БЕЛАРУСКІ ДЗЯРЖАЎНЫ УНІВЕРСІТЭТ

Факультэт прыкладной матэматыкі і інфарматыкі

Метады лікавага аналізу

Справаздача па лабараторнай працы
па тэме

Пошук набліжанага рашэння задачы Кашы для АДУ-1

Богдана Уладзіслава
3 курс, 3 група

Выкладчык:
Буднік А. М.

Мінск, 2016

Перад намі ставіцца задача набліжанымі метадамі на адрэзку $[x_0, x_1]$ рашыць задачу Кашы для АДУ-1:

$$\begin{cases} u' = f(x, u), \quad x \in [x_0, x_1], \\ y(x_0) = u_0. \end{cases} \tag{1}$$

і задачу Кашы для сістэмы АДУ-1:

$$\begin{cases} u' = f(x, u, v), \\ v' = g(x, u, v), \quad x \in [x_0, x_1], \\ u(x_0) = u_0, \quad v(x_0) = v_0. \end{cases} \tag{2}$$

Вынікам працы кожнага алгарытма будзе разбіццё адрэзка $[x_0, x_1]$ на N частак і падлік у адпаведных кропках набліжаных значэнняў функцыі $u(x)$ альбо сістэмы функцый $u(x), v(x)$.

Да ўраўнення прыменім метад шэрагаў трэцяга парадка дакладнасці, яўны і няяўны метады Эйлера, метад паслядоўнага павышэння парадка дакладнасці (прэдыктар-карэктар) трэцяга парадка дакладнасці, метад Рунге-Кутта другога парадка дакладнасці, экстрапаляцыйны метад Адамса другога парадка.

Да сістэмы ўраўненняў будуць прыменены яўны і няяўны метады Эйлера, метад паслядоўнага павышэння парадка дакладнасці другога парадка, метад Рунге-Кутта другога парадка дакладнасці, інтэрпаляцыйны метад Адамса трэцяга парадка дакладнасці.

Умовы канкрэтнай задачы

Ураўненне:

$$\begin{cases} u' = -ycos(x) + sin(x)cos(x), \quad x \in [0, 1], \\ y(0) = -1 \end{cases} \tag{3}$$

Сістэма ўраўненняў:

$$\begin{cases} y' = e^{-(y^2+z^2)} + 2x, \\ z' = 2y^2 + z, \quad x \in [0, 0.5], \\ y(0) = 0.5, \\ z(0) = 1. \end{cases} \tag{4}$$

Вызначэнне ў кодзе

Ураўненне:

```
# Defining the Cauchy problem for equation.
f = lambda x, u: - u * math.cos(x) + math.sin(x) * math.cos(x)
x_range = [0., 1.]
x0 = 0.
u0 = -1.

du_dx = f

# Defining a grid to make calculations on.
n = 10
h = (x_range[1] - x_range[0]) / n
grid = [x_range[0] + i * h for i in range(n+1)]

# Exact solution of the problem.
exact_solution = lambda x: -1 + math.sin(x)

# Some pre-calculations. k is for highest degree of derivations.
k = 3
d_dx = [
    lambda x, u: u0,
    lambda x, u: du_dx(x, u),
    lambda x, u: u * math.sin(x) + math.cos(2 * x) + f(x, u) * (- math.cos(x)),
    lambda x, u: u * math.cos(x) - 2 * math.sin(2 * x) + 2 * math.sin(x) * f(x, u) -
            math.cos(x) * d_dx[2](x, u),
]
```

Сістэма ўраўненняў:

```
# Defining the problem (system of differential equations).
dy_dx = lambda x, y, z: math.e**(-(y*y + z*z)) + 2 * x
dz_dx = lambda x, y, z: 2 * y * y + z
x_range = [0., 0.5]
x0 = 0
y0 = 0.5
z0 = 1.

f = dy_dx
g = dz_dx

# Defining the grid to make the calculations.
n = 10
h = (x_range[1] - x_range[0]) / n
grid = []
for i in range(0, n):
    grid.append(x_range[0] + i * h)
```

