БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Казачинский Глеб Всеволодович

**Лабораторная работа №1-2**

**Разностные схемы для ОДУ 2-ого порядка**

Студента 3 курса 6б группы

Зачтено \_ марта 2020 г. Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Радкевич Елена Владимировна

(подпись преподавателя)

Минск 2020

***1. Постановка задачи***

**Вариант 4**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(x) | k(x) | q(x) |  |  |  |  |
| sin2(x) | cos2(x) + 1 | 1 | 1 | 0 | 1 | 1 |

Указанными ниже способами построить разностные схемы для решения третьей краевой задачи для ОДУ следующего вида.

, x

k(0)

-*k(1)*

1.Используя разностные операторы вместо дифференциальных, аппроксимировать поставленную задачу разностной схемой 2-го порядка на минимальном шаблоне.

2.Интегро-интерполяционным методом построить консервативную разностную схему. Для вычисления коэффициентов этой схемы использовать формулу средних прямоугольников.  
  
3.Аппроксимировать исходную задачу вариационно-разностным методом. Для вычисления коэффициентов полученной схемы использовать формулу трапеций.

4.Методом разностной прогонки реализовать полученные в п.п. 1-3 разностные схемы при h = 0,1. Провести сравнительный анализ сеточных решений при разных шагах.

***2. Алгоритм решения***

**Общая теория :**

Построим на заданном отрезке [0,1] сетку , с числом разбиений N: h = 0.1 = Итого, = {

Также, для упрощения, раскроем исходное ОДУ:

*Матрицу системы сводим к следующему трехдиагональному виду:*

*+*

*i =*

*+*

*Имеем:*

*В таком случае можно использовать метод прогонки с следующими прогоночными коэффициентами:*

*i =*

*i =*

*В результате получаем следующие формулы для определения приближенных значений, будем определять требуемые коэффициенты в каждом из рассматриваемых случаев:*

**1. Аппроксимация на минимальном шаблоне**

Приведя исходную матрицу к трехдиагональному виду

получаем:



Теперь определим начальные прогоночные коэффициенты из краевых условий:

+

=

+

= =

Код программы

1. **import** numpy **as** np

2.

3.

4. **def** f\_func(x):

5. **return** (np.sin(x)) \*\* 2

6.

7.

8. **def** k\_func(x):

9. **return** (np.cos(x)) \*\* 2 + 1

10.

11.

12. **def** q\_func(x):

13. **return** 1

14.

15.

16. N, h, ae\_0, g\_0, ae\_1, g\_1 = 10, 1 / 10, 1, 0, 1, 1

17. x, k, q, f, C, F, A, B = [], [], [], [], [], [], [], []

18.

19. **for** i **in** range(0, N + 1):

20. x.append(i \* h)

21. k.append(k\_func(x[i]))

22. q.append(q\_func(x[i]))

23. f.append(f\_func(x[i]))

24. C.append(2 \* k[i] + h \* h \* q[i])

25. F.append(f[i] \* h \* h)

26.

27. R = h \* ae\_0 + h \* h \* q[0] / 2 - h \* h \* (-2 \* np.sin(x[0]) \* np.cos(x[0])) \* ae\_0 / (2 \* k[0]) + k[0]

28. O\_1 = k[0] / R

29. W1 = (h \* g\_0 + h \* h \* f[0] / 2 - h \* h \* (-2 \* np.sin(x[0]) \* np.cos(x[0])) \* g\_0 / (2 \* k[0])) / R

30. Q = h \* ae\_1 + h \* h \* q[N] / 2 + h \* h \* (-2 \* np.sin(x[0]) \* np.cos(x[0])) \* ae\_1 / (2 \* k[N]) + k[N]

31. O\_2 = k[N] / Q

32. W2 = (h \* g\_1 + h \* h \* f[N] / 2 + h \* h \* (-2 \* np.sin(x[0]) \* np.cos(x[0])) \* g\_1 / (2 \* k[N])) / Q

33.

34. A.append(0)

35. B.append(O\_1)

36. **for** i **in** range(1, N):

37. A.append(k[i] - (k[i + 1] - k[i - 1]) / 4)

38. B.append(k[i] + (k[i + 1] - k[i - 1]) / 4)

39.

40. A.append(O\_2)

41. B.append(0)

42. C[0], F[0], C[N], F[N] = 1, W1, 1, W2

43.

44. alfa, beta, y = np.zeros(N + 1), np.zeros(N + 1), np.zeros(N + 1)

45. alfa[1], beta[1] = O\_1, W1

46.

