Steps: Empty list to store progress
for $i = 0$ **to** $n - 2$ **do**
  for $j = i$ **to** $n - 1$ **do**
    $L[j][i] \leftarrow A[j][i]$
    for $k = 0$ **to** $i - 1$ **do**
      $L[j][i] \leftarrow L[j][i] - L[j][k] \cdot U[k][i]$
    end for
  end for
  Store Step: "Matrix L updated after column $i + 1$"
  for $j = i + 1$ **to** $n - 1$ **do**
    $U[i][j] \leftarrow A[i][j]$
    for $k = 0$ **to** $i - 1$ **do**
      $U[i][j] \leftarrow U[i][j] - L[i][k] \cdot U[k][j]$
    end for
    $U[i][j] \leftarrow U[i][j]/L[i][i]$
  end for
  Store Step: "Matrix U updated after row $i + 1$"
end for
$L[n-1][n-1] \leftarrow A[n-1][n-1]$
for $k = 0$ **to** $n - 2$ **do**
  $L[n-1][n-1] \leftarrow L[n-1][n-1] - L[n-1][k] \cdot U[k][n-1]$
end for
Store Step: "Final L matrix computed"
Solve: $Ly = b$ using forward substitution
Store Step: "Intermediate solution vector y computed"
Solve: $Ux = y$ using backward substitution
Store Step: "Solution vector x computed"
Return: x, L, U =0

12. Jacobi

Algorithm 11 Jacobi Method

Require: Matrix A of size $n \times n$, vector b of size n, initial guess $x^{(0)}$, tolerance tol, maximum iterations maxIter

Ensure: Approximate solution vector x

Set $k \leftarrow 0$
Set $x^{(k)} \leftarrow x^{(0)}$
repeat
  Set $x^{(k+1)} \leftarrow x^{(k)}$
  for $i = 1$ to n **do**
    sum $\leftarrow 0$
    for $j = 1$ to n **do**
      if $j \neq i$ **then**
        sum $\leftarrow$ sum $+ A_{ij} \cdot x_j^{(k)}$
      end if
    end for
    $x_i^{(k+1)} \leftarrow \dfrac{b_i - \text{sum}}{A_{ii}}$
  end for
  Compute error $\leftarrow \|x^{(k+1)} - x^{(k)}\|$
  $k \leftarrow k + 1$
until error $<$ tol **or** $k \geq$ maxIter
return $x^{(k)}$ =0

13. Newton's Divided Differences

Algorithm 12 Newton's Divided Differences Method

Require: Data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$
Ensure: Newton's interpolating polynomial $P(x)$

Compute divided differences:
for $i = 0$ to n **do**
  $f[x_i] \leftarrow y_i$
end for
for $j = 1$ to n **do**
  for $i = n$ down to j **do**
    $f[x_i, x_{i-1}, \ldots, x_{i-j}] \leftarrow \dfrac{f[x_i, x_{i-1}, \ldots, x_{i-j+1}] - f[x_{i-1}, x_{i-2}, \ldots, x_{i-j}]}{x_i - x_{i-j}}$
  end for
end for
Construct the interpolating polynomial:
$P(x) \leftarrow f[x_0]$
for $k = 1$ to n **do**
  term $\leftarrow f[x_k, x_{k-1}, \ldots, x_0]$
  for $i = 0$ to $k - 1$ **do**
    term $\leftarrow$ term $\times (x - x_i)$
  end for
  $P(x) \leftarrow P(x) +$ term
end for
return $P(x)$ =0

13. Secant

Algorithm 13 Secant Method

Require: Function $f(x)$, initial guesses x_0 and x_1, tolerance tol, maximum iterations maxIter
Ensure: Approximate root x

