9. Obične diferencijalne jednačine, problem početne vrednosti

1. Rešavanje ODJ 1. reda

ODJ 1. reda je definisana kao:

$$ODJ_{impl}\big(x, f(x), f'(x)\big) = 0$$
$$f'(x) = ODJ_{ekspl}\big(x, f(x)\big) \tag{1}$$

, gde je ODJ_{ekspl} eksplicitni oblik diferencijalne jednačine po 1. izvodu $f'(x)$ funkcije.

Ako se funkcija nad intervalom $x \in [a, b]$ izdeli na n podintervala širine h, tada je:

$$h = \frac{b-a}{n}$$

Vrednost funkcije u bilo kojoj tački se može razviti u Tejlorov red (beskonačna suma):

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x)$$

Ojlerova metoda

Ojlerov metod uzima u obzir prva 2 člana reda:

$$f(x+h) \approx f(x) + hf'(x)$$

Iterativni zapis:

$$f_i \approx f_{i-1} + hf'_{i-1} \tag{2}$$

Zamenom (1) u (2) dobija se:

$$\boxed{f_i = f_{i-1} + h \cdot ODJ_{ekspl}(x_{i-1}, f_{i-1})} \tag{3}$$

Vrednost funkcije je poznata u početku intervala (početni uslov):

$$f_0 = f_a = f(a)$$

U n koraka se može izračunati vrednost funkcije u n tačaka:

$$\begin{aligned}
f_1 &= f_a + h \cdot ODJ_{ekspl}(a, f_a) \\
f_2 &= f_1 + h \cdot ODJ_{ekspl}(x_1, f_1) \\
&\vdots \\
f_{n-1} &= f_{n-2} + h \cdot ODJ_{ekspl}(x_{n-2}, f_{n-2}) \\
f_b &= f_{n-1} + h \cdot ODJ_{ekspl}(x_{n-1}, f_{n-1})
\end{aligned}$$

Rešenje ODJ je vektor:

$$f = \begin{bmatrix} f_a \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_b \end{bmatrix}$$

Vektor predstavlja vrednosti funkcije u tačkama $a + ih, i \in [0, n]$, tj. numeričko rešenje ODJ sa problemom početne vrednosti (PPV).

Zadatak 1. Napisati Ojlerovu metodu za rešavanje diferencijalne jednačine 1. reda sa PPV nad proizvoljnim intervalom. Pokušati prvo ručno jednu iteraciju Ojlerove metode nad funkcijom na intervalu $x \in [0,2\pi]$ sa 5 podintervala za sledeći PPV:

$$f'(x) = \cos x$$

$$f(0) = 0$$

```
dfX = @(x, fX) cos(x);
fX0 = 0;

a = 0;
b = 2*pi;
h = (b - a)/5;
```

1. Napraviti vektor nezavisno promenljive `x` u krajevima podintervala, a zatim napraviti vektor praznih vrednosti `fX` funkcije iste dužine. Postaviti prvi element vektora vrednosti funkcije po početnom uslovu:

```
x = a:h:b;
n = length(x);

fX = NaN(1, n);
fX(1) = fX0;
```

2. Napraviti jedan korak Ojlerove metode definisane u jednačini (3):

```
fX(2) = fX(1) + h*dfX(x(1), fX(1))
```

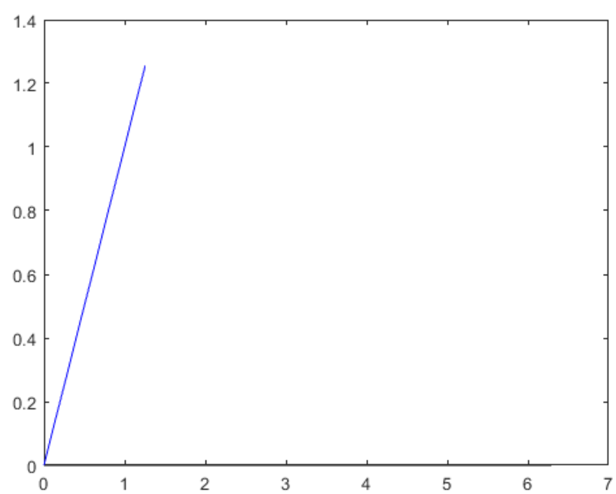
Rezultat:

```
fX =
         0    1.2566       NaN       NaN       NaN       NaN
```

