1 基础算法

对于两点边值问题：

$$\begin{cases} \varepsilon \dfrac{d^2y}{dx^2} + \dfrac{dy}{dx} = a & 0 < a < 1 \\ y(0) = 0, y(1) = 1 \end{cases}$$

通过离散化处理得到先行方程组：

$$(\varepsilon + h)\, y_{i+1} - (2\varepsilon + h)\, y_i + \varepsilon y_{i-1} = ah^2$$

其中 h 是利用商差近似求导时设置的系数，考虑编制条件后，可表示为矩阵形式：

$$\begin{pmatrix} -(2\varepsilon+h) & \varepsilon+h & & & \\ \varepsilon & -(2\varepsilon+h) & \varepsilon+h & & \\ & \varepsilon & -(2\varepsilon+h) & \ddots & \\ & & \ddots & \ddots & \varepsilon+h \\ & & & \varepsilon & \varepsilon+h \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} ah^2 \\ ah^2 \\ ah^2 \\ \vdots \\ ah^2 - (\varepsilon+h) \end{pmatrix}$$

其中 y_i 的精确值 $y_i^* = y(x_i) = \dfrac{1-a}{1-e^{-1/\varepsilon}}\left(1 - e^{-(x_i/\varepsilon)}\right) + ax_i$ 由解析解直接得出，其中的 $x_i = ih$, $i = 1, 2, 3, \cdots, n-1$，而 n 是离散化时区间的分割系数。

在上述理论基础上，我们可以利用 Gauss 消元法或 Gauss-Seidel 迭代法求解先行方程组来数值求解该两点边值问题。

2 误差分析

我选用的误差分析方法是均方根误差：

$$\text{RMSE} = \sqrt{\frac{\sum (y_i^* - y_i)^2}{n-1}}$$

Listing 1: RMSE

```
double ComputeRMSE(double numerical_solution[], double exact_solution[], int num_points){
    double sum_squared_error = 0.0;

    for (int i = 0; i < num_points; ++i) {
        double error = numerical_solution[i] - exact_solution[i];
        sum_squared_error += error * error;
    }

    double rmse = sqrt(sum_squared_error / num_points);

    return rmse;
}
```

3 C 语言实现

3.1 Gauss-Seidel 迭代法

3.1.1 代码

Listing 2: Gauss Seidel Iteration

```c
void Gauss_Seidel(double S_Matrix[][Demention], double B[], double initial_X[],
    double Max_error, double Result_X[]){
    double Difference_X[Demention], temp;

    for (int index = 0; index < Demention; index++){
        Result_X[index] = initial_X[index];
    }

    do{
        for (int index_row = 0; index_row < Demention; index_row++){
            temp = B[index_row];
            for (int index_col = 0; index_col < Demention; index_col++){
                if (index_row != index_col){
                    temp -= S_Matrix[index_row][index_col] * Result_X[index_col];
                }
            }
            Result_X[index_row] = temp / S_Matrix[index_row][index_row];
        }

        Vector_Minus(Result_X, initial_X, Difference_X);

        for (int index = 0; index < Demention; index++){
            initial_X[index] = Result_X[index];
        }
    } while (Infinite_Norm(Difference_X) > Max_error);
}
```

3.1.2 稳定性分析

该算法在 $\varepsilon = 1$ 时误差较小；当 $\varepsilon = 0.1$ 时，算法的误差开始变大，计算所得 y_i 与 y_i^* 的误差随在 i 较小时较大，i 越接近 $n-1$ 越小；随着 ε 继续减小，误差逐渐增大。

