

```

1  //Utility.h
2
3  #include <cmath>
4  #include <fstream>
5  #include <iostream>
6  #include <string>
7  #include <windows.h>
8  #include <vector>
9
10 class Save
11 {
12 public:
13     //public constructor/s
14     Save(const double h, const double k, const size_t M, const size_t N, const std::
        vector<std::vector<double>>& u, const std::string save_file_name);
15 };
16
17 class MidPoint
18 {
19 public:
20     //public constructor/s
21     MidPoint(const double h, const size_t M, double& midpoint);
22 };
23
24 class IniVector
25 {
26 public:
27     //public constructor/s
28     IniVector(const size_t M, std::vector<double>& vec);
29 };
30
31 class IniMatrix
32 {
33 public:
34     //public constructor/s
35     IniMatrix(const size_t M, const size_t N, std::vector<std::vector<double>>& mat);
36 };
37
38 class InitialAndBoundaryConditionsMatrix
39 {
40 public:
41     //public constructor/s
42     InitialAndBoundaryConditionsMatrix(const double h, const size_t M, const size_t N,
        std::vector<std::vector<double>>& mat, const double midpoint);
43 };
44
45 class InitialAndBoundaryConditionsVector
46 {
47 public:
48     //public constructor/s
49     InitialAndBoundaryConditionsVector(const double h, const size_t M, std::vector<double
        >& vec, const double midpoint);
50 };
51
52 class SetMandN
53 {
54 public:
55     //public constructor/s
56     SetMandN(const double h, const double k, size_t& M, size_t& N);
57 };
58
59 class SetM
60 {
61 public:
62     //public constructor/s
63     SetM(const double h, size_t& M);
64 };

```

```

1 //Utility.cpp
2
3 #include "0.0.Utility.h"
4
5 Save::Save(const double h, const double k, const size_t M, const size_t N, const std::
vector<std::vector<double>>& u, const std::string save_file_name)
6 {
7     std::ofstream out;
8     out.open(save_file_name);
9     if (out.is_open())
10     {
11         for (size_t j = 0; j < N; j++)
12         {
13             for (size_t i = 0; i < M; i++)
14             {
15                 out << i * h << "," << j * k << "," << u[i][j] << "\n";
16             }
17         }
18     }
19     else
20     {
21         throw std::string("Could not save to file!");
22     }
23     out.close();
24 }
25
26 MidPoint::MidPoint(const double h, const size_t M, double& midpoint)
27 {
28     midpoint = ((M - 1) * h) / 2;;
29 }
30
31 IniVector::IniVector(const size_t M, std::vector<double>& vec)
32 {
33     vec.resize(M);
34     for (size_t i = 0; i < M; i++)
35     {
36         vec[i] = 0;
37     }
38 }
39
40 IniMatrix::IniMatrix(const size_t M, const size_t N, std::vector<std::vector<double>>&
mat)
41 {
42     mat.resize(M, std::vector<double>(N));
43     for (size_t i = 0; i < M; i++)
44     {
45         for (size_t j = 0; j < N; j++)
46         {
47             mat[i][j] = 0;
48         }
49     }
50 }
51
52 InitialAndBoundaryConditionsMatrix::InitialAndBoundaryConditionsMatrix(const double h,
const size_t M, const size_t N, std::vector<std::vector<double>>& mat, const double
midpoint)
53 {
54     for (size_t i = 0; i < M; i++)
55     {
56         mat[i][0] = exp(-pow((i * h - midpoint), 2));
57     }
58     for (size_t j = 0; j < N; j++)
59     {
60         mat[0][j] = 0;
61         mat[M - 1][j] = 0;
62     }
63 }
64
65 InitialAndBoundaryConditionsVector::InitialAndBoundaryConditionsVector(const double h,

