Algorithm: RBF Collocation with Localized Method for Approximate Solutions to the Navier-Stokes Equations

This algorithm numerically solves the incompressible Navier-Stokes equations using radial basis functions (RBFs) and Hermite interpolation to approximate localized solutions. Below is the step-by-step procedure with the associated mathematics.

1. Grid Initialization

Goal: Generate computational nodes (interior and boundary) in a 2D domain.

Mathematics:

$$\text{Domain: } \Omega = [0,1] \times [0,1]$$

Nodes are distributed as:

- **Boundary nodes:** $x = 0$ or $x = 1$, $y = 0$ or $y = 1$.
- **Interior nodes:** $(x, y) \in \Omega \setminus \partial\Omega$.

Implementation:

- Define evenly spaced x, y points on a unit square grid.
- Identify boundary nodes ($\partial\Omega$) and interior nodes.

2. Boundary Condition Setup

Goal: Specify velocity and pressure conditions on the domain boundaries.

Mathematics:

- Velocity (u, v):

$$u = 1,\ v = 0 \quad \text{on inflow } (y = 0), \quad u = v = 0 \quad \text{elsewhere.}$$

- Pressure (p):

$$p = 0 \quad \text{(or derived using velocity gradients at boundaries).}$$

Implementation:

- Initialize velocity fields u, v as zero everywhere, then impose boundary conditions.

3. Compute Localized Gradients and Weights

Goal: Use RBFs and Hermite interpolation to compute localized approximations of gradients and Laplacian.

Mathematics:

- **Radial Basis Function (RBF):**

$$\phi(r) = \sqrt{1 + c^2 r^2},$$

  where $r = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, and c is the shape parameter.

- Gradients and Laplacian:

$$\frac{\partial \phi}{\partial x},\ \frac{\partial^2 \phi}{\partial x^2},\ \frac{\partial \phi}{\partial y},\ \frac{\partial^2 \phi}{\partial y^2}.$$

- Augment with polynomial terms $P(x, y)$ for consistency and stability.
- Localized system for weights (w):

$$\mathbf{A}_{\text{aug}} \mathbf{w} = \mathbf{b}_{\text{aug}},$$

  where $\mathbf{A}_{\text{aug}}$ is a block matrix combining RBF and polynomial terms.

Implementation:

- Compute local neighbor sets based on a radius R_0.
- Solve the augmented system for weights.

4. Solve Momentum Equation

Goal: Update velocity fields using the momentum equation.
Mathematics:

- Momentum equation in component form:

$$\frac{\partial u}{\partial t} = -\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) + \nu\nabla^2 u - \frac{\partial p}{\partial x},$$

$$\frac{\partial v}{\partial t} = -\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) + \nu\nabla^2 v - \frac{\partial p}{\partial y},$$

  where $\nu = \frac{1}{\mathrm{Re}}$ is the kinematic viscosity.

- Time discretization (Euler):

$$u^{n+1} = u^n + \Delta t\left[\nu\nabla^2 u^n - (u^n\nabla u^n) - \nabla p^n\right],$$

  similarly for v^{n+1}.

Implementation:

- Use weights to compute gradient and Laplacian terms.
- Update u, v using time-stepping.

5. Solve Pressure Poisson Equation

Goal: Enforce incompressibility by solving the pressure Poisson equation.
Mathematics:

- Pressure Poisson equation:

$$\nabla^2 p = \nabla\cdot\mathbf{f},$$

  where $\mathbf{f} = -\left[u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right]$ is derived from velocity divergence.

- Boundary condition:

$$p = 0 \quad \text{at Dirichlet boundaries.}$$

Implementation:

- Assemble the sparse global matrix for $\nabla^2 p$ using local RBF weights.
- Solve $\mathbf{A}_p\mathbf{p} = \mathbf{rhs}$.

6. Enforce Boundary Conditions

Goal: Ensure updated velocity and pressure satisfy physical constraints.
Mathematics:

- Set $u = 0, v = 0$ at solid boundaries.
- For free-slip or inflow boundaries: - Velocity or pressure gradient imposed based on problem setup.

Implementation:

- Directly modify boundary values in arrays u, v, p.