47. **for** i **in** range(1, N):

48. alfa[i + 1] = B[i] / (C[i] - alfa[i] \* A[i])

49. beta[i + 1] = (F[i] + beta[i] \* A[i]) / (C[i] - alfa[i] \* A[i])

50.

51. y[N] = (W2 + O\_2 \* beta[N]) / (1 - alfa[N] \* O\_2)

52.

53. **for** i **in** range(N - 1, -1, -1):

54. y[i] = alfa[i + 1] \* y[i + 1] + beta[i + 1]

55.

56. **for** i **in** range(0, N + 1):

57. **print**("u(", x[i], ") = ", y[i])

**2. Интегро-интерполяционный метод**

Для данной краевой задачи консервативная разностная схема, построенная интегро-интерполяционным методом, имеет следующий вид в индексной записи:

*Использованные коэффициенты расшифровываются следующим образом:*

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Приведём исходную матрицу к трехдиагональному виду:

-

Итого, получаем:



Определим начальные прогоночные коэффициенты из краевых условий:

+

* =

+

Код программы

1. **import** numpy **as** np

2.

3.

4. **def** f\_func(x):

5. **return** (np.sin(x)) \*\* 2

6.

7.

8. **def** k\_func(x):

9. **return** 1 / ((np.cos(x)) \*\* 2 + 1)

10.

11.

12. **def** q\_func(x):

13. **return** 1

14.

15.

16. **def** integr(predicate, a, b):

17. count = 5000

18. integral, step = 0, (b - a) / count

19. **for** i **in** range(1, count + 1):

20. integral += step \* predicate(a + (i - 1) \* step)

21. **return** integral

22.

23.

24. N, h, ae\_0, g\_0, ae\_1, g\_1 = 10, 1 / 10, 1, 0, 1, 1

25. x, a, d, fi, C, F, A, B = [], [], [], [], [], [], [], []

26.

27. **for** i **in** range(0, N + 1):

28. x.append(i \* h)

29. a.append(1 / ((1 / h) \* integr(**lambda** x1: k\_func(x1), x[i - 1], x[i])))

30. d.append((1 / h) \* integr(**lambda** x1: q\_func(x1), x[i] - h / 2, x[i] + h / 2))

31. fi.append((1 / h) \* integr(**lambda** x1: f\_func(x1), x[i] - h / 2, x[i] + h / 2))

32. A.append(a[i])

33.

34. d[0] = (2 / h) \* integr(**lambda** x1: q\_func(x1), 0, h / 2)

35. fi[0] = (2 / h) \* integr(**lambda** x1: f\_func(x1), 0, h / 2)

36. d[N] = (2 / h) \* integr(**lambda** x1: q\_func(x1), 1 - h / 2, 1)

37. fi[N] = (2 / h) \* integr(**lambda** x1: f\_func(x1), 1 - h / 2, 1)

38.

39. R = h \* ae\_0 + h \* h \* d[0] / 2 + a[1]

40. O\_1 = a[1] / R

41. W1 = (h \* g\_0 + h \* h \* fi[0] / 2) / R

42. Q = h \* ae\_1 + h \* h \* d[N] / 2 + a[N]

43. O\_2 = a[N] / Q

44. W2 = (h \* g\_1 + h \* h \* fi[N] / 2) / Q

45.

46. C.append(1)

47. B.append(O\_1)

48. F.append(W1)

49. **for** i **in** range(1, N):

50. C.append(a[i + 1] + a[i] + d[i] \* h \* h)

51. B.append(a[i + 1])

52. F.append(fi[i] \* h \* h)

53.

54. A[0] = 0

55. A[N] = O\_2

56. C.append(1)

57. B.append(0)

58. F.append(W2)

59.

60. alfa, beta, y = np.zeros(N + 1), np.zeros(N + 1), np.zeros(N + 1)

61. alfa[1], beta[1] = O\_1, W1

62.

63. **for** i **in** range(1, N):

64. alfa[i + 1] = B[i] / (C[i] - alfa[i] \* A[i])

65. beta[i + 1] = (F[i] + beta[i] \* A[i]) / (C[i] - alfa[i] \* A[i])

66.

67. y[N] = (W2 + O\_2 \* beta[N]) / (1 - alfa[N] \* O\_2)

68.

69. **for** i **in** range(N - 1, -1, -1):

70. y[i] = alfa[i + 1] \* y[i + 1] + beta[i + 1]

71.

72. **for** i **in** range(0, N + 1):

73. **print**("u(", x[i], ") = ", y[i])

74.