1 Метад Шэрагаў

Апраксімацыя АДУ-1 найпрасцейшым чынам - праз расклад у шэраг Тэйлара:

$$u(x) \approx \sum_{i=0}^{k} \frac{u^{(i)}(\xi)}{i!}(x-\xi)^i, \tag{5}$$

Апраксімаваць будзем у кропцы x_0, $k = 3$. Справядлівая наступная формула:

$$u(x) \approx \sum_{i=0}^{3} \frac{u^{(i)}(x_0)}{i!}(x-x_0)^i, \tag{6}$$

Для $i = 0, 1$ значэнні вядомыя; вытворныя вышэйшых парадкаў атрымаем, дыферэнцуючы функцыю $f(x, u)$ па x. Апраксімацыя на раўнамернай сетцы $x_j \in [x_0, x_1]$:

$$u(x) \approx u_j^k = \sum_{i=1}^{k} \frac{u_j^{(i)}(x_j)}{i!}(x-x_j)^i, \tag{7}$$

За $u_j^{(0)}$ прымаецца папярэдне знойдзенае значэнне u_{j-1}. Праз формулы (5) - (7) атрымліваем наступныя вынікі.

```
def series():
    yi = [u0]
    for i in range(1, len(grid)):
        x = grid[i-1]
        step = grid[i] - grid[i-1]
        y = yi[i-1]
        for j in range(1, k+1):
            y += d_dx[j](x, yi[i-1]) / math.factorial(j) * (step**j)
        yi.append(y)
    return yi
```

Вынік працы алгарытма:

```
Series method:
y(0.0) = -1.0
y(0.1) = -0.900166666667
y(0.2) = -0.80133124341
y(0.3) = -0.704481223709
y(0.4) = -0.610584270389
y(0.5) = -0.520578547982
y(0.6) = -0.435363349425
y(0.7) = -0.355790110828
y(0.8) = -0.282653904223
y(0.9) = -0.216685493369
y(1.0) = -0.158544032078
```

Маем сетку значэнняў, па якой можна пабудаваць набліжанае рашэнне ураўнення (3).

2 Яўны метад Эйлера

Ураўненне

$$y_{n+1} = y_n + hf_n, \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}. \tag{8}$$

```
def euler_explicit():
    yi = [u0]
    for i in range(1, len(grid)):
        x = grid[i-1]
        step = grid[i] - grid[i-1]
        yi.append( yi[-1] + step*f(x,yi[-1]) )
    return yi
```

```
Explicit Euler:
y(0.0) = -1.0
y(0.1) = -0.9
y(0.2) = -0.800516158585
y(0.3) = -0.702589328265
y(0.4) = -0.607236282379
y(0.5) = -0.51543831245
y(0.6) = -0.428130795736
y(0.7) = -0.346193682081
y(0.8) = -0.270442842279
y(0.9) = -0.201622227856
y(1.0) = -0.140396807646
```

Сістэма

$$\begin{cases} y_{n+1} = y_n + hf_n, \\ z_{n+1} = z_n + hg_n, \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}. \end{cases} \tag{9}$$

```
def euler_explicit_system():
    yi = [y0]
    zi = [z0]
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]
        yi.append(yi[i-1] + step * f(grid[i-1], yi[i-1], zi[i-1]))
        zi.append(zi[i-1] + step * g(grid[i-1], yi[i-1], zi[i-1]))
    return (yi, zi)
```

```
Explicit Euler for system:
Y:
y(0.0) = 0.5
y(0.05) = 0.514325239843
y(0.1) = 0.531409062083
y(0.15) = 0.551334821493
y(0.2) = 0.574239609375
y(0.25) = 0.600311215312
y(0.3) = 0.629780591884
y(0.35) = 0.662909402276
y(0.4) = 0.699973080276
y(0.45) = 0.741241007894
y(0.5) = 0.786956675174
Z:
z(0.0) = 1.0
z(0.05) = 1.075
z(0.1) = 1.15520304523
z(0.15) = 1.24120275662
z(0.2) = 1.33365990299
z(0.25) = 1.43331801104
z(0.3) = 1.54102126711
z(0.35) = 1.65773468986
z(0.4) = 1.78456631192
z(0.45) = 1.92279085882
z(0.5) = 2.07387422494
```

3 Няяўны метад Эйлера

Ураўненне

$$y_{n+1} = y_n + hf_{n+1}, \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1} \tag{10}$$