```

```

66     const size_t M, std::vector<double>& vec, const double midpoint)
67     {
68         for (size_t i = 0; i < M; i++)
69         {
70             vec[i] = exp(-pow((i * h - midpoint), 2));
71         }
72         vec[0] = 0;
73         vec[M - 1] = 0;
74     }
75
76     SetMandN::SetMandN(const double h, const double k, size_t& M, size_t& N)
77     {
78         M = (10 / h) + 1;
79         N = (5 / k) + 1;
80     }
81
82     SetM::SetM(const double h, size_t& M)
83     {
84         M = (10 / h) + 1;
85     }

```

```
1  //MatrixOperations.h
2
3  class MatrixCopy
4  {
5  public:
6      MatrixCopy(const size_t M, const std::vector<std::vector<double>>& mat_from, std:::
          vector<std::vector<double>>& mat_to);
7  };
8
9  class VectorCopy
10 {
11 public:
12     VectorCopy(const size_t M, const std::vector<double>& vec_from, std::vector<double>&
          vec_to);
13 };
```

```
1  //MatrixOperations.cpp
2
3  #include "0.0.Utility.h"
4  #include "0.1.MatrixOperations.h"
5
6  MatrixCopy::MatrixCopy(const size_t M, const std::vector<std::vector<double>>& mat_from,
7  std::vector<std::vector<double>>& mat_to)
8  {
9      for (size_t i = 0; i < M; i++)
10     {
11         for (size_t j = 0; j < M; j++)
12         {
13             mat_to[i][j] = mat_from[i][j];
14         }
15     }
16
17  VectorCopy::VectorCopy(const size_t M, const std::vector<double>& vec_from, std::vector<
18  double>& vec_to)
19  {
20      for (size_t i = 0; i < M; i++)
21      {
22          vec_to[i] = vec_from[i];
23      }
24  }
```

```
1 //LUGaussSpecial.h
2
3 class LUGaussSpecial
4 {
5 public:
6     //public constructor/s
7     LUGaussSpecial(const size_t M, const double s_0, const double s_1, const double s_2,
8         std::vector<double>& b, std::vector<double>& x);
9 private:
10    //private function/s
11    void SpecialGaussianElimination(std::vector<double>& b, std::vector<double>& x);
12    void ExtractTriVectors(const double s_0, const double s_1, const double s_2);
13    //private variable/s
14    const size_t M;
15    std::vector<double> a{};
16    std::vector<double> c{};
17    std::vector<double> d{};
18 };
19
```

```

1  //LUGaussSpecial.cpp
2
3  #include "0.0.Utility.h"
4  #include "0.1.MatrixOperations.h"
5  #include "0.2.LUGaussSpecial.h"
6
7  //public constructor/s
8  LUGaussSpecial::LUGaussSpecial(const size_t M, const double s_0, const double s_1, const
double s_2, std::vector<double>& b, std::vector<double>& x)
9      : M(M)
10 {
11     IniVector(M, a);
12     IniVector(M, c);
13     IniVector(M, d);
14
15     ExtractTriVectors(s_0, s_1, s_2);
16     SpecialGaussianElimination(b, x);
17 }
18 //private function/s
19 void LUGaussSpecial::SpecialGaussianElimination(std::vector<double>& b, std::vector<
double>& x)
20 {
21     for (size_t i = 1; i < M; i++)
22     {
23         d[i] = d[i] - (a[i - 1] / d[i - 1]) * c[i - 1];
24         b[i] = b[i] - (a[i - 1] / d[i - 1]) * b[i - 1];
25     }
26     x[M - 1] = b[M - 1] / d[M - 1];
27     for (int i = M - 2; i >= 0; i--)
28     {
29         x[i] = (b[i] - (c[i] * x[i + 1])) / d[i];
30     }
31 }
32 void LUGaussSpecial::ExtractTriVectors(const double s_0, const double s_1, const double
s_2)
33 {
34     for (size_t i = 0; i < M; i++)
35     {
36         a[i] = s_0;
37         d[i] = s_1;
38         c[i] = s_2;
39     }
40 }