3. Nacrtati grafik (`x, fX`):

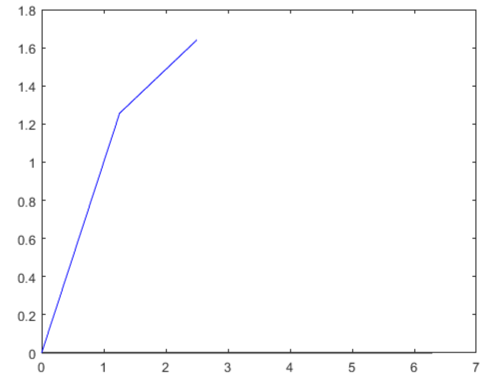
```
plot(x, fX, 'blue', [a b], [0 0], 'black')
```

Rezultat:



4. Ponoviti korake 2 i 3 za ostale podintervale:

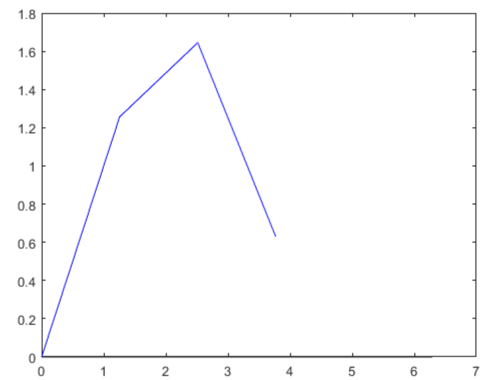
```
fX(3) = fX(2) + h*dfX(x(2), fX(2))
```



Rezultat:

```
fX =
     0    1.2566    1.6450       NaN       NaN       NaN
```

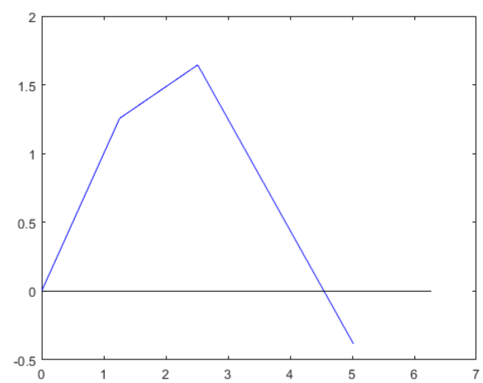
```
fX(4) = fX(3) + h*dfX(x(3), fX(3))
```



Rezultat:

```
fX =
     0    1.2566    1.6450    0.6283       NaN       NaN
```

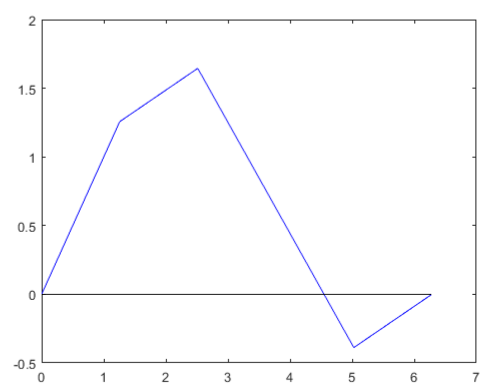
```
fX(5) = fX(4) + h*dfX(x(4), fX(4))
```



Rezultat:

```
fX =
     0    1.2566    1.6450    0.6283   -0.3883       NaN
```

```
fX(6) = fX(5) + h*dfX(x(5), fX(5))
```



Rezultat:

```
fX =
     0    1.2566    1.6450    0.6283   -0.3883   -0.0000
```

Uporediti korake:

```
fX(2) = fX(1) + h*dfX(x(1), fX(1))
fX(3) = fX(2) + h*dfX(x(2), fX(2))
fX(4) = fX(3) + h*dfX(x(3), fX(3))
fX(5) = fX(4) + h*dfX(x(4), fX(4))
fX(6) = fX(5) + h*dfX(x(5), fX(5))
```

Proširiti indekse:

```
fX(2) = fX(2 - 1) + h*dfX(x(2 - 1), fX(2 - 1))
fX(3) = fX(3 - 1) + h*dfX(x(3 - 1), fX(3 - 1))
fX(4) = fX(4 - 1) + h*dfX(x(4 - 1), fX(4 - 1))
fX(5) = fX(5 - 1) + h*dfX(x(5 - 1), fX(5 - 1))
fX(6) = fX(6 - 1) + h*dfX(x(6 - 1), fX(6 - 1))
```