Для падліку па формуле (10) выкарыстоўваўся метад Ньютана рашэння нелінейных ураўненняў з дакладнасцю $\epsilon = 10^{-5}$; ітэрацыйны працэс здзяйсняўся па формуле:

$$y_{n+1}^{k+1} = y_n^k - \frac{y_n^k - y_n^0 - hf(x_{n+1}, y_{n+1}^k)}{1 - h\frac{df}{dy}(x_{n+1}, y_{n+1}^k)} \tag{11}$$

```
def euler_implicit():
    yi = [u0]
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]
        yi.append(newton_iterations(step, grid[i], yi[i-1]))
    return yi

def newton_iterations(step, x, y0):
    y_prev = y0
    iter = MAX_NEWTON_ITERATIONS
    df_dy = lambda x, u: - math.cos(x)
    while iter > 0:
        y_new = y_prev - (y_prev - y0 - step * f(x, y_prev)) / (1. - step * df_dy(x, y_prev))
        iter -= 1
        if abs(y_prev - y_new) < EPS:
            return y_new
        y_prev = y_new
    print "newton iterations: no convergence"
    return
```

```
Implicit Euler:
y(0.0) = -1
y(0.1) = -0.900469448285
y(0.2) = -0.802361738815
y(0.3) = -0.706623311765
y(0.4) = -0.614185295356
y(0.5) = -0.525954864383
y(0.6) = -0.442806511629
y(0.7) = -0.365573427165
y(0.8) = -0.295039170081
y(0.9) = -0.231929800918
y(1.0) = -0.176906623896
```

Сістэма

$$\begin{cases} y_{n+1} = y_n + hf_{n+1}, \\ z_{n+1} = z_n + hg_{n+1}, \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}. \end{cases} \tag{12}$$

Для сістэмы ураўненняў будзем выкарыстоўваць метад простай ітэрацыі для падліку па формулах (12):

$$\begin{pmatrix} y \\ z \end{pmatrix}_{n+1}^{k+1} = \begin{pmatrix} y \\ z \end{pmatrix}_n^0 + h\begin{pmatrix} f(x_{n+1}, y_{n+1}^k, z_{n+1}^k) \\ g(x_{n+1}, y_{n+1}^k, z_{n+1}^k) \end{pmatrix} \tag{13}$$

```
def euler_implicit_system():
    yi = [y0]
    zi = [z0]
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]
        result = simple_iterations(yi[i-1], zi[i-1], step, f, g, grid[i], yi[i-1], zi[i-1])
        yi.append(result[0])
        zi.append(result[1])
    return (yi, zi)
```

```
def simple_iterations(base_y, base_z, step, f, g, x, y0, z0):
    y_prev = y0
    z_prev = z0
    iter = MAX_ITERATIONS
    while iter > 0:
        y_new = base_y + step * f(x, y_prev, z_prev)
        z_new = base_z + step * g(x, y_prev, z_prev)
        iter -= 1
        if abs(y_prev - y_new) < EPS and abs(z_prev - z_new) < EPS:
            return (y_prev, z_prev)
        y_prev = y_new
        z_prev = z_new
    print "simple iterations: no convergence"
    return
```

```
Implicit Euler for system:
Y:
y(0.0) = 0.5
y(0.05) = 0.516902593331
y(0.1) = 0.536487074459
y(0.15) = 0.558923336938
y(0.2) = 0.584443312167
y(0.25) = 0.613329740347
y(0.3) = 0.64589751371
y(0.35) = 0.682466528395
y(0.4) = 0.723340807541
y(0.45) = 0.768775793109
y(0.5) = 0.818964285088
Z:
z(0.0) = 1.0
z(0.05) = 1.08076711989
z(0.1) = 1.167955543
z(0.15) = 1.26231766847
z(0.2) = 1.36471426924
z(0.25) = 1.47613753186
z(0.3) = 1.59773611648
z(0.35) = 1.73085453666
z(0.4) = 1.87702730938
z(0.45) = 2.03802914727
z(0.5) = 2.21589231809
```

4 Метад паслядоўнага павышэння парадка дакладнасці

Ураўненне

Для ўраўнення мы будзем выкарыстоўваць метад паслядоўнага павышэння парадка дакладнасці 3-га парадка дакладнасці ($O(h^4)$). Для набліжэння значэнняў інтэгралаў будзем выкарыстоўваць формулу трапецыяў. Атрымліваем наступныя формулы:

$$y^{[4]}_{n+1} = y^{[4]}_n + \frac{h}{6}(f^{[4]}_n + 4f^{[3]}_{n+\frac{1}{2}} + f^{[3]}_{n+1})$$