```

```

1  //Tri.h
2
3  class Tri
4  {
5  public:
6      //public constructor/s
7      Tri(double s_0, double s_1, double s_2, size_t M, size_t column, std::vector<double>&
        b, std::vector<std::vector<double>>& mat);
8  private:
9      //private function/s
10     void ExecuteTri(const double s_0, const double s_1, const double s_2, const size_t
        column, std::vector<double>& b, std::vector<std::vector<double>>& mat);
11     void AssignResults(const size_t column, std::vector<std::vector<double>>& mat);
12     //private variable/s
13     const size_t M;
14     std::vector<double> x{};
15 };

```



```

1  //Tri.cpp
2
3  #include "0.0.Utility.h"
4  #include "0.2.LUGaussSpecial.h"
5  #include "0.3.Tri.h"
6
7  //public constructor/s
8  Tri::Tri(double s_0, double s_1, double s_2, size_t M, size_t column, std::vector<double
9  >& b, std::vector<std::vector<double>>& mat)
10     : M(M)
11 {
12     IniVector(M, x);
13     ExecuteTri(s_0, s_1, s_2, column, b, mat);
14 }
15 //private function/s
16 void Tri::ExecuteTri(const double s_0, const double s_1, const double s_2, const size_t
17 column, std::vector<double>& b, std::vector<std::vector<double>>& mat)
18 {
19     LUGaussSpecial solve(M, s_0, s_1, s_2, b, x);
20     AssignResults(column, mat);
21 }
22 void Tri::AssignResults(const size_t column, std::vector<std::vector<double>>& mat)
23 {
24     for (size_t i = 1; i < M - 1; i++)
25     {
26         mat[i][column + 1] = x[i];
27     }
28 }

```

```

1 //Eq1.h
2
3 //par_u/par_t + par_u/par_x = 0 differential equation using Euler + Runge-Kutta 2nd
  order methods
4
5 class ParDiffEq1Euler
6 {
7 public:
8     //public constructor/s
9     ParDiffEq1Euler(double h, double k);
10    //public function/s
11    void Euler();
12 private:
13    //private variable/s
14    double midpoint{};
15    size_t M{};
16    size_t N{};
17    std::vector<std::vector<double>> u{};
18
19    const double h;
20    const double k;
21 };
22
23 class ParDiffEq1RK2
24 {
25 public:
26     //public constructor/s
27     ParDiffEq1RK2(double h, double k);
28     //public function/s
29     void RK2();
30 private:
31     //private variable/s
32     double midpoint{};
33     size_t M{};
34     size_t N{};
35     std::vector<std::vector<double>> u{};
36     std::vector<double> v{};
37
38     const double h;
39     const double k;
40 };

```

```

1 //Eq1.cpp
2
3 #include "0.0.Utility.h"
4 #include "1.Eq1.h"
5
6 //public constructor/s
7 ParDiffEq1Euler::ParDiffEq1Euler(double h, double k)
8     : h(h), k(k)
9 {
10     SetMandN(h, k, M, N);
11     MidPoint(h, M, midpoint);
12     IniMatrix(M, N, u);
13     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
14 }
15 //public function/s
16 void ParDiffEq1Euler::Euler()
17 {
18     const double s{ k / (2 * h) };
19
20     for (size_t j = 0; j < N - 1; j++)
21     {
22         for (size_t i = 1; i < M - 1; i++)
23         {
24             u[i][j + 1] = u[i][j] - s * (u[i + 1][j] - u[i - 1][j]);
25         }
26     }
27
28     std::string save_file_name{ "Results/Eq1/Euler_Eq_1_h_" + std::to_string(h) + "_k_" +
29         std::to_string(k) + ".csv" };
30
31     Save(h, k, M, N, u, save_file_name);
32 }
33 //public constructor/s
34 ParDiffEq1RK2::ParDiffEq1RK2(double h, double k)
35     : h(h), k(k)
36 {
37     SetMandN(h, k, M, N);
38     MidPoint(h, M, midpoint);
39     IniVector(M, v);
40     IniMatrix(M, N, u);
41     InitialAndBoundaryConditionsVector(h, M, v, midpoint);
42     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
43 }
44 //public function/s
45 void ParDiffEq1RK2::RK2()
46 {
47     const double r{ k / (4 * h) };
48     const double s{ k / (2 * h) };
49
50     for (size_t j = 0; j < N - 1; j++)
51     {
52         for (size_t i = 1; i < M - 1; i++)
53         {
54             v[i] = u[i][j] - r * (u[i + 1][j] - u[i - 1][j]);
55         }
56         for (size_t i = 1; i < M - 1; i++)
57         {
58             u[i][j + 1] = u[i][j] - s * (v[i + 1] - v[i - 1]);
59         }
60     }
61
62
63     std::string save_file_name{ "Results/Eq1/RK2_Eq_1_h_" + std::to_string(h) + "_k_" +
64         std::to_string(k) + ".csv" };
65
66     Save(h, k, M, N, u, save_file_name);
67 }