7. Visualization

Goal: Display velocity field, pressure distribution, and convergence over time.
Implementation:

- Use `quiver` to plot velocity vectors.
- Use `scatter` for pressure distribution.
- Streamline visualization for flow trajectories.

Summary

This method applies RBF collocation with Hermite interpolation to approximate derivatives and solve the Navier-Stokes equations iteratively. The localized approach ensures computational efficiency and adaptability to arbitrary node distributions while maintaining stability and accuracy through polynomial augmentation.

Algorithm for LHMAPS Error Computation and Shape Parameter Optimization

Inputs:

- n: Array of integers specifying the number of nodes in each dimension for different simulations.
- m: Number of simulations, calculated as the length of n.

Outputs:

- mE: Maximum absolute errors for each simulation.
- $rMSE$: Root mean square errors for each simulation.
- $rELAE$: Relative errors for each simulation.
- $sPARA$: Optimized shape parameter values for each simulation.

Steps:

1. **Initialize Variables:**

- Set mE, $rMSE$, $rELAE$, and $sPARA$ as zero arrays of size m.
- Initialize a loop counter $j = 0$.

2. **Loop Over Simulations:**

- For each value of $n[i]$:
  1. Increment the loop counter: $j = j + 1$.
  2. Call the subroutine `LHMAPS_MQ_MQ_LOOCV` with input $n[i]$.
  3. Retrieve outputs:

$$\text{maxE, rmsE, relativeE, c1} = \texttt{LHMAPS\_MQ\_MQ\_LOOCV}(n[i])$$

where:

- maxE: Maximum absolute error.
- rmsE: Root mean square error.
- relativeE: Relative error.
- c1: Optimized shape parameter.

4. Store the errors and shape parameter in their respective arrays:

$$mE[j] = \text{maxE}, \quad rMSE[j] = \text{rmsE}, \quad rELAE[j] = \text{relativeE}, \quad sPARA[j] = c1.$$

3. **Plot Errors:**

- Create a log-log plot for errors:
  1. Plot n vs mE in red.
  2. Plot n vs $rMSE$ in blue.
  3. Plot n vs $rELAE$ in green.
- Set axis limits:

$$\text{axis}([\min(n), \max(n), 10^{-9}, 10^{-1}])$$

- Label the axes:

$$\text{ylabel('Errors')}, \quad \text{xlabel('N')}$$

4. **Print Shape Parameter Values:**

- Use a formatted print statement to display:

$$\text{fprintf('Shape Parameter = \%8f', sPARA)}$$

Visualization:

- The log-log plot visually represents how errors scale with increasing n.
- Different errors (mE, $rMSE$, $rELAE$) are color-coded for clarity.

Algorithm for the LHMAPS_MQ_MQ_LOOCV Subroutine

Purpose:

- Solve the Poisson equation with Dirichlet boundary conditions using Local Hermite MQ Approximation (LHMAPS).
- Compute the solution errors (maximum, relative, and RMS) compared to the exact solution.
- Determine the optimal shape parameter using Leave-One-Out Cross Validation (LOOCV).

Inputs:

- n: Number of nodes per side, yielding $(n+1)^2$ total nodes.

Outputs:

- maxE: Maximum absolute error.
- rmsE: Root mean square error.
- relativeE: Maximum relative error.
- c_1: Optimal shape parameter obtained via LOOCV.

Algorithm:

Step 1: Initialization

- Set the stencil parameters:

$$n_s = 9, \quad m = 5$$

where n_s is the number of local neighborhood nodes, and m is the number of additional Hermite interpolation points.

Step 2: Generate Collocation Nodes

- Call `Nodes_dist` to generate $(n+1)^2$ rectangular collocation nodes:

$$\text{coor} = \texttt{Nodes\_dist}(n, 1)$$

- Extract interior and boundary nodes:

$$\begin{aligned} \text{Interior nodes:} \quad & y = \text{coor}(\text{int\_ind}, 1:2), \quad \text{int\_ind} = \{i \mid \text{coor}(i,3) = 0\} \\ \text{Boundary nodes:} \quad & y_b = \text{coor}(\text{int\_ind\_b}, 1:2), \quad \text{int\_ind\_b} = \{i \mid \text{coor}(i,3) > 0\} \end{aligned}$$