$$y^{[3]}_{n+1} = y^{[4]}_n + \frac{h}{2}(f^{[4]}_n + f^{[2]}_{n+1})$$

$$y^{[2]}_{n+1} = y^{[4]}_n + hf^{[4]}_n$$

$$y^{[3]}_{n+\frac{1}{2}} = y^{[4]}_n + \frac{h}{4}(f^{[4]}_n + f^{[2]}_{n+\frac{1}{2}})$$

$$y^{[2]}_{n+\frac{1}{2}} = y^{[4]}_n + \frac{h}{2}f^{[4]}_n, \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}.$$

```
def predictor_corrector():
    yi4 = [u0]
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]
```

```
        y_n4_2 = yi4[i-1] + step * f(grid[i-1], yi4[i-1]) / 4.
        y_n2_3 = yi4[i-1] + step * f(grid[i-1] + step / 4., y_n4_2) / 2.
        y3 = yi4[i-1] + step * f(grid[i-1] + step / 2., y_n2_3)
        y4 = yi4[i-1] + step * ( f(grid[i-1], yi4[i-1]) +
                4 * f(grid[i-1] + step/2., y_n2_3) + f(grid[i], y3) ) / 6.
        yi4.append(y4)
    return yi4
```

```
Predictor-corrector:
y(0.0) = -1
y(0.1) = -0.900167599506
y(0.2) = -0.801332483673
y(0.3) = -0.704482201153
y(0.4) = -0.610584474262
y(0.5) = -0.520577527809
y(0.6) = -0.43536071269
y(0.7) = -0.355785518045
y(0.8) = -0.282647061914
y(0.9) = -0.216676145457
y(1.0) = -0.158531950178
```

Сістэма

Сістэму метадам прэдыктар-карэктар будзем рашаць з 2-ім парадкам дакладнасці ($O(h^3)$), то бок справядлівыя формулы:

$$y_{n+1}^{[3]} = y_n^{[3]} + \frac{h}{2}(f_n^{[3]} + f_{n+1}^{[2]})$$

$$z_{n+1}^{[3]} = z_n^{[3]} + \frac{h}{2}(g_n^{[3]} + g_{n+1}^{[2]})$$

$$y_{n+1}^{[2]} = y_n^{[3]} + hf_n^{[3]}$$

$$z_{n+1}^{[2]} = z_n^{[3]} + hg_n^{[3]}, \qquad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}.$$

```
def predictor_corrector_system():
    yi3 = [y0]
    zi3 = [z0]
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]

        y2 = yi3[i-1] + step * f(grid[i-1], yi3[i-1], zi3[i-1])
        y3 = yi3[i-1] + step * (f(grid[i-1], yi3[i-1], zi3[i-1]) + f(grid[i], y2, zi3[i-1])) / 2.
        yi3.append(y3)

        z2 = zi3[i-1] + step * g(grid[i-1], yi3[i-1], zi3[i-1])
        z3 = zi3[i-1] + step * (g(grid[i-1], yi3[i-1], zi3[i-1]) + g(grid[i], yi3[i-1], z2)) / 2.
        zi3.append(z3)
    return (yi3, zi3)
```

```
Predictor-corrector for system:
Y:
y(0.0) = 0.5
y(0.05) = 0.516721916223
y(0.1) = 0.536121101104
y(0.15) = 0.558296358137
y(0.2) = 0.583404263928
y(0.25) = 0.611655373195
y(0.3) = 0.643304953647
y(0.35) = 0.678637891004
y(0.4) = 0.717948576343
y(0.45) = 0.761518135242
y(0.5) = 0.809592858914
Z:
z(0.0) = 1.0
z(0.05) = 1.076875
z(0.1) = 1.15943250147
```

```
z(0.15) = 1.24831456526
z(0.2) = 1.34423940614
z(0.25) = 1.44801863056
z(0.3) = 1.56057712067
z(0.35) = 1.6829754276
z(0.4) = 1.81643423044
z(0.45) = 1.96236012598
z(0.5) = 2.12237184414
```