3.2 Gauss 消元法

3.2.1 代码

Listing 3: Gauss Elimination

```
void Gauss_elimination(double S_Matrix[][Demention], double solution[], double B
    []){
    int index_row, index_col, index_temp;

    for (index_col = 0; index_col < Demention; index_col++){
        double Max_col = fabs(S_Matrix[index_col][index_col]);
        index_temp = index_col;

        for (index_row = index_col; index_row < Demention; index_row++){
            if (fabs(S_Matrix[index_row][index_col]) > Max_col){
                Max_col = fabs(S_Matrix[index_row][index_col]);
                index_temp = index_row;
            }
        }

        if (index_temp != index_col) Swap_Matrix(S_Matrix, B, index_col, index_temp
            );

        for (index_row = index_col + 1; index_row < Demention; index_row++){
            double temp = - (S_Matrix[index_row][index_col]);
            for (int col = index_col; col < Demention; col++){
                double scale = (S_Matrix[index_col][col] / S_Matrix[index_col][index
                    _col]);
                S_Matrix[index_row][col] += scale * temp;
            }
            B[index_row] += B[index_col]/S_Matrix[index_col][index_col] * temp;
        }
    }

    for (index_row = Demention - 1; index_row >= 0; index_row--){
        if (index_row == Demention - 1){
            solution[index_row] = B[index_row] / S_Matrix[index_row][index_row];
        }else{
            double solution_temp = B[index_row];
            for (int col = index_row + 1; col < Demention; col++){
                solution_temp -= S_Matrix[index_row][col] * solution[col];
            }
            solution[index_row] = solution_temp / S_Matrix[index_row][index_row];
        }
    }
}
```

3.2.2 稳定性分析

该算法在 $\varepsilon = 1$ 时误差较小；当 $\varepsilon = 0.1$ 时，算法的误差开始变大，计算所得 y_i 与 y_i^* 的误差随在 i 较小时较大，i 越接近 $n-1$ 越小；随着 ε 继续减小，误差逐渐增大。

4 输出结果

4.1 精确结果

```
epsilon = 1.0
0.01287  0.02566  0.03838  0.05102  0.06358  0.07606  0.08848  0.10081  0.11308
0.12527  0.13739  0.14944  0.16143  0.17334  0.18518  0.19695  0.20866  0.22030
0.23187  0.24338  0.25483  0.26621  0.27752  0.28877  0.29997  0.31110  0.32216
0.33317  0.34412  0.35501  0.36584  0.37661  0.38733  0.39799  0.40859  0.41913
0.42963  0.44006  0.45044  0.46077  0.47105  0.48127  0.49144  0.50156  0.51163
0.52165  0.53162  0.54154  0.55141  0.56123  0.57100  0.58073  0.59041  0.60004
0.60963  0.61917  0.62866  0.63812  0.64752  0.65688  0.66620  0.67548  0.68471
0.69391  0.70306  0.71216  0.72123  0.73026  0.73925  0.74820  0.75710  0.76597
0.77480  0.78360  0.79235  0.80107  0.80975  0.81839  0.82700  0.83557  0.84411
0.85261  0.86108  0.86951  0.87791  0.88627  0.89460  0.90290  0.91116  0.91940
0.92760  0.93576  0.94390  0.95201  0.96008  0.96812  0.97614  0.98412  0.99208

epsilon = 0.1