5. Primetiti šta je promenljivo. Koraci 2 i 3 se mogu zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje:

```
fX = NaN(1, n);
fX(1) = fX0;
for it = 2:n
    fX(it) = fX(it - 1) + h*dfX(x(it - 1), fX(it - 1))

    plot(x, fX, 'blue', [a b], [0 0], 'black')
end
```

6. Sada je moguće definisati funkciju koja sadrži prethodni algoritam. Na početku funkcije zatvoriti eventualno postojeći prethodni grafik:

```
function fX = euler(a, b, h, fX0, dfX , plotSpeed)
    close
    .
    .
    .
end
```

7. Pauzirati algoritam u zavisnosti od tražene brzine iscrtavana postupka:

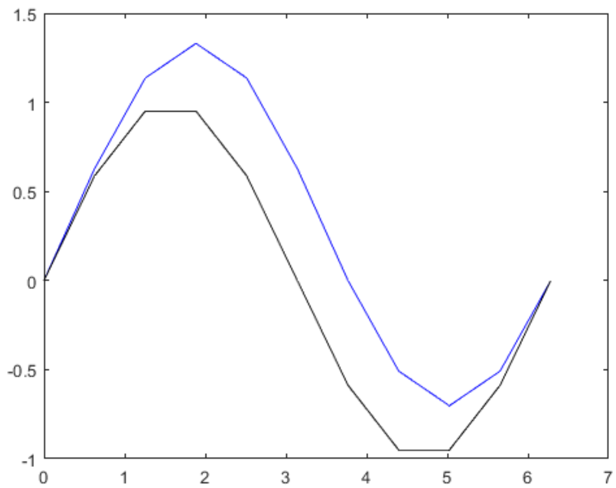
```
    .
    .
    .

    for it = 2:n
        .
        .
        .

        pause(1/plotSpeed)
    end
end
```

8. Testirati funkciju `euler` na primeru:

```
% f(x)' = cos(x)
% eksplicitni oblik jedna?ine po 1. izvodu funkcije
dfX = @(x, fX) cos(x)

% f'(0) = 0
% PPV mora da ima definisanu vrednost funkcije u početnoj tački
fX0 = 0;

a = 0;
b = 2*pi;
```

```
h = (b - a)/10;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x); % rešenje: f(x) = sin(x)
fXEuler = euler(a, b, h, fX0, dfX, 1.0);

plot(x, fXEuler, 'blue', x, fXTrue, 'black')
```

Rezultat:



Slika 1. Rešenje diferencijalne jednačine sa 10 koraka

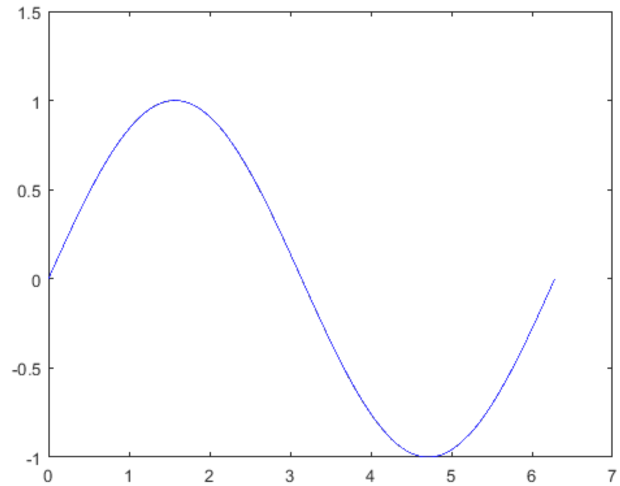
9. Napraviti 2 podfunkcije u funkciji `euler`. Ako je prosleđena brzina iscrtavanja postupka 0 ili manja, pozvati varijantu funkcije bez naredbi za iscrtavanje i pauziranje algoritma:

```
function fX = euler(a, b, h, fX0, dfX, plotSpeed)
    if plotSpeed <= 0
        fX = eulerWithoutPlot(a, b, h, fX0, dfX);
        return
    end
    fX = eulerWithPlot(a, b, h, fX0, dfX, plotSpeed);
end
```

10. Testirati funkciju `euler` na primeru:

```
dfX = @(x, fX) cos(x)
fX0 = 0;

a = 0;
b = 2*pi;
h = (b - a)/10000;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x);
fXEuler = euler(a, b, h, fX0, dfX, 0.0);

plot(x, fXEuler, 'blue', x, fXTrue, 'black')
```

Rezultat:



Slika 2. Rešenje diferencijalne jednačine sa 10000 koraka

Ojlerova metoda greši za veliki korak `h` (tj. za mali broj tačaka)!