5 Метад Рунге-Кутта

Ураўненне

Для ўраўнення маем наступныя параметры: $q = 1$, $A_0 = 0$, $A_1 = 1$, $\alpha_1 = \beta_{10} = \frac{1}{2}$.
Тады формулы для падліку y_{n+1} маюць наступны выгляд:

$$y_{n+1} = y_n + A_0\varphi_0 + A_1\varphi_1 = y_n + \varphi_1, \quad \varphi_0 = hf_n, \quad \varphi_1 = hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}\varphi_0), \quad h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}$$

```
def runge_kutta():
    yi = [u0]
    a0 = 0.
    a1 = 1.
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]
        fi0 = step * f(grid[i-1], yi[i-1])
        fi1 = step * f(grid[i-1] + step/2., yi[i-1] + fi0/2.)
        y_new = yi[i-1] + a0 * fi0 + a1 * fi1
        yi.append(y_new)
    return yi
```

```
Runge-Kutta:
y(0.0) = -1
y(0.1) = -0.90012705443
y(0.2) = -0.801268046565
y(0.3) = -0.704408609366
y(0.4) = -0.610514268109
y(0.5) = -0.520520896324
y(0.6) = -0.435325448379
y(0.7) = -0.355777061627
y(0.8) = -0.282668618599
y(0.9) = -0.2167288559
y(1.0) = -0.15861510132
```

Сістэма

Для сістэмы вызначаем параметры наступным чынам: $q = 2$, $A_0 = \frac{1}{4}$, $A_1 = 0$, $A_2 = \frac{3}{4}$, $\alpha_1 = \frac{1}{3}$, $\alpha_2 = \frac{2}{3}$, $\beta_{10} = \frac{1}{3}$, $\beta_{20} = 0$, $\beta_{21} = \frac{2}{3}$. Будуем формулы падліку для y_{n+1}, z_{n+1}:

$$y_{n+1} = y_n + A_0\varphi_{0y} + A_1\varphi_{1y} + A_2\varphi_{2y} = y_n + \frac{1}{4}\varphi_{0y} + \frac{3}{4}\varphi_{2y},$$

$$z_{n+1} = z_n + A_0\varphi_{0z} + A_1\varphi_{1z} + A_2\varphi_{2z} = z_n + \frac{1}{4}\varphi_{0z} + \frac{3}{4}\varphi_{2z},$$

$$\varphi_{0y} = hf_n, \quad \varphi_{0z} = hg_n,$$

$$\varphi_{1y} = hf(x_n + \frac{1}{3}h, y_n + \frac{1}{3}\varphi_{0y}, z_n + \frac{1}{3}\varphi_{0z}), \quad \varphi_{1z} = hg(x_n + \frac{1}{3}h, y_n + \frac{1}{3}\varphi_{0y}, z_n + \frac{1}{3}\varphi_{0z}),$$

$$\varphi_{2y} = hf(x_n + \frac{2}{3}h, y_n + \frac{2}{3}\varphi_{1y}, z_n + \frac{2}{3}\varphi_{1z}), \quad \varphi_{2z} = hg(x_n + \frac{2}{3}h, y_n + \frac{2}{3}\varphi_{1y}, z_n + \frac{2}{3}\varphi_{1z}),$$

$$h = \frac{x_1 - x_0}{N}, \quad n = \overline{0, N-1}$$

```
def runge_kutta_system():
    yi = [y0]
    zi = [z0]
    a0 = 0.25
    a1 = 0.
    a2 = 0.75
    for i in range(1, len(grid)):
        step = grid[i] - grid[i-1]

        fi_0_y = step * f(grid[i-1], yi[i-1], zi[i-1])
        fi_0_z = step * g(grid[i-1], yi[i-1], zi[i-1])
        fi_1_y = step * f(grid[i-1] + step/3., yi[i-1] + fi_0_y/3., zi[i-1] + fi_0_z/3.)
        fi_1_z = step * g(grid[i-1] + step/3., yi[i-1] + fi_0_y/3., zi[i-1] + fi_0_z/3.)
        fi_2_y = step * f(grid[i-1] + 2.*step/3., yi[i-1] + 2.*fi_1_y/3., zi[i-1] + 2.*fi_1_z/3.)
        fi_2_z = step * g(grid[i-1] + 2.*step/3., yi[i-1] + 2.*fi_1_y/3., zi[i-1] + 2.*fi_1_z/3.)

        y_new = yi[i-1] + a0 * fi_0_y + a1 * fi_1_y + a2 * fi_2_y
        yi.append(y_new)

        z_new = zi[i-1] + a0 * fi_0_z + a1 * fi_1_z + a2 * fi_2_z
        zi.append(z_new)

    return (yi, zi)
```

```
Runge-Kutta for system:
Y:
y(0.0) = 0.5
y(0.05) = 0.515656434837
y(0.1) = 0.534026537105
y(0.15) = 0.555232681281
y(0.2) = 0.579455495042
y(0.25) = 0.606927585975
y(0.3) = 0.637921385868
y(0.35) = 0.672731182208
y(0.4) = 0.711650821141
y(0.45) = 0.75495025345
y(0.5) = 0.802855450697
Z:
z(0.0) = 1.0
z(0.05) = 1.07768694392
z(0.1) = 1.16115607324
z(0.15) = 1.25107154582
z(0.2) = 1.34818290595
z(0.25) = 1.45334412973
z(0.3) = 1.56753506833
z(0.35) = 1.69188489268
z(0.4) = 1.827696858
z(0.45) = 1.97647353019
z(0.5) = 2.13994167798
```