```

```

1 //Eq2.h
2
3 //par_u/par_t + u*par_u/par_x = 0 differential equation using Euler + Runge-Kutta 2nd
  order methods
4
5 class ParDiffEq2Euler
6 {
7 public:
8     //public constructor/s
9     ParDiffEq2Euler(double h, double k);
10    //public function/s
11    void Euler();
12 private:
13    //private variable/s
14    double midpoint{};
15    size_t M{};
16    size_t N{};
17    std::vector<std::vector<double>> u{};
18
19    const double h;
20    const double k;
21 };
22
23 class ParDiffEq2RK2
24 {
25 public:
26     //public constructor/s
27     ParDiffEq2RK2(double h, double k);
28     //public function/s
29     void RK2();
30 private:
31     //private variable/s
32     double midpoint{};
33     size_t M{};
34     size_t N{};
35     std::vector<std::vector<double>> u{};
36     std::vector<double> v{};
37
38     const double h;
39     const double k;
40 };

```

```

1 //Eq2.cpp
2
3 #include "0.0.Utility.h"
4 #include "2.Eq2.h"
5
6 //public constructor/s
7 ParDiffEq2Euler::ParDiffEq2Euler(double h, double k)
8     : h(h), k(k)
9 {
10     SetMandN(h, k, M, N);
11     MidPoint(h, M, midpoint);
12     IniMatrix(M, N, u);
13     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
14 }
15 //public function/s
16 void ParDiffEq2Euler::Euler()
17 {
18     const double s{ k / (4 * h) };
19
20     for (size_t j = 0; j < N - 1; j++)
21     {
22         for (size_t i = 1; i < M - 1; i++)
23         {
24             u[i][j + 1] = u[i][j] - s * (pow(u[i + 1][j], 2) - pow(u[i - 1][j], 2));
25         }
26     }
27
28     std::string save_file_name{ "Results/Eq2/Euler_Eq_2_h_" + std::to_string(h) + "_k_" +
29         std::to_string(k) + ".csv" };
30
31     Save(h, k, M, N, u, save_file_name);
32 }
33 //public constructor/s
34 ParDiffEq2RK2::ParDiffEq2RK2(double h, double k)
35     : h(h), k(k)
36 {
37     SetMandN(h, k, M, N);
38     MidPoint(h, M, midpoint);
39     IniVector(M, v);
40     IniMatrix(M, N, u);
41     InitialAndBoundaryConditionsVector(h, M, v, midpoint);
42     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
43 }
44 //public function/s
45 void ParDiffEq2RK2::RK2()
46 {
47     const double r{ k / (8 * h) };
48     const double s{ k / (4 * h) };
49
50     for (size_t j = 0; j < N - 1; j++)
51     {
52         for (size_t i = 1; i < M - 1; i++)
53         {
54             v[i] = u[i][j] - r * (pow(u[i + 1][j], 2) - pow(u[i - 1][j], 2));
55         }
56         for (size_t i = 1; i < M - 1; i++)
57         {
58             u[i][j + 1] = u[i][j] - s * (pow(v[i + 1], 2) - pow(v[i - 1], 2));
59         }
60     }
61
62     std::string save_file_name{ "Results/Eq2/RK2_Eq_2_h_" + std::to_string(h) + "_k_" +
63         std::to_string(k) + ".csv" };
64
65     Save(h, k, M, N, u, save_file_name);
66 }