Initialize $k \leftarrow 0$
while $k < \text{maxIter}$ **do**
  $f(x_0) \leftarrow f(x_0)$
  $f(x_1) \leftarrow f(x_1)$
  if $f(x_1) = 0$ **then**
    return x_1
  end if
  if $f(x_1) - f(x_0) = 0$ **then**
    return **Error**: Division by zero
  end if
  $x_2 \leftarrow x_1 - f(x_1) \times \dfrac{(x_1 - x_0)}{f(x_1) - f(x_0)}$
  error $\leftarrow |x_2 - x_1|$
  if error $<$ tol **then**
    return x_2
  end if
  $x_0 \leftarrow x_1$
  $x_1 \leftarrow x_2$
  $k \leftarrow k + 1$
end while
return **No root found within the maximum number of iterations** =0

14. LU no pivoting

Algorithm 14 LU Decomposition without Pivoting

Require: Matrix A of size $n \times n$, vector b of size n
Ensure: Lower triangular matrix L, upper triangular matrix U, solution vector x

Initialize L as an $n \times n$ zero matrix
Initialize U as an $n \times n$ zero matrix
for $i = 1$ to n **do**
  for $k = i$ to n **do**
    sum $\leftarrow 0$
    for $j = 1$ to $i - 1$ **do**
      sum $\leftarrow$ sum $+ L_{ij} \times U_{jk}$
    end for
    $U_{ik} \leftarrow A_{ik} -$ sum
  end for
  for $k = i$ to n **do**
    if $i = k$ **then**
      $L_{ik} \leftarrow 1$
    else
      sum $\leftarrow 0$
      for $j = 1$ to $i - 1$ **do**
        sum $\leftarrow$ sum $+ L_{kj} \times U_{ji}$
      end for
      if $U_{ii} = 0$ **then**
        return **Error**: Zero pivot encountered
      end if
      $L_{ki} \leftarrow \dfrac{A_{ki} - \text{sum}}{U_{ii}}$
    end if
  end for
end for
Forward Substitution: Solve $Ly = b$
Initialize vector y of size n
for $i = 1$ to n **do**
  sum $\leftarrow 0$
  for $j = 1$ to $i - 1$ **do**
    sum $\leftarrow$ sum $+ L_{ij} \times y_j$
  end for
  $y_i \leftarrow b_i -$ sum
end for
Back Substitution: Solve $Ux = y$
Initialize vector x of size n
for $i = n$ downto 1 **do**
  sum $\leftarrow 0$
  for $j = i + 1$ to n **do**
    sum $\leftarrow$ sum $+ U_{ij} \times x_j$
  end for
  if $U_{ii} = 0$ **then**
    return **Error**: Zero pivot encountered
  end if
  $x_i \leftarrow \dfrac{y_i - \text{sum}}{U_{ii}}$
end for
return L, U, x =0

0.1 15. Cholesky

Algorithm 15 Cholesky Decomposition

Require: Symmetric positive definite matrix A of size $n \times n$, vector b of size n

Ensure: Lower triangular matrix L, solution vector x

```
Initialize L as an n × n zero matrix
for i = 1 to n do
    for j = 1 to i do
        sum ← 0
        for k = 1 to j − 1 do
            sum ← sum + L_ik × L_jk
        end for
        if i = j then
            if A_ii − sum ≤ 0 then
                return Error: Matrix is not positive definite
            end if
            L_ij ← sqrt(A_ii − sum)
        else
            L_ij ← (1 / L_jj) × (A_ij − sum)
        end if
    end for
end for
Forward Substitution: Solve Ly = b
Initialize vector y of size n
for i = 1 to n do
    sum ← 0
    for j = 1 to i − 1 do
        sum ← sum + L_ij × y_j
    end for
    y_i ← (b_i − sum) / L_ii
end for
Back Substitution: Solve L^T x = y
Initialize vector x of size n
for i = n downto 1 do
    sum ← 0
    for j = i + 1 to n do
        sum ← sum + L_ji × x_j
    end for
    x_i ← (y_i − sum) / L_ii
end for
return L, x =0
```