0.05258  0.10064  0.14460  0.18485  0.22174  0.25560  0.28672  0.31535  0.34173
0.36607  0.38858  0.40942  0.42875  0.44672  0.46345  0.47907  0.49368  0.50737
0.52023  0.53235  0.54379  0.55462  0.56489  0.57466  0.58398  0.59288  0.60142
0.60962  0.61751  0.62513  0.63250  0.63964  0.64658  0.65334  0.65992  0.66636
0.67266  0.67884  0.68490  0.69086  0.69674  0.70252  0.70824  0.71388  0.71947
0.72500  0.73047  0.73591  0.74130  0.74665  0.75197  0.75726  0.76253  0.76776
0.77298  0.77817  0.78335  0.78851  0.79365  0.79878  0.80390  0.80901  0.81410
0.81919  0.82427  0.82934  0.83441  0.83947  0.84452  0.84957  0.85461  0.85965
0.86468  0.86972  0.87475  0.87977  0.88480  0.88982  0.89484  0.89985  0.90487
0.90989  0.91490  0.91991  0.92492  0.92993  0.93494  0.93995  0.94495  0.94996
0.95497  0.95997  0.96498  0.96998  0.97499  0.97999  0.98499  0.98999  0.99500

epsilon = 0.01
0.32106  0.44233  0.49011  0.51084  0.52163  0.52876  0.53454  0.53983  0.54494
0.54998  0.55499  0.56000  0.56500  0.57000  0.57500  0.58000  0.58500  0.59000
0.59500  0.60000  0.60500  0.61000  0.61500  0.62000  0.62500  0.63000  0.63500
0.64000  0.64500  0.65000  0.65500  0.66000  0.66500  0.67000  0.67500  0.68000
0.68500  0.69000  0.69500  0.70000  0.70500  0.71000  0.71500  0.72000  0.72500
0.73000  0.73500  0.74000  0.74500  0.75000  0.75500  0.76000  0.76500  0.77000
0.77500  0.78000  0.78500  0.79000  0.79500  0.80000  0.80500  0.81000  0.81500
```

```
0.82000   0.82500   0.83000   0.83500   0.84000   0.84500   0.85000   0.85500   0.86000
0.86500   0.87000   0.87500   0.88000   0.88500   0.89000   0.89500   0.90000   0.90500
0.91000   0.91500   0.92000   0.92500   0.93000   0.93500   0.94000   0.94500   0.95000
0.95500   0.96000   0.96500   0.97000   0.97500   0.98000   0.98500   0.99000   0.99500

epsilon = 0.0001
0.50500   0.51000   0.51500   0.52000   0.52500   0.53000   0.53500   0.54000   0.54500
0.55000   0.55500   0.56000   0.56500   0.57000   0.57500   0.58000   0.58500   0.59000
0.59500   0.60000   0.60500   0.61000   0.61500   0.62000   0.62500   0.63000   0.63500
0.64000   0.64500   0.65000   0.65500   0.66000   0.66500   0.67000   0.67500   0.68000
0.68500   0.69000   0.69500   0.70000   0.70500   0.71000   0.71500   0.72000   0.72500
0.73000   0.73500   0.74000   0.74500   0.75000   0.75500   0.76000   0.76500   0.77000
0.77500   0.78000   0.78500   0.79000   0.79500   0.80000   0.80500   0.81000   0.81500
0.82000   0.82500   0.83000   0.83500   0.84000   0.84500   0.85000   0.85500   0.86000
0.86500   0.87000   0.87500   0.88000   0.88500   0.89000   0.89500   0.90000   0.90500
0.91000   0.91500   0.92000   0.92500   0.93000   0.93500   0.94000   0.94500   0.95000
0.95500   0.96000   0.96500   0.97000   0.97500   0.98000   0.98500   0.99000   0.99500
```