6 Экстрапаляцыйны метад Адамса

Будзем будаваць экстрапаляцыйны метад Адамса 3-га парадка дакладнасці, то бок $q = 2$. $A_i, i = \overline{0,2}$ знаходзім з сістэмы:

$$\sum_{i=0}^{2} A_i(-i)^j = \frac{1}{j+1}, \quad j = \overline{0,2} \tag{14}$$

Атрымліваем: $A_0 = \frac{23}{12}$, $A_1 = -\frac{16}{12}$, $A_2 = \frac{5}{12}$.
Тады y_{n+1} падлічваецца па формуле:

$$y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}), \qquad h = \frac{x_1 - x_0}{N}, \qquad n = \overline{2, N-1} \tag{15}$$

Патрэбныя для запуску алгарытма y_1, y_2 возьмем з метада паслядоўнага павышэння дакладнасці 3-га парадка дакладнасці.

```
def extra_adams(approx):
    yi = approx
    for i in range(3, len(grid)):
        step = grid[i] - grid[i-1]
        y_new = yi[i-1] + step * (23 * f(grid[i-1], yi[i-1]) - 15 * f(grid[i-2], yi[i-2]) +
                5 * f(grid[i-3], yi[i-3])) / 12.
        yi.append(y_new)
    return yi
```

```
Extra Adams:
y(0.0) = -1.0
y(0.1) = -0.900167599506
y(0.2) = -0.801332483673
y(0.3) = -0.704486738927
y(0.4) = -0.610596538535
y(0.5) = -0.520600122764
y(0.6) = -0.435396478576
y(0.7) = -0.355836889006
y(0.8) = -0.282716292556
y(0.9) = -0.216765374633
y(1.0) = -0.158643257314
```

7 Інтэрпаляцыйны метад Адамса

Інтэрпаляцыйны метад Адамса 2-га парадка дакладнасці пабудуем для сістэмы: $q = 1$.

$$\sum_{i=-1}^{q} A_i(-i)^j = \frac{1}{j+1}, \quad j = \overline{0,2} \tag{16}$$

З сістэмы вызначаем $A_i, i = \overline{-1,1}$: $A_{-1} = \frac{5}{12}$, $A_0 = \frac{8}{12}$, $A_1 = -\frac{1}{12}$.
Тады формулы для y_{n+1}, z_{n+1} выглядаюць наступным чынам:

$$\begin{cases} y_{n+1} = y_n + \frac{h}{12}(A_{-1}f_{n+1} + A_0 f_n + A_1 f_{n-1}), \\ z_{n+1} = z_n + \frac{h}{12}(A_{-1}g_{n+1} + A_0 g_n + A_1 g_{n-1}), \\ h = \frac{x_1 - x_0}{N}, \quad n = \overline{1, N-1}. \end{cases} \tag{17}$$