16. LU with partial pivoting

Algorithm 16 LU Decomposition with Partial Pivoting

Input: Matrix A, vector b
Initialize: $n \leftarrow$ rows of A, $L \leftarrow 0_{n\times n}$
$U \leftarrow A$, $P \leftarrow [0, 1, \ldots, n-1]$
Steps: Empty list to store progress
for $k = 0$ **to** $n-1$ **do**
    Find pivot: $pivot \leftarrow k$
    for $i = k+1$ **to** $n-1$ **do**
        if $|U[i][k]| > |U[pivot][k]|$ **then**
            $pivot \leftarrow i$
        end if
    end for
    if $pivot \neq k$ **then**
        Swap rows k and $pivot$ in U
        Swap $P[k]$ and $P[pivot]$
        Store Step: "Pivoting: Rows k and $pivot$ swapped"
    end if
    for $i = k+1$ **to** $n-1$ **do**
        $m \leftarrow U[i][k]/U[k][k]$
        $L[i][k] \leftarrow m$
        for $j = k$ **to** $n-1$ **do**
            $U[i][j] \leftarrow U[i][j] - m \cdot U[k][j]$
        end for
    end for
    Store Step: "Matrix U updated, and L column k computed"
end for
Solve: $Ly = Pb$ using forward substitution
Store Step: "Solution vector y computed"
Solve: $Ux = y$ using backward substitution
Store Step: "Solution vector x computed"
Return: x, L, U, P =0

17. Gauss-Seidel

Algorithm 17 Gauss-Seidel Method

Input: Matrix A, vector b, initial guess x_0, tolerance tol, maximum iterations $maxIter$
Initialize: $n \leftarrow$ rows of A, $x \leftarrow x_0$, $k \leftarrow 0$, $error \leftarrow \infty$
Iterations: Empty list to store progress
while $k < maxIter$ **and** $error > tol$ **do**
  $x_{old} \leftarrow x$ {Save previous iteration values}
  for $i = 0$ **to** $n - 1$ **do**
    $sum \leftarrow 0$
    for $j = 0$ **to** $n - 1$ **do**
      if $j \neq i$ **then**
        $sum \leftarrow sum + A[i][j] \cdot x[j]$
      end if
    end for
    if $A[i][i] = 0$ **then**
      Throw Error: "Zero on diagonal at row $i + 1$. Cannot proceed."
    end if
    $x[i] \leftarrow (b[i] - sum)/A[i][i]$
  end for
  $error \leftarrow \sqrt{\sum_{i=0}^{n-1} (x[i] - x_{old}[i])^2}$
  Store Iteration: "$x^{(k+1)} = [x_1, \ldots, x_n]$, Error = $error$"
  $k \leftarrow k + 1$
end while
if $k = maxIter$ **and** $error > tol$ **then**
  Throw Error: "Maximum number of iterations reached without convergence."
end if
Return: Solution x, Iterations =0

18. Vandermonde Method (Interpolation)

Algorithm 18 Vandermonde Method (Interpolation)

Input: List of points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
Initialize: Matrix A of size $n \times n$, vector b of size n
for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    $A[i][j] \leftarrow x_i^{n-j}$
  end for
  $b[i] \leftarrow y_i$
end for
Solve: Solve the system $A \cdot \text{coef} = b$ to get the coefficients of the polynomial
Return: Polynomial coefficients =0

19. Doolittle Method (LU Factorization)

Algorithm 19 Doolittle Method (LU Factorization)

Input: Matrix A of size $n \times n$
Initialize: Matrices L and U of size $n \times n$ filled with zeros
for $i = 1$ to n **do**
  for $j = i$ to n **do**
    $U[i][j] \leftarrow A[i][j] - \sum_{k=1}^{i-1} L[i][k] \cdot U[k][j]$
  end for
  for $k = i + 1$ to n **do**
    $L[k][i] \leftarrow \frac{A[k][i] - \sum_{j=1}^{i-1} L[k][j] \cdot U[j][i]}{U[i][i]}$
  end for
end for
Return: Matrices L and U =0

20. TrazCub Method (Cubic Spline Interpolation)

Algorithm 20 TrazCub Method (Cubic Spline Interpolation)