- Compute the total, interior, and boundary node counts:

$$N = |\text{coor}|, \quad N_i = |\text{int\_ind}|, \quad N_b = |\text{int\_ind\_b}|$$

Step 3: Define Forcing Term, Exact Solution, and Boundary Conditions

$$\begin{aligned} f(x,y) &= y(1-e^{y-1})(2+x)e^{x-1} + x(1-e^{x-1})(2+y)e^{y-1} \\ u(x,y) &= xy(1-e^{x-1})(1-e^{y-1}) \quad \text{(Exact solution)} \\ g(x,y) &= xy(1-e^{x-1})(1-e^{y-1}) \quad \text{(Dirichlet boundary condition)} \end{aligned}$$

Step 4: Assemble Global Sparse Matrix

- Call the `localmpsmatrix_MQ_MQ_LOOCV` function:

$$[B, f_0, c_1] = \texttt{localmpsmatrix\_MQ\_MQ\_LOOCV}(\text{coor}, \text{int\_ind}, \text{int\_ind\_b}, n, n_s, m)$$

  where:

  - B: Global sparse matrix.
  - f_0: Adjustment term for forcing function.
  - c_1: Optimal shape parameter via LOOCV.

Step 5: Compute Numerical Solution

- Define global coordinates:

$$x = \text{coor}(:, 1), \quad y = \text{coor}(:, 2)$$

- Compute forcing term values:

$$\begin{aligned} & F = f(x,y), \quad F(\text{int\_ind\_b}) = 0 \quad \text{(Boundary condition adjustment)} \\ & G = g(x,y), \quad G(\text{int\_ind}) = 0 \quad \text{(Interior node adjustment)} \\ & F = F - f_0 f(x,y) + G \end{aligned}$$

- Solve for numerical solution u_N:

$$b = F(\text{int\_ind}) - B(\text{int\_ind}, \text{int\_ind\_b})F(\text{int\_ind\_b})$$

$$u_N = B(\text{int\_ind}, \text{int\_ind}) \backslash b$$

Step 6: Compute Exact Solution

- Evaluate exact solution at interior nodes:

$$u_{\text{exact}} = u(\text{coor}(\text{int\_ind}, 1), \text{coor}(\text{int\_ind}, 2))$$

Step 7: Compute Errors

- Maximum absolute error:
$$\text{maxE} = \max(|u_{\text{exact}} - u_N|)$$
- Maximum relative error:
$$\text{relativeE} = \max\left(\frac{|u_N - u_{\text{exact}}|}{\|u_{\text{exact}}\|/\sqrt{N_i}}\right)$$
- Root mean square (RMS) error:
$$\text{rmsE} = \frac{\|u_N - u_{\text{exact}}\|_2}{\sqrt{N_i}}$$

Step 8: Visualization

- Plot pointwise errors:
$$\text{scatter3}(\text{coor}(\text{int\_ind}, 1), \text{coor}(\text{int\_ind}, 2), u_N - u_{\text{exact}})$$

Algorithm for `localmpsmatrix_MQ_MQ_LOOCV`

Purpose:

- Construct the global sparse matrix for the Local Hermite Multiquadric Approximation with LOOCV (LHMAPS).
- Compute optimal shape parameter c_1 using Leave-One-Out Cross Validation (LOOCV).

Inputs:

- `coor`: Coordinates of all nodes (x, y) with a boundary flag.
- `int_ind`: Indices of interior nodes.
- `int_ind_b`: Indices of boundary nodes.
- n: Total number of nodes.
- n_s: Number of nodes in each local neighborhood.
- m: Number of Hermite interpolation points.

Outputs:

- `localmpsmatrix` (B): Global sparse matrix for the system.
- `f0`: Sparse matrix for adjustment terms.
- c_1: Optimal shape parameter from LOOCV.