RK4 metoda

RK metode vrednost funkcije u svakoj tački računaju na osnovu težinske sume koeficijenata čija se vrednost dobija tako da odgovara dodatnim članovima Tejlorovog reda, bez potrebe za računanjem izvoda višeg reda (koji figurišu u tim članovima). Time se povećava tačnost rešenja bez smanjivanja koraka h.

$$f(x+h) \approx f(x) + hf'(x) + \cdots$$
$$f(x+h) \approx f(x) + h \cdot \sum(k_1, k_2, \ldots)$$

, gde je $\sum(k_1, k_2, \ldots)$ težinska suma koeficijenata $k_1, k_2, \ldots$

RK4 metoda definiše 4 koeficijenta:

$$\boxed{\begin{aligned} k_1 &= ODJ_{ekspl}(x_{i-1}, f_{i-1}) \\ k_2 &= ODJ_{ekspl}(x_{i-1} + \frac{h}{2}, f_{i-1} + \frac{1}{2}k_1 h) \\ k_3 &= ODJ_{ekspl}(x_{i-1} + \frac{h}{2}, f_{i-1} + \frac{1}{2}k_2 h) \\ k_4 &= ODJ_{ekspl}(x_{i-1} + h, f_{i-1} + k_3 h) \\ f_i &= f_{i-1} + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}} \tag{4}$$

Zadatak 2. Napisati RK4 metodu za rešavanje diferencijalne jednačine 1. reda sa PPV nad proizvoljnim intervalom.

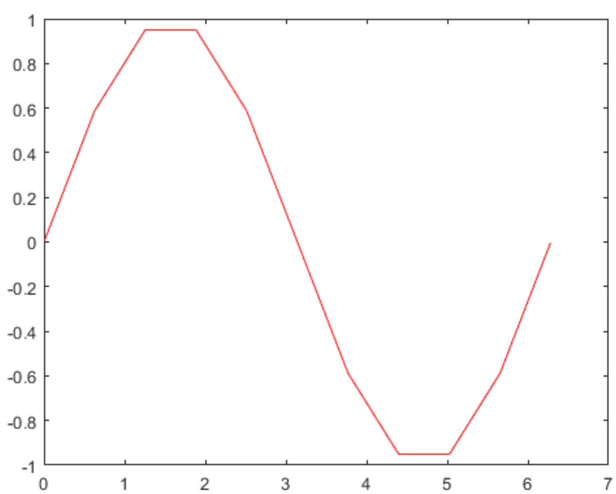
1. RK4 metoda ima isti iterativni postupak kao Ojlerova metoda, ali dolazi do vrednosti funkcije po jednačini (4):

```
function fX = rk4WithPlot(a, b, h, fX0, dfX, plotSpeed)
    .
    .
    .

    for it = 2:n
        k1 = dfX(x(it - 1), fX(it - 1));
        k2 = dfX(x(it - 1) + h/2, fX(it - 1) + k1*h/2);
        k3 = dfX(x(it - 1) + h/2, fX(it - 1) + k2*h/2);
        k4 = dfX(x(it - 1) + h, fX(it - 1) + k3*h);

        fX(it) = fX(it - 1) + h/6*(k1 + 2*k2 + 2*k3 + k4);

        .
        .
        .
    end
end
```

2. Testirati funkciju `rk4` na primeru:

```
dfX = @(x, fX) cos(x)
fX0 = 0;

a = 0;
b = 2*pi;
h = (b - a)/10;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x);
fXRK4 = rk4(a, b, h, fX0, dfX, 1.0);

plot(x, fXTrue, 'black', x, fXRK4, 'red')
```

Rezultat:

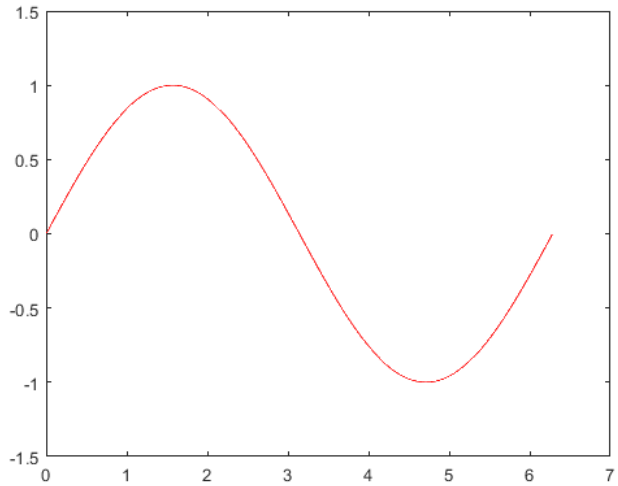


Slika 3. Rešenje diferencijalne jednačine sa 10 koraka

3. Testirati funkciju `rk4` na primeru:

```
dfX = @(x, fX) cos(x)
fX0 = 0;

a = 0;
b = 2*pi;
h = (b - a)/1000;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x);
fXRK4 = rk4(a, b, h, fX0, dfX, 0.0);

plot(x, fXTrue, 'black', x, fXRK4, 'red')
```

Rezultat:



Slika 4. Rešenje diferencijalne jednačine sa 1000 koraka

RK4 metoda greši mnogo manje od Ojlerove metode čak i za veliki korak `h` (tj. za mali broj tačaka)!

2. Rešavanje ODJ proizvoljnog reda

Ojlerova metoda

ODJ r-tog reda je definisana kao:

$$\begin{aligned} ODJ_{impl}\left(x, f(x), f'(x), f''(x), \dots, f^{(r)}(x)\right) &= 0 \\ f^{(r)}(x) &= ODJ_{ekspl}\left(x, f(x), f'(x), f''(x), \dots, f^{(r-1)}(x)\right) \end{aligned} \tag{1}$$

, gde je ODJ_{ekspl} eksplicitni oblik diferencijalne jednačine po r-tom izvodu $f^{(r)}(x)$ funkcije.

Ojlerov korak je moguće diferencirati $(r-1)$ puta:

$$\begin{aligned} f_i &= f_{i-1} + hf'_{i-1} \\ f'_i &= f'_{i-1} + hf''_{i-1} \\ &\vdots \\ f^{(r-2)}_i &= f^{(r-2)}_{i-1} + hf^{(r-1)}_{i-1} \\ f^{(r-1)}_i &= f^{(r-1)}_{i-1} + hf^{(r)}_{i-1} \end{aligned} \tag{2}$$

Zamenom (1) u (2) dobija se:

$$\boxed{\begin{aligned} f_i &= f_{i-1} + hf'_{i-1} \\ f'_i &= f'_{i-1} + hf''_{i-1} \\ &\vdots \\ f^{(r-2)}_i &= f^{(r-2)}_{i-1} + hf^{(r-1)}_{i-1} \\ f^{(r-1)}_i &= f^{(r-1)}_{i-1} + h \cdot ODJ_{ekspl}\left(x_{i-1}, f_{i-1}, f'_{i-1}, \dots, f^{(r-2)}_{i-1}, f^{(r-1)}_{i-1}\right) \end{aligned}} \tag{3}$$

Vrednost funkcije i svih njenih izvoda do reda $(r-1)$ je poznata u početku intervala (početni uslov):

$$\begin{aligned} f_0 &= f_a = f(a) \\ f'_0 &= f'_a = f'(a) \\ &\vdots \\ f^{(r-2)}_0 &= y^{(r-2)}_a = y^{(r-2)}(a) \\ y^{(r-1)}_0 &= y^{(r-1)}_a = y^{(r-1)}(a) \end{aligned}$$

U n koraka se može izračunati vrednost funkcije u n tačaka:

korak 1:

$$\begin{aligned} f_1 &= f_a + hf'_a \\ f'_1 &= f'_a + hf''_a \\ &\vdots \\ f^{(r-2)}_1 &= f^{(r-2)}_a + hf^{(r-1)}_a \\ f^{(r-1)}_1 &= f^{(r-1)}_a + h \cdot ODJ_{ekspl}\left(a, f_a, f'_a, \dots, f^{(r-2)}_a, f^{(r-1)}_a\right) \end{aligned}$$

korak 2:

$$\begin{aligned} f_2 &= f_1 + hf'_1 \\ f'_2 &= f'_1 + hf''_1 \\ &\vdots \\ f^{(r-2)}_2 &= f^{(r-2)}_1 + hf^{(r-1)}_1 \\ f^{(r-1)}_2 &= f^{(r-1)}_1 + h \cdot ODJ_{ekspl}\left(x_1, f_1, f'_1, \dots, f^{(r-2)}_1, f^{(r-1)}_1\right) \end{aligned}$$

korak $(n-1