```

```

1 //Eq3.h
2
3 //par_u/par_t - beta*par_u^2/par_x^2 = 0 differential equation using Euler + Runge-Kutta
  2nd order + implicit methods
4
5 class ParDiffEq3Euler
6 {
7 public:
8     //public constructor/s
9     ParDiffEq3Euler(double h, double k);
10    //public function/s
11    void Euler();
12 private:
13    //private variable/s
14    double midpoint{};
15    size_t M{};
16    size_t N{};
17    std::vector<std::vector<double>> u{};
18
19    double beta;
20    const double h;
21    const double k;
22 };
23
24 class ParDiffEq3RK2
25 {
26 public:
27     //public constructor/s
28     ParDiffEq3RK2(double h, double k);
29     //public function/s
30     void RK2();
31 private:
32     //private variable/s
33     double midpoint{};
34     size_t M{};
35     size_t N{};
36     std::vector<std::vector<double>> u{};
37     std::vector<double> v{};
38
39     double beta;
40     const double h;
41     const double k;
42 };
43
44 class ParDiffEq3Implicit
45 {
46 public:
47     //public constructor/s
48     ParDiffEq3Implicit(double h, double k);
49     //public function/s
50     void Implicit();
51 private:
52     //private function/s
53     void SetVectorb(const size_t column);
54     //private variable/s
55     double midpoint{};
56     size_t M{};
57     std::vector<std::vector<double>> u{};
58     std::vector<double> b{};
59
60     double beta;
61     const double h;
62     const double k;
63 };

```

```

1 //Eq3.cpp
2
3 #include "0.0.Utility.h"
4 #include "0.3.Tri.h"
5 #include "3.Eq3.h"
6
7 //public constructor/s
8 ParDiffEq3Euler::ParDiffEq3Euler(double h, double k)
9     : h(h), k(k)
10 {
11     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
12
13     SetMandN(h, k, M, N);
14     MidPoint(h, M, midpoint);
15     IniMatrix(M, N, u);
16     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
17 }
18 //public function/s
19 void ParDiffEq3Euler::Euler()
20 {
21     const double s{ (beta * k) / pow(h,2) };
22
23     for (size_t j = 0; j < N - 1; j++)
24     {
25         for (size_t i = 1; i < M - 1; i++)
26         {
27             u[i][j + 1] = u[i][j] + s * (u[i + 1][j] - 2 * u[i][j] + u[i - 1][j]);
28         }
29     }
30
31     std::string save_file_name{ "Results/Eq3/Euler_Eq_3_h_" + std::to_string(h) + "_k_" +
32         std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };
33
34     Save(h, k, M, N, u, save_file_name);
35 }
36 //public constructor/s
37 ParDiffEq3RK2::ParDiffEq3RK2(double h, double k)
38     : h(h), k(k)
39 {
40     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
41
42     SetMandN(h, k, M, N);
43     MidPoint(h, M, midpoint);
44     IniVector(M, v);
45     IniMatrix(M, N, u);
46     InitialAndBoundaryConditionsVector(h, M, v, midpoint);
47     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
48 }
49 //public function/s
50 void ParDiffEq3RK2::RK2()
51 {
52     const double r{ (beta * k) / (2 * pow(h,2)) };
53     const double s{ (beta * k) / pow(h,2) };
54
55     for (size_t j = 0; j < N - 1; j++)
56     {
57         for (size_t i = 1; i < M - 1; i++)
58         {
59             v[i] = u[i][j] + r * (u[i + 1][j] - 2 * u[i][j] + u[i - 1][j]);
60         }
61         for (size_t i = 1; i < M - 1; i++)
62         {
63             u[i][j + 1] = u[i][j] + s * (v[i + 1] - 2 * v[i] + v[i - 1]);
64         }
65     }
66
67     std::string save_file_name{ "Results/Eq3/RK2_Eq_3_h_" + std::to_string(h) + "_k_" +
68         std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };

```

```

68
69     Save(h, k, M, N, u, save_file_name);
70 }
71
72 //public constructor/s
73 ParDiffEq3Implicit::ParDiffEq3Implicit(double h, double k)
74     : h(h), k(k)
75 {
76     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
77
78     SetM(h, M);
79     MidPoint(h, M, midpoint);
80     IniMatrix(M, M, u);
81     IniVector(M, b);
82     InitialAndBoundaryConditionsMatrix(h, M, M, u, midpoint);
83 }
84 //public function/s
85 void ParDiffEq3Implicit::Implicit()
86 {
87     const double s{ (beta * k) / pow(h,2) };
88     const double s_0{ -s };
89     const double s_1{ 1 + 2 * s };
90     const double s_2{ -s };;
91
92     for (size_t j = 0; j < M-1; j++)
93     {
94         SetVectorb(j);
95         Tri solve(s_0, s_1, s_2, M, j, b, u);
96     }
97
98     std::string save_file_name{ "Results/Eq3/Implicit_Eq_3_h_" + std::to_string(h) +
99     "_k_" + std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };
100
101     Save(h, k, M, M, u, save_file_name);
102 }
103 //private function/s
104 void ParDiffEq3Implicit::SetVectorb(const size_t column)
105 {
106     for (size_t i = 0; i < M; i++)
107     {
108         b[i] = u[i][column];
109     }
110 }

```



```

1 //Burgers.h
2
3 //Burgers' equation:  $\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \beta \frac{\partial^2 u}{\partial x^2} = 0$  using
  Euler + Runge-Kutta 2nd order + implicit methods
4
5 class ParDiffEqBurgersEuler
6 {
7 public:
8     //public constructor/s
9     ParDiffEqBurgersEuler(double h, double k);
10    //public function/s
11    void Euler();
12 private:
13    //private variable/s
14    double midpoint{};
15    size_t M{};
16    size_t N{};
17    std::vector<std::vector<double>> u{};
18
19    double beta;
20    double h;
21    double k;
22 };
23
24 class ParDiffEqBurgersRK2
25 {
26 public:
27     //public constructor/s
28     ParDiffEqBurgersRK2(double h, double k);
29    //public function/s
30    void RK2();
31 private:
32    //private variable/s
33    double midpoint{};
34    size_t M{};
35    size_t N{};
36    std::vector<std::vector<double>> u{};
37    std::vector<double> v{};
38
39    double beta;
40    const double h;
41    const double k;
42 };
43
44 class ParDiffEqBurgersImplicit
45 {
46 public:
47     //public constructor/s
48     ParDiffEqBurgersImplicit(double h, double k);
49    //public function/s
50    void Implicit();
51 private:
52    //private function/s
53    void SetVectorb(const size_t column);
54    //private variable/s
55    double midpoint{};
56    size_t M{};
57    std::vector<std::vector<double>> u{};
58    std::vector<double> b{};
59
60    double beta;
61    double h;
62    double k;
63 };