4.2 Gauss-Seidel 迭代法

```
epsilon = 1.0
0.013   0.026   0.038   0.051   0.064   0.076   0.088   0.101   0.113
0.125   0.137   0.149   0.161   0.173   0.185   0.197   0.208   0.220
0.232   0.243   0.255   0.266   0.277   0.289   0.300   0.311   0.322
0.333   0.344   0.355   0.366   0.376   0.387   0.398   0.408   0.419
0.429   0.440   0.450   0.460   0.471   0.481   0.491   0.501   0.511
0.521   0.531   0.541   0.551   0.561   0.571   0.580   0.590   0.600
0.609   0.619   0.628   0.638   0.647   0.657   0.666   0.675   0.684
0.694   0.703   0.712   0.721   0.730   0.739   0.748   0.757   0.766
0.775   0.783   0.792   0.801   0.810   0.818   0.827   0.835   0.844
0.852   0.861   0.869   0.878   0.886   0.894   0.903   0.911   0.919
0.928   0.936   0.944   0.952   0.960   0.968   0.976   0.984   0.992
Iteration error: 0.00022

epsilon = 0.1
0.014   0.027   0.039   0.050   0.061   0.071   0.081   0.089   0.098
0.105   0.113   0.119   0.126   0.132   0.137   0.142   0.147   0.151
0.156   0.159   0.163   0.166   0.169   0.172   0.175   0.178   0.180
```

```
0.182  0.184  0.186  0.188  0.190  0.191  0.193  0.194  0.195
0.197  0.198  0.199  0.200  0.201  0.202  0.203  0.204  0.205
0.206  0.207  0.208  0.209  0.209  0.210  0.211  0.212  0.213
0.213  0.214  0.215  0.216  0.216  0.217  0.218  0.218  0.219
0.220  0.221  0.221  0.222  0.223  0.223  0.224  0.225  0.225
0.226  0.227  0.228  0.228  0.229  0.230  0.230  0.231  0.232
0.232  0.233  0.234  0.235  0.235  0.236  0.237  0.237  0.238
0.239  0.239  0.240  0.241  0.242  0.261  0.448  0.636  0.827
Iteration error: 0.526

epsilon = 0.01
-48.515  -72.770  -84.893  -90.946  -93.961  -95.451  -96.175  -96.509  -96.642
-96.668  -96.632  -96.558  -96.456  -96.331  -96.186  -96.020  -95.835  -95.630
-95.404  -95.156  -94.887  -94.596  -94.283  -93.946  -93.586  -93.202  -92.794
-92.362  -91.906  -91.425  -90.919  -90.388  -89.832  -89.251  -88.644  -88.012
-87.355  -86.673  -85.965  -85.231  -84.473  -83.688  -82.879  -82.044  -81.185
-80.300  -79.390  -78.455  -77.495  -76.510  -75.501  -74.467  -73.408  -72.325
-71.218  -70.087  -68.931  -67.751  -66.548  -65.321  -64.070  -62.795  -61.497
-60.176  -58.832  -57.464  -56.073  -54.660  -53.223  -51.764  -50.283  -48.779
-47.252  -45.703  -44.133  -42.540  -40.925  -39.288  -37.629  -35.949  -34.247
-32.524  -30.780  -29.014  -27.227  -25.419  -23.590  -21.740  -19.869  -17.978
-16.066  -14.133  -12.180  -10.207  -8.214  -6.200  -4.166  -2.113-0.039
Iteration error: 72.490

epsilon = 0.0001
-191.06  -192.95  -192.96  -192.94  -192.90  -192.85  -192.79  -192.70  -192.59
-192.45  -192.28  -192.09  -191.86  -191.60  -191.30  -190.97  -190.60  -190.19
-189.73  -189.24  -188.70  -188.12  -187.49  -186.82  -186.10  -185.33  -184.52
-183.65  -182.74  -181.78  -180.77  -179.71  -178.60  -177.43  -176.22  -174.96
-173.64  -172.28  -170.86  -169.40  -167.88  -166.31  -164.70  -163.03  -161.31
-159.54  -157.72  -155.85  -153.93  -151.96  -149.94  -147.88  -145.76  -143.59
-141.38  -139.12  -136.81  -134.45  -132.04  -129.59  -127.09  -124.54  -121.94
-119.30  -116.61  -113.88  -111.10  -108.27  -105.40  -102.48  -99.521  -96.513
-93.461  -90.364  -87.223  -84.037  -80.808  -77.534  -74.218  -70.858  -67.455
-64.009  -60.520  -56.989  -53.415  -49.800  -46.142  -42.442  -38.701  -34.919
-31.096  -27.231  -23.326  -19.380  -15.393  -11.366  -7.299   -3.193   0.954
Iteration error: 145.540
```