Input: List of points $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
Initialize: Number of intervals $m \leftarrow n - 1$
Build: Construct the system for the second derivatives using boundary conditions
Solve: Solve the system to find the second derivatives at each point
for $i = 1$ to m **do**
  Construct: For each interval $[x_i, x_{i+1}]$, find the cubic polynomial using the second derivatives
end for
Return: Cubic spline function that interpolates the points =0

21. Composite Trapezoidal Rule

Algorithm 21 Trapecio Compuesto (Composite Trapezoidal Rule)

Input: Function $f(x)$, lower limit a, upper limit b, number of subintervals n
$h \leftarrow \frac{b-a}{n}$
$sum \leftarrow f(a) + f(b)$
for $i = 1$ to $n - 1$ **do**
  $x \leftarrow a + i \cdot h$
  $sum \leftarrow sum + 2 \cdot f(x)$
end for
Output: Integral approximation Integral $\leftarrow \frac{h}{2} \cdot sum$
Return: Integral approximation =0

22. Lagrange

Algorithm 22 Lagrange Interpolation Method

Input: Data points $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$
Output: Coefficients of the Lagrange polynomial $P(x)$
Initialize: $L \leftarrow []$ (to store Lagrange basis polynomials)
function POLYVAL(coefficients, x)
  Return: $\sum_{i=0}^{m} \text{coefficients}[i] \cdot x^{m-i}$, where m is the degree of the polynomial
end function
function CONV(poly1, poly2)
  Initialize: result $\leftarrow$ [0] of size (length of poly1 + length of poly2 $-$ 1)
  for $i \leftarrow 0$ **to** length of poly1 $-$ 1 **do**
    for $j \leftarrow 0$ **to** length of poly2 $-$ 1 **do**
      $\text{result}[i+j] \leftarrow \text{result}[i+j] + \text{poly1}[i] \cdot \text{poly2}[j]$
    end for
  end for
  Return: result
end function
for $i \leftarrow 1$ **to** n **do**
  aux0 $\leftarrow X$ excluding x_i
  aux $\leftarrow [1, -\text{aux0}[1]]$ (initial polynomial for $L_i(x)$)
  for $j \leftarrow 2$ **to** length of aux0 **do**
    aux $\leftarrow$ CONV(aux, $[1, -\text{aux0}[j]]$)
  end for
  normalizer $\leftarrow$ POLYVAL(aux, x_i)
  $L[i] \leftarrow$ aux/normalizer (normalize $L_i(x)$ so $L_i(x_i) = 1$)
end for
Initialize: $Coef \leftarrow$ [0] of size length($L[0]$)
for col $\leftarrow 0$ **to** length($L[0]$) $-$ 1 **do**
  $Coef[\text{col}] \leftarrow \sum_{i=1}^{n} L[i][\text{col}] \cdot Y[i]$
end for
Return: L (Lagrange basis polynomials), $Coef$ (Lagrange polynomial coefficients) =0

23. Simpson

Algorithm 23 Simpson's Rule for Numerical Integration

Input: Function $f(x)$, lower limit a, upper limit b, number of subintervals n
Output: Approximation of the integral $\int_a^b f(x)\,dx$
Validate Inputs:
if $n \leq 0$ or n is odd **then**
    Error: "Number of subintervals (n) must be a positive even integer."
    Exit.
end if
if $a \geq b$ **then**
    Error: "The lower limit must be less than the upper limit."
    Exit.
end if
Compute Step Size: $h = \frac{b-a}{n}$
Initialize: $S = f(a) + f(b)$
Iterate Over Subintervals:
for $i = 1$ to $n - 1$ **do**
    $x_i = a + i \cdot h$
    if i is odd **then**
        $S \leftarrow S + 4 \cdot f(x_i)$
    else
        $S \leftarrow S + 2 \cdot f(x_i)$
    end if
end for
Compute Integral: $I = \frac{h}{3} \cdot S$
Return: I =0