```

```

1 //Burgers.cpp
2
3 #include "0.0.Utility.h"
4 #include "0.3.Tri.h"
5 #include "4.Burgers.h"
6
7 //public constructor/s
8 ParDiffEqBurgersEuler::ParDiffEqBurgersEuler(double h, double k)
9     : h(h), k(k)
10 {
11     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
12
13     SetMandN(h, k, M, N);
14     MidPoint(h, M, midpoint);
15     IniMatrix(M, N, u);
16     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
17 }
18 //public function/s
19 void ParDiffEqBurgersEuler::Euler()
20 {
21     const double s_0{ k / (4 * h) };
22     const double s_1{ (beta * k) / pow(h,2) };
23
24     for (size_t j = 0; j < N - 1; j++)
25     {
26         for (size_t i = 1; i < M - 1; i++)
27         {
28             u[i][j + 1] = u[i][j] - s_0 * (pow(u[i + 1][j], 2) - pow(u[i - 1][j], 2)) +
29                 s_1 * (u[i + 1][j] - 2 * u[i][j] + u[i - 1][j]);
30         }
31
32         std::string save_file_name{ "Results/Eq4/Euler_Burgers_h_" + std::to_string(h) +
33             "_k_" + std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };
34
35         Save(h, k, M, N, u, save_file_name);
36     }
37 //public constructor/s
38 ParDiffEqBurgersRK2::ParDiffEqBurgersRK2(double h, double k)
39     : h(h), k(k)
40 {
41     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
42
43     SetMandN(h, k, M, N);
44     MidPoint(h, M, midpoint);
45     IniVector(M, v);
46     IniMatrix(M, N, u);
47     InitialAndBoundaryConditionsVector(h, M, v, midpoint);
48     InitialAndBoundaryConditionsMatrix(h, M, N, u, midpoint);
49 }
50 //public function/s
51 void ParDiffEqBurgersRK2::RK2()
52 {
53     const double r_0{ k / (8 * h) };
54     const double r_1{ (beta * k) / (2 * pow(h,2)) };
55
56     const double s_0{ k / (4 * h) };
57     const double s_1{ (beta * k) / pow(h,2) };
58
59     for (size_t j = 0; j < N - 1; j++)
60     {
61         for (size_t i = 1; i < M - 1; i++)
62         {
63             v[i] = u[i][j] - r_0 * (pow(u[i + 1][j], 2) - pow(u[i - 1][j], 2)) + r_1 * (u
64                 [i + 1][j] - 2 * u[i][j] + u[i - 1][j]);
65         }
66         for (size_t i = 1; i < M - 1; i++)
67         {

```

```

67         u[i][j + 1] = u[i][j] - s_0 * (pow(v[i + 1], 2) - pow(v[i - 1], 2)) + s_1 * (
        v[i + 1] - 2 * v[i] + v[i - 1]);
68     }
69 }
70
71 std::string save_file_name{ "Results/Eq4/RK2_Burgers_h_" + std::to_string(h) + "_k_"
+ std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };
72
73 Save(h, k, M, N, u, save_file_name);
74 }
75
76 //public constructor/s
77 ParDiffEqBurgersImplicit::ParDiffEqBurgersImplicit(double h, double k)
78 : h(h), k(k)
79 {
80     beta = ((0.5 * pow(h, 2)) / k) - 0.01; /*satisfy (beta*k)/h^2 < 0.5 for stability*/
81
82     SetM(h, M);
83     MidPoint(h, M, midpoint);
84     IniMatrix(M, M, u);
85     IniVector(M, b);
86     InitialAndBoundaryConditionsMatrix(h, M, M, u, midpoint);
87 }
88 //public function/s
89 void ParDiffEqBurgersImplicit::Implicit()
90 {
91     const double s{ (beta * k) / pow(h,2) };
92     const double s_0{ -s };
93     const double s_1{ 1 + 2 * s };
94     const double s_2{ -s };
95
96     for (size_t j = 0; j < M - 1; j++)
97     {
98         SetVectorb(j);
99         Tri solve(s_0, s_1, s_2, M, j, b, u);
100     }
101
102     std::string save_file_name{ "Results/Eq4/Implicit_Burgers_h_" + std::to_string(h) +
    "_k_" + std::to_string(k) + "_beta_" + std::to_string(beta) + ".csv" };
103
104     Save(h, k, M, M, u, save_file_name);
105 }
106 void ParDiffEqBurgersImplicit::SetVectorb(const size_t column)
107 {
108     b[0] = u[0][column];
109     for (size_t i = 1; i < M - 1; i++)
110     {
111         b[i] = u[i][column] - (k / (4 * h)) * (pow(u[i + 1][column], 2) - pow(u[i - 1][
            column], 2));
112     }
113     b[M - 1] = u[M - 1][column];
114 }