3. Вариационно-разностный метод

Если использовать метод Ритца для построения разностной схемы вариационно-разностным методом, то благодаря специальным обозначениям для данной краевой задачи можно получить разностную схему, что будет аналогична построенной в предыдущем пункте:

*Однако, использованные коэффициенты будут расшифровываться иным образом:*

Поскольку коэффициенты те же, лишь подсчитываются иначе, можно воспользоваться результатами, полученными в предыдущем пункте.

# ***3. Листинг программы***

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

double fF(double x) {

return 2 \* cos(x) - sin(x);

}

double kF(double x) {

return 2 - x;

}

double qF(double x) {

return x;

}

int main()

{

system("chcp 1251");

system("cls");

int i, j, N = 10;

double h = pow(N, -1), l = h \* h;

double \*x = new double[N], \*k = new double[N], \*q = new double[N], \*f = new double[N], \*y = new double[N];

double \*F = new double[N], \*A = new double[N], \*B = new double[N], \*C = new double[N];

double X0 = 1, q0 = 1, X1 = tan(1), q1 = 0;

for (i = 0; i <= N; i++)

{

x[i] = i \* h;

k[i] = kF(x[i]);

q[i] = qF(x[i]);

f[i] = fF(x[i]);

C[i] = 2 \* k[i] + l \* q[i];

F[i] = f[i] \* l;

}

for (i = 0; i <= N; i++)

{

A[i] = k[i] - (k[i + 1] - k[i - 1])\*pow(4, -1);

B[i] = k[i] + (k[i + 1] - k[i - 1])\*pow(4, -1);

}

double R = h \* X0 + l \* q[0] / 2 - l \* (-1)\*X0 / (2 \* k[0]) + k[0];

double O1 = k[0] / R, W1 = (h \* q0 + l \* f[0] / 2 - l \* (-1)\*q0 / (2 \* k[0])) / R;

double Q = h \* X1 + l \* q[N] / 2 + l \* (-1) \*X1 / (2 \* k[N]) + k[N];

double O2 = k[N] / Q, W2 = (h \*q1 + l \* f[N] / 2 + l \* (-1)\*q1 / (2 \* k[N])) / Q;

A[0] = 0;

C[0] = 1;

B[0] = O1;

F[0] = W1;

A[N] = O2;

C[N] = 1;

B[N] = 0;

F[N] = W2;

double \* alfa = new double[N], \*beta = new double[N];

alfa[1] = O1; beta[1] = W1;

for (i = 1; i <= N - 1; i++) {

alfa[i + 1] = B[i] / (C[i] - alfa[i] \* A[i]);

beta[i + 1] = (F[i] + beta[i] \* A[i]) / (C[i] - alfa[i] \* A[i]);

}

y[N] = (W2 + O2 \* beta[N]) / (1 - alfa[N] \* O2);

for (i = N - 1; i >= 0; i--) {

y[i] = alfa[i + 1] \* y[i + 1] + beta[i + 1];

}

for (i = 0; i <= N; i++)

{

cout << "u(" << x[i] << ") = " << y[i] << endl;

}

system("pause");

return 0;

}

***4. Вывод программы***

Значения решения в точках сетки:

|  |  |  |  |
| --- | --- | --- | --- |
| Результат | Минимальный шаблон | Интегро-интерполяционный метод | Вариационно-разностный метод |
| u(0) | 0.999882 | 1.00023 |  |
| u(0.1) | 0.994876 | 0.995238 |  |
| u(0.2) | 0.97992 | 0.980304 |  |
| u(0.3) | 0.955164 | 0.955572 |  |
| u(0.4) | 0.920855 | 0.921291 |  |
| u(0.5) | 0.877337 | 0.877802 |  |
| u(0.6) | 0.825045 | 0.825543 |  |
| u(0.7) | 0.764501 | 0.765034 |  |
| u(0.8) | 0.696311 | 0.696882 |  |
| u(0.9) | 0.621157 | 0.621767 |  |
| u(1) | 0.53979 | 0.54044 |  |

#include "stdafx.h"

#include <conio.h>

#include <iostream>

#include <math.h>

using namespace std;

double fF(double x) {

return 2 \* cos(x) - sin(x);

}

double kF1(double x) {

return pow(2 - x, -1);

}

double qF(double x) {

return x;

}

double sred(double(\*Function)(double), double a, double b)

{

int count = 5000;

double integral = 0, step = (b - a) / count;

for (int i = 1; i <= count; ++i) {

integral += step \* Function(a + (i - 1) \* step);