4.3 Gauss 消元法

```
epsilon = 1.0
0.013  0.026  0.038  0.051  0.064  0.076  0.088  0.101  0.113
0.125  0.137  0.149  0.161  0.173  0.185  0.197  0.208  0.220
0.232  0.243  0.255  0.266  0.277  0.289  0.300  0.311  0.322
0.333  0.344  0.355  0.366  0.376  0.387  0.398  0.408  0.419
0.429  0.440  0.450  0.460  0.471  0.481  0.491  0.501  0.511
0.521  0.531  0.541  0.551  0.561  0.571  0.580  0.590  0.600
0.609  0.619  0.628  0.638  0.647  0.657  0.666  0.675  0.684
0.694  0.703  0.712  0.721  0.730  0.739  0.748  0.757  0.766
0.775  0.783  0.792  0.801  0.810  0.818  0.827  0.835  0.844
0.852  0.861  0.869  0.878  0.886  0.894  0.903  0.911  0.919
0.928  0.936  0.944  0.952  0.960  0.968  0.976  0.984  0.992
Iteration error: 0.00022
```

```
epsilon = 0.1
-0.758  -1.446  -2.071  -2.639  -3.153  -3.620  -4.042  -4.425  -4.770
-5.082  -5.363  -5.616  -5.843  -6.046  -6.228  -6.389  -6.532  -6.658
-6.768  -6.864  -6.947  -7.017  -7.076  -7.124  -7.162  -7.191  -7.212
-7.225  -7.230  -7.228  -7.220  -7.205  -7.185  -7.159  -7.128  -7.092
-7.052  -7.007  -6.958  -6.905  -6.848  -6.787  -6.722  -6.654  -6.583
-6.509  -6.431  -6.351  -6.267  -6.181  -6.092  -6.000  -5.905  -5.808
-5.708  -5.606  -5.501  -5.394  -5.284  -5.172  -5.058  -4.941  -4.822
-4.701  -4.578  -4.452  -4.325  -4.195  -4.063  -3.928  -3.792  -3.654
-3.513  -3.371  -3.226  -3.080  -2.931  -2.780  -2.628  -2.473  -2.317
-2.158  -1.998  -1.835  -1.671  -1.505  -1.337  -1.166  -0.995  -0.821
-0.645  -0.468  -0.288  -0.107  0.076  0.261  0.448  0.636  0.827
Elimination error: 5.732
```

```
epsilon = 0.01
-48.515  -72.770  -84.893  -90.946  -93.961  -95.451  -96.175  -96.509  -96.642
-96.668  -96.632  -96.558  -96.456  -96.331  -96.186  -96.020  -95.835  -95.630
-95.404  -95.156  -94.887  -94.596  -94.283  -93.946  -93.586  -93.202  -92.794
-92.362  -91.906  -91.425  -90.919  -90.388  -89.832  -89.251  -88.644  -88.012
-87.355  -86.673  -85.965  -85.231  -84.473  -83.688  -82.879  -82.044  -81.185
-80.300  -79.390  -78.455  -77.495  -76.510  -75.501  -74.467  -73.408  -72.325
-71.218  -70.087  -68.931  -67.751  -66.548  -65.321  -64.070  -62.795  -61.497
```

```
-60.176  -58.832  -57.464  -56.073  -54.660  -53.223  -51.764  -50.283  -48.779
-47.252  -45.703  -44.133  -42.540  -40.925  -39.288  -37.629  -35.949  -34.247
-32.524  -30.780  -29.014  -27.227  -25.419  -23.590  -21.740  -19.869  -17.978
-16.066  -14.133  -12.180  -10.207  -8.214   -6.200   -4.166   -2.113    -0.039
Elimination error: 72.490

epsilon = 0.0001
-191.06  -192.95  -192.96  -192.94  -192.90  -192.85  -192.79  -192.70  -192.59
-192.45  -192.28  -192.09  -191.86  -191.60  -191.30  -190.97  -190.60  -190.19
-189.73  -189.24  -188.70  -188.12  -187.49  -186.82  -186.10  -185.33  -184.52
-183.65  -182.74  -181.78  -180.77  -179.71  -178.60  -177.43  -176.22  -174.96
-173.64  -172.28  -170.86  -169.40  -167.88  -166.31  -164.70  -163.03  -161.31
-159.54  -157.72  -155.85  -153.93  -151.96  -149.94  -147.88  -145.76  -143.59
-141.38  -139.12  -136.81  -134.45  -132.04  -129.59  -127.09  -124.54  -121.94
-119.30  -116.61  -113.88  -111.10  -108.27  -105.40  -102.48  -99.521  -96.513
-93.461  -90.364  -87.223  -84.037  -80.808  -77.534  -74.218  -70.858  -67.455
-64.009  -60.520  -56.989  -53.415  -49.800  -46.142  -42.442  -38.701  -34.919
-31.096  -27.231  -23.326  -19.380  -15.393  -11.366  -7.299   -3.193   0.954
Elimination error: 145.540
```

5 两种算法比较

两种算法的误差均随着 ε 的减小而增大，但是 Gauss-Seidel 迭代法的误差随 ε 减小而增大的速度比 Gauss 消元法慢，从上节的输出结果可以看出，Gauss-Seidel 迭代法在 $\varepsilon = 0.1$ 时的误差小于 Gauss 消元法，这说明 Gauss-Seidel 迭代法的稳定性优于 Gauss 消元法。