```

```

1 //main.cpp
2
3 #include "0.0.Utility.h"
4 #include "1.Eq1.h"
5 #include "2.Eq2.h"
6 #include "3.Eq3.h"
7 #include "4.Burgers.h"
8
9 const static double h{ 0.1 }; /*set your h here*/
10 const static double k{ 0.005 }; /*set your k here*/
11
12 void DemoParDiffEq1()
13 {
14     ParDiffEq1Euler eq1_eul(h, k);
15     eq1_eul.Euler();
16
17     ParDiffEq1RK2 eq1_rk2(h, k);
18     eq1_rk2.RK2();
19 }
20 void DemoParDiffEq2()
21 {
22     ParDiffEq2Euler eq2_eul(h, k);
23     eq2_eul.Euler();
24
25     ParDiffEq2RK2 eq2_rk2(h, k);
26     eq2_rk2.RK2();
27 }
28 void DemoParDiffEq3()
29 {
30     ParDiffEq3Euler eq3_eul(h, k);
31     eq3_eul.Euler();
32
33     ParDiffEq3RK2 eq3_rk2(h, k);
34     eq3_rk2.RK2();
35
36     ParDiffEq3Implicit eq3_implicit(h, k);
37     eq3_implicit.Implicit();
38 }
39 void DemoBurgersEq()
40 {
41     ParDiffEqBurgersEuler Burgers_eul(h, k);
42     Burgers_eul.Euler();
43
44     ParDiffEqBurgersRK2 Burgers_rk2(h, k);
45     Burgers_rk2.RK2();
46
47     ParDiffEqBurgersImplicit Burgers_implicit(h, k);
48     Burgers_implicit.Implicit();
49 }
50
51 void MainMenu(bool& run)
52 {
53     size_t choice{ 0 };
54     const size_t final_choice{ 4 };
55
56     system("cls");
57     do
58     {
59         std::cout << "0. par_u/par_t + par_u/par_x = 0, PARABOLIC" << "\n";
60         std::cout << "1. par_u/par_t + u*par_u/par_x = 0, PARABOLIC" << "\n";
61         std::cout << "2. par_u/par_t - beta*par_u^2/par_x^2 = 0, beta > 0 => ELIPTIC,  
beta < 0 => HYPERBOLIC" << "\n";
62         std::cout << "3. Burgers' equation: par_u/par_t + u*par_u/par_x -  
beta*par_u^2/par_x^2=0 = 0 , beta > 0 => ELIPTIC, beta < 0 => HYPERBOLIC" << "\n";
63         std::cout << "4. Exit" << "\n";
64         std::cin >> choice;
65         if (choice > final_choice)
66         {

```

```

67         std::cout << "Wrong choice!";
68         Sleep(1000);
69         system("cls");
70     }
71     system("cls");
72 } while (choice > final_choice);
73 if (choice == 0)
74 {
75     DemoParDiffEq1();
76 }
77 else if (choice == 1)
78 {
79     DemoParDiffEq2();
80 }
81 else if (choice == 2)
82 {
83     DemoParDiffEq3();
84 }
85 else if (choice == 3)
86 {
87     DemoBurgersEq();
88 }
89 else if (choice == final_choice)
90 {
91     run = false;
92 }
93 else
94 {
95     throw std::string("Fatal Error!");
96 }
97 }
98 int main()
99 {
100     bool run{ true };
101
102     while (run == true)
103     {
104         MainMenu(run);
105     }
106     return 0;
107 }

```