Сістэму (17) рашаем метадам простай ітэрацыі з дакладнасцю $\epsilon = 10^{-5}$, то бок справядлівая формула (13) (метад простай ітэрацыі для сістэмы быў запісаны раней пры апісанні нявяўнага метада Эйлера для сістэмы дыферэнцыяльных ураўненняў).

```
def inter_adams_system(y_approx, z_approx):
    yi = y_approx
    zi = z_approx
    a0 = 5./12
    a1 = 8./12
    a2 = -1./12
    for i in range(2, len(grid)):
        step = grid[i] - grid[i-1]
        f_local = lambda x, y, z: a0 * f(x, y, z) + a1 * f(grid[i-1], yi[i-1], zi[i-1]) +
                a2 * f(grid[i-2], yi[i-2], zi[i-2])
        g_local = lambda x, y, z: a0 * g(x, y, z) + a1 * g(grid[i-1], yi[i-1], zi[i-1]) +
                a2 * g(grid[i-2], yi[i-2], zi[i-2])
        result = simple_iterations(yi[i-1], zi[i-1], step, f_local, g_local, grid[i], yi[i-1], zi[i-1])
        yi.append(result[0])
        zi.append(result[1])
    return (yi, zi)

def simple_iterations(base_y, base_z, step, f, g, x, y0, z0):
    y_prev = y0
    z_prev = z0
    iter = MAX_ITERATIONS
    while iter > 0:
        y_new = base_y + step * f(x, y_prev, z_prev)
        z_new = base_z + step * g(x, y_prev, z_prev)
```

```
        iter -= 1
        if abs(y_prev - y_new) < EPS and abs(z_prev - z_new) < EPS:
            return (y_prev, z_prev)
        y_prev = y_new
        z_prev = z_new
    print "simple iterations: no convergence"
    return
```

```
Y:
y(0.0) = 0.5
y(0.05) = 0.516721916223
y(0.1) = 0.53510175299
y(0.15) = 0.556314365063
y(0.2) = 0.580541312026
y(0.25) = 0.60801529502
y(0.3) = 0.639008967289
y(0.35) = 0.673816952611
y(0.4) = 0.712733545778
y(0.45) = 0.756029244514
y(0.5) = 0.803930618642
Z:
z(0.0) = 1.0
z(0.05) = 1.076875
z(0.1) = 1.1604277317
z(0.15) = 1.2504319975
z(0.2) = 1.34764324985
z(0.25) = 1.45291660095
z(0.3) = 1.5672330863
z(0.35) = 1.69172315068
z(0.4) = 1.82769143332
z(0.45) = 1.97664202857
z(0.5) = 2.14030342682
```

Высновы

Ураўненне

Для задачы Кашы (3) вядомае дакладнае рашэнне:

$$u(x) = sin(x) - 1 \tag{18}$$

Таму мы маем магчымасць пабудаваць дакладную сетку значэнняў і параўнаць гэтыя значэнні з вынікамі працы кожнага з алгарытмаў.

```
Exact solution:
y(0.0) = -1.0
y(0.1) = -0.900166583353
y(0.2) = -0.801330669205
y(0.3) = -0.704479793339
y(0.4) = -0.610581657691
y(0.5) = -0.520574461396
y(0.6) = -0.435357526605
y(0.7) = -0.355782312762
y(0.8) = -0.2826439091
y(0.9) = -0.216673090373
y(1.0) = -0.158529015192
```

Знойдзем хібнасці кожнага з метадаў, то бок для кожнага x_i з сеткі знойдзем розніцу $u_i - y_i = u(x_i) - y(x_i)$, дзе u_i - дакладнае значэнне, y_i - набліжэнне.

Метад шэрагаў:

```
Series method
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = 8.33134948808e-08
u(0.2) - y(0.2) = 5.74204981008e-07
```

```
u(0.3) - y(0.3) = 1.43037073175e-06
u(0.4) - y(0.4) = 2.6126972027e-06
u(0.5) - y(0.5) = 4.08658620521e-06
u(0.6) - y(0.6) = 5.82281987083e-06
u(0.7) - y(0.7) = 7.798065847e-06
u(0.8) - y(0.8) = 9.99512268285e-06
u(0.9) - y(0.9) = 1.24029964328e-05
u(1.0) - y(1.0) = 1.50168860044e-05
```

Яўны метад Эйлера:

```
Explicit Euler:
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = -0.000166583353172
u(0.2) - y(0.2) = -0.000814510619714
u(0.3) - y(0.3) = -0.00189046507399
u(0.4) - y(0.4) = -0.00334537531258
u(0.5) - y(0.5) = -0.00513614894625
u(0.6) - y(0.6) = -0.0072267308694
u(0.7) - y(0.7) = -0.00958863068112
u(0.8) - y(0.8) = -0.0122010668214
u(0.9) - y(0.9) = -0.0150508625166
u(1.0) - y(1.0) = -0.0181322075461
```

Нияўны метад Эйлера:

```
Implicit Euler:
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = 0.000302864932127
u(0.2) - y(0.2) = 0.00103106960983
u(0.3) - y(0.3) = 0.00214351842673
u(0.4) - y(0.4) = 0.0036036376648
u(0.5) - y(0.5) = 0.00538040298753
u(0.6) - y(0.6) = 0.00744898502389
u(0.7) - y(0.7) = 0.00979111440253
u(0.8) - y(0.8) = 0.0123952609807
u(0.9) - y(0.9) = 0.0152567105458
u(1.0) - y(1.0) = 0.0183776087035
```