}

return integral;

}

int main()

{

int i, j, N = 10;

double h = pow(N, -1), l = h \* h, t = pow(h, -1), u = 0.5\*h;

double X0 = 1, q0 = 1, X1 = tan(1), q1 = 0;

double \*x = new double[N], \*d = new double[N], \*fi = new double[N], \*a = new double[N], \*y = new double[N];

double \*A = new double[N], \*C = new double[N], \*B = new double[N], \*F = new double[N];

for (i = 0; i <= N; i++)

{

x[i] = i \* h;

a[i] = pow(t\*sred(&kF1, x[i - 1], x[i]), -1);

d[i] = t \* sred(&qF, x[i] - u, x[i] + u);

fi[i] = t \* sred(&fF, x[i] - u, x[i] + u);

A[i] = a[i];

}

d[0] = 2 \* t\*sred(&qF, 0, u);

fi[0] = 2 \* t\*sred(&fF, 0, u);

d[N] = 2 \* t\* sred(&qF, 1 - u, 1);

fi[N] = 2 \* t\*sred(&fF, 1 - u, 1);

for (i = 0; i <= N; i++)

{

C[i] = a[i + 1] + a[i] + d[i] \* l;

B[i] = a[i + 1];

F[i] = fi[i] \* l;;

}

double R = h \* X0 + l \* d[0] / 2 + a[1];

double O1 = a[1] / R, W1 = (h \* q0 + l \*fi[0] / 2) / R;

double Q = h \* X1 + l \* d[N] / 2 + a[N];

double O2 = a[N] / Q, W2 = (h \* q1 + l \* fi[N] / 2) / Q;

A[0] = 0;

C[0] = 1;

B[0] = O1;

F[0] = W1;

A[N] = O2;

C[N] = 1;

B[N] = 0;

F[N] = W2;

double \* alfa = new double[N], \*beta = new double[N];

alfa[1] = O1; beta[1] = W1;

for (i = 1; i <= N - 1; i++) {

alfa[i + 1] = B[i] / (C[i] - alfa[i] \* A[i]);

beta[i + 1] = (F[i] + beta[i] \* A[i]) / (C[i] - alfa[i] \* A[i]);

}

y[N] = (W2 + O2 \* beta[N]) / (1 - alfa[N] \* O2);

for (i = N - 1; i >= 0; i--) {

y[i] = alfa[i + 1] \* y[i + 1] + beta[i + 1];

}

for (i = 0; i <= N; i++)

{

cout << "u(" << x[i] << ") = " << y[i] << endl;

}

system("pause");

return 0;

}

***5. Выводы***

Погрешность вычислений значений функции с помощью естественного кубического сплайна достаточно низкая, к тому же данный тип сплайна можно легко применять на практике, используя метод прогонки.

Третий случай:  
  
#include "stdafx.h"

#include<iostream>

#include <cmath>

#include <string>

using namespace std;

double h = 0.1, start=0, end=1;

double f(double x) {

return 2\*cos(x)-sin(x);

}

double k(double x) {

return 2-x;

}

double q(double x) {

return x;

}

double d0(double x) {

return q(x)\*(h-x);

}

double dN(double x) {

return q(x) \* (x-1+h);

}

double fi0(double x) {

return f(x)\*(h - x);

}

double fiN(double x) {

return f(x) \* (x - 1 + h);

}

double trap(double (\*foo)(double x),double intervalBegin, double intervalEnd) {

int count = 500;

double integral = 0.5 \* (foo(intervalBegin) + foo(intervalEnd));

double step = (intervalEnd - intervalBegin) / count;

for (int i = 0; i < count; ++i) {

integral += foo(intervalBegin + step \* i);

}

integral \*= step;

cout << "Integral is equal to: " << integral << endl;

return integral;

}

void main() {

int i, j,N=11;

double h = pow(N - 1,-1);

double \*x = new double[N], \*a = new double[N], \*d = new double[N], \*fi = new double[N];

double X0 = 1, q0 = 1, X1 = tan(1), q1 = 0;

for (i = 0; i < N; i++)

{

x[i] = i \* h;

d[i] = 0;

fi[i] = 0;

a[i] = 0;

}

double r = 2 / (h\*h);

d[0] = trap(d0, 0, h)\*r;

d[N-1]=trap(dN, 1-h, 1)\*r;

fi[0] = trap(fi0, 0, h)\*r;

fi[N - 1] = trap(fiN, 1 - h, 1)\*r;

system("pause");

}