Algorithm:

Step 1: Define Basis Functions

- Define the multiquadric (MQ) radial basis function and its derivatives:
$$\begin{aligned}
\phi(e, r) &= \sqrt{1 + r^2 e^2}\left(\frac{r^2}{9} + \frac{4}{9e^2}\right) - \frac{\log(\sqrt{1 + e^2 r^2} + 1)}{3e^2} \\
d_1\phi(e, r) &= -\sqrt{1 + (re)^2} \quad \text{(1st derivative)} \\
d_2\phi(e, r) &= -\sqrt{1 + (re)^2} \quad \text{(2nd derivative)} \\
d_{12}\phi(e, r) &= \frac{2e^2}{\sqrt{1 + (re)^2}} - \frac{e^4 r^2}{(1 + (re)^2)^{3/2}} \quad \text{(Mixed derivative)}
\end{aligned}$$

Step 2: Distance Functions

- Define functions for computing distances and offsets:

$$\begin{aligned} D(x,y) &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad \text{(Euclidean distance)} \\ DX(x,y) &= x_i - x_j \quad \text{(Offset in } x \text{ direction)} \\ DY(x,y) &= y_i - y_j \quad \text{(Offset in } y \text{ direction)} \end{aligned}$$

Step 3: Initialization

- Initialize global sparse matrix entries:

$$B1 = [], \quad B2 = [], \quad B3 = [], \quad f1 = [], \quad f2 = [], \quad f3 = []$$

- Build a KD-tree for efficient neighbor search:

$$\text{tree} = \texttt{KDTreeSearcher}(\text{coor}(:, 1:2))$$

- Compute the number of nodes:

$$N = |\text{coor}|, \quad N_i = |\text{int\_ind}|, \quad N_b = N - N_i$$

Step 4: Local Approximation for Interior Nodes

- For each interior node i:

1. Find n_s nearest neighbors using the KD-tree:

$$\text{id}, \text{Dis} = \texttt{knnsearch}(\text{tree}, \text{coor}(\text{int\_ind}(i), :), \text{k} = n_s)$$

2. Define local coordinates:

$$x_{\text{hat}} = \text{coor}(\text{id}, :)$$

3. Compute local distance matrices:

$$\begin{aligned} DM &= D(x_{\text{hat}}, x_{\text{hat}}) \\ DMX &= DX(x_{\text{hat}}, x_{\text{hat}}), \quad DMY = DY(x_{\text{hat}}, x_{\text{hat}}) \end{aligned}$$

4. Solve for the optimal shape parameter c_1 using LOOCV:

$$c_1 = \arg\min_c \texttt{costEps}(c, d_1\phi, \text{Dis}, \phi, DM, DMX, DMY)$$

5. Assemble the local matrix A:

$$A = \begin{bmatrix} \phi(c_1, DM) & d_2\phi(c_1, DMX, DMY) & 1 \\ d_1\phi(c_1, DMX, DMY) & d_{12}\phi(c_1, DMX, DMY) & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

6. Solve the linear system:

$$w = A^{-1}\Theta, \quad \Theta = \begin{bmatrix} d_2\phi(c_1, DMX) \\ d_{12}\phi(c_1, DMX) \\ 0 \end{bmatrix}$$

7. Store weights in the global sparse matrix:

$$B1 = [B1, \text{int\_ind}(i)], \quad B2 = [B2, \text{id}], \quad B3 = [B3, w]$$

Step 5: Handle Boundary Conditions

- For each boundary node i:

$$B1 = [B1, \text{int\_ind\_b}(i)], \quad B2 = [B2, \text{int\_ind\_b}(i)], \quad B3 = [B3, 1]$$

Step 6: Construct Sparse Matrices

- Construct the global sparse matrix:

$$\text{localmpsmatrix} = \texttt{sparse}(B1, B2, B3, N, N)$$

- Construct the adjustment matrix:

$$f_0 = \texttt{sparse}(f1, f2, f3, N, N)$$

Step 7: LOOCV Cost Function

- The cost function is given by:

$$\text{costEps}(c) = \|\text{errorvector}\|, \quad \text{errorvector} = \frac{A^{-1}\phi(c, \text{Dis})}{\text{diag}(A^{-1})}$$

Algorithm for `Nodes_dist`

Purpose:

This function generates collocation points (nodes) for solving partial differential equations using three different distribution methods. These points can include interior nodes and boundary nodes, depending on the selected mode.