Метад паслядоўнага павышэння парадка дакладнасці 3-га парадка:

```
Predictor-corrector:
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = 1.01615236159e-06
u(0.2) - y(0.2) = 1.81446849989e-06
u(0.3) - y(0.3) = 2.40781471561e-06
u(0.4) - y(0.4) = 2.81657031409e-06
u(0.5) - y(0.5) = 3.06641340897e-06
u(0.6) - y(0.6) = 3.18608459438e-06
u(0.7) - y(0.7) = 3.20528264675e-06
u(0.8) - y(0.8) = 3.15281398755e-06
u(0.9) - y(0.9) = 3.05508399595e-06
u(1.0) - y(1.0) = 2.93498577661e-06
```

Метад Рунге-Кутта:

```
Runge-Kutta:
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = -3.9528923035e-05
u(0.2) - y(0.2) = -6.26226403999e-05
u(0.3) - y(0.3) = -7.11839729668e-05
u(0.4) - y(0.4) = -6.73895820182e-05
u(0.5) - y(0.5) = -5.35650719187e-05
u(0.6) - y(0.6) = -3.20782258875e-05
u(0.7) - y(0.7) = -5.25113498062e-06
u(0.8) - y(0.8) = 2.47094983472e-05
u(0.9) - y(0.9) = 5.57655270245e-05
u(1.0) - y(1.0) = 8.60861279295e-05
```

Экстрапаляцыйны метад Адамса 3-га парадка:

```
Extra Adams:
u(0.0) - y(0.0) = 0.0
u(0.1) - y(0.1) = 1.01615236159e-06
u(0.2) - y(0.2) = 1.81446849989e-06
u(0.3) - y(0.3) = 6.94558823411e-06
u(0.4) - y(0.4) = 1.48808438898e-05
u(0.5) - y(0.5) = 2.56613680059e-05
u(0.6) - y(0.6) = 3.89519706653e-05
u(0.7) - y(0.7) = 5.45762439096e-05
u(0.8) - y(0.8) = 7.23834555442e-05
u(0.9) - y(0.9) = 9.2284260969e-05
u(1.0) - y(1.0) = 0.000114242122081
```

Такім чынам, найлепшыя набліжэнні былі атрыманыя метадамі паслядоўнага павышэння дакладнасці 3-га парадку і метадам Рунге-Кутта - абодва далі 4-6 знакаў пасля коскі, якія супалі з дакладным рашэннем. Разам з тым экстрапаляцыйны метад Адамса, які таксама мусіць даваць 3-ці парадак, даў горшыя вынікі за два вышэй згаданых падыхода. Метады Эйлера далі вынікі, якія і чакаліся, якія, аднак, на практыцы маюць меншую вартасць за вынікі апошніх метадаў. Няяўны метад Эйлера, да таго ж, адносна складаны ў рэалізацыі. У сваю чаргу, метады Рунге-Кутта і прэдыктар-карэктар простыя і прыемныя ў рэалізацыі і даюць пажаданыя вынікі.

Сістэма

У адрозненні ад ўраўнення, дакладнага рашэння для сістэмы ўраўненняў мы не маем, таму вынікі можам рабіць толькі на падставе параўнання вынікаў працы алгарытмаў паміж сабой. Гэта кепскі падыход, які не дае дакладных лічбаў хібнасці падлікаў і па якім нават нельга меркаваць пра слушнасць знойдзенага рашэння. Разам з тым заўважым, што раскід значэнняў вельмі малы і рэдка дасягае значэння ў 0.1. Пры параўнанні вынікаў метадаў вышэйшых парадкаў якія, як мяркуецца, павінны даваць бліжэйшы да дакладнага рашэння вынік, можна заўважыць, што розніца ў значэннях з'яўляецца толькі ў 3 ці 4 знаку пасля коскі (напрыклад, пры параўнанні інтэрпаляцыйнага метада Адамса і метада Рунге-Кутта).

Як і для ўраўнення, так і для сістэмы найбольш надзейнымі апынуліся метады Рунге-Кутта, а таксама паслядоўнага павышэння парадка дакладнасці (прэдыктар-карэктар).