Inputs:

- n: Number of subdivisions in each dimension.
- d: Distribution mode (integer):
  - `1`: Rectangular grid on a unit square.
  - `2`: Circular domain with interior and boundary nodes.
  - `3`: Sobol sequence-based distribution.

Outputs:

- `coor`: Matrix of coordinates with an additional column indicating boundary types:
  - 0: Interior node.
  - Positive integers: Different types of boundary nodes.

Algorithm:

Case 1: Rectangular Grid on Unit Square

1. Generate a uniform grid:

$$t = \text{linspace}(0, 1, n + 1)$$
$$[X, Y] = \text{meshgrid}(t, t)$$

   Flatten the grid:

$$x_1 = \text{reshape}(X, [], 1), \quad y_1 = \text{reshape}(Y, [], 1)$$

2. Initialize the coordinates matrix:

$$\text{coor} = \begin{bmatrix} x_1 & y_1 & \mathbf{0} \end{bmatrix}$$

3. Identify boundary nodes:
   - p_1: Nodes on the bottom edge excluding the corners.
   - p_2: Nodes on the right edge.
   - q_1: Nodes on the top edge excluding the corners.

- q_2: Nodes on the left edge.

Assign boundary types:

$$\text{coor}(p_1, 3) = 1, \quad \text{coor}(p_2, 3) = 2, \quad \text{coor}(q_1, 3) = 3, \quad \text{coor}(q_2, 3) = 4$$

4. Plot all nodes and highlight boundary nodes.

Case 2: Circular Domain with Interior and Boundary Nodes

1. Generate a uniform grid in $[-1, 1] \times [-1, 1]$:

$$t = \text{linspace}(-1, 1, n + 1)$$

Flatten the grid:

$$[X, Y] = \text{meshgrid}(t, t)$$
$$q = \begin{bmatrix} \text{reshape}(X, [], 1) & \text{reshape}(Y, [], 1) \end{bmatrix}$$

2. Identify interior points:

$$\text{rr} = \sqrt{q(:,1)^2 + q(:,2)^2}, \quad \text{ss} = \text{find}(\text{rr} < 1 - \frac{1}{2n})$$
$$\text{intnode} = \begin{bmatrix} q(\text{ss}, :) & \mathbf{0} \end{bmatrix}$$

3. Generate circular boundary points:

$$t = 0 : \frac{2}{n} : 2\pi - \frac{1}{n}$$
$$\text{bdpt} = \begin{bmatrix} \cos(t)' & \sin(t)' & \mathbf{1} \end{bmatrix}$$

4. Combine interior and boundary nodes:

$$\text{coor} = \begin{bmatrix} \text{intnode}; \text{bdpt} \end{bmatrix}$$

5. Plot interior and boundary nodes.

Case 3: Sobol Sequence-Based Distribution

1. Generate Sobol sequence for interior points:

$$p = \text{sobolset}(2, \text{Skip} = 1000, \text{Leap} = 100)$$
$$\text{intnode} = \text{net}(p, n^2)$$

2. Create boundary nodes using a uniform grid:

$$\text{nx} = \frac{0 : n}{n + 1}, \quad \text{ny} = \frac{n + 1 : -1 : 1}{n + 1}$$

Combine boundary points into a matrix:

$$\text{bdpt} = \begin{bmatrix} \text{nx}' & \mathbf{0} \\ \text{ones}(n + 1, 1) & \text{nx}' \\ \text{ny}' & \text{ones}(n + 1, 1) \\ \mathbf{0} & \text{ny}' \end{bmatrix}$$

3. Filter interior nodes to ensure no overlap with boundary:

$$\text{D} = \text{DistanceMatrix}(\text{intnode}, \text{bdpt})$$

Remove nodes within a threshold h:

$$h = \frac{1}{n + 1}, \quad \text{Row} = \text{find}(\sqrt{\text{D}} < h)$$
$$\text{intnode}(\text{Row}, :) = []$$

4. Combine interior and boundary nodes:

$$\text{coor} = [\text{intnode}; \text{bdpt}]$$

Assign labels for interior and boundary nodes:

$$\text{coor}(:, 3) = [\mathbf{0}; \mathbf{1}]$$

5. Plot all nodes.

Default Case:

If d is invalid, display an error message:

$$\text{disp('Error!')}$$

Algorithms for LHMAPS Implementation

1. Generating LHMAPS Solutions and Computing Errors

Mathematical Formulation:

1. **Nodes and Errors Initialization:**

$$n = \{10, 20, 40, 80\}, \quad N = (n+1)^2.$$

Initialize error arrays:

$mE,\ rMSE,\ rELAE,\ sPARA$ (Maximum Error, RMS Error, Relative Error, Shape Parameter).

2. **Iterative Computation Over Nodes:**
For each grid size $n_i \in n$:

- Call the subroutine LHMAPS_MQ_MQ_LOOCV(n_i) to compute:

$$\text{maxE},\ \text{rmsE},\ \text{relativeE},\ c_1.$$

- Store the results in the corresponding error arrays:

$$mE[j] = \text{maxE}, \quad rMSE[j] = \text{rmsE},$$
$$rELAE[j] = \text{relativeE}, \quad sPARA[j] = c_1.$$

3. **Visualization:**

- Plot the errors on a logarithmic scale:

$$\text{Log-Log Plot: } mE,\ rMSE,\ rELAE \text{ against } n.$$

- Print the computed shape parameter values.

2. LHMAPS MQ LOOCV Subroutine

Mathematical Formulation:

1. **Problem Setup:**

- Define a rectangular grid with $(n+1)^2$ nodes.
- Solve the Poisson equation with Dirichlet boundary conditions:

$$-\Delta u = f(x, y), \quad u = g(x, y) \text{ on the boundary.}$$

- Exact solution:

$$u(x, y) = xy(1 - e^{x-1})(1 - e^{y-1}).$$

2. **Formulation of the RBF Matrix:**

   - Use a compact stencil with Hermite derivatives:

$$u(x, y) = \sum_{j=1}^{n} a_j \phi(\|x - x_j\|, c) + \text{polynomial terms}.$$

3. **Error Computation:**

$$\begin{aligned} \text{Maximum Error:} \quad & \text{maxE} = \max |u_{\text{exact}} - u_{\text{numerical}}|, \\ \text{RMS Error:} \quad & \text{rmsE} = \sqrt{\frac{1}{N_i} \sum_{j=1}^{N_i} (u_{\text{exact}} - u_{\text{numerical}})^2}, \\ \text{Relative Error:} \quad & \text{relativeE} = \max \left(\frac{|u_{\text{numerical}} - u_{\text{exact}}|}{\|u_{\text{exact}}\|} \right). \end{aligned}$$

3. Local Matrix and Shape Parameter Optimization

Mathematical Formulation:

1. **Matrix Formulation:**

   - Use the Multiquadric RBF:

$$\phi(c, r) = \sqrt{1 + c^2 r^2}.$$

   - Construct the matrix for local collocation:

$$\mathbf{A} = \begin{bmatrix} \phi(c, D) & \nabla\phi & \text{polynomials} \\ \nabla\phi & \Delta\phi & 0 \\ \text{polynomials}^T & 0 & 0 \end{bmatrix}.$$

2. **Shape Parameter Optimization:**

   - Minimize the error using LOOCV:

$$c_1 = \arg \min_{c \in [0.1, 10]} \|\mathbf{A}^{-1}\mathbf{rhs} - \mathbf{u}\|^2.$$

3. **Sparse Matrix Construction:**

   - Populate the global sparse matrix:

$$\mathbf{B} \text{ (global matrix)}, \quad \mathbf{f_0} \text{ (boundary conditions)}.$$

4. Node Distribution Algorithm

Mathematical Formulation:

1. **Uniform Grid (Mode 1):**

$$x = \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}, \quad y = \{0, \frac{1}{n}, \frac{2}{n}, \ldots, 1\}.$$

2. **Circular Domain (Mode 2):**

$$r = \sqrt{x^2 + y^2} < 1 - \frac{1}{2n}.$$

3. **Sobol Sequence (Mode 3):**

   - Generate low-discrepancy points:

$$(x_i, y_i) \in [0, 1]^2, \quad \text{optimized for uniform coverage.}$$

   - Filter points to avoid overlap with boundaries:

$$\text{threshold } h = \frac{1}{n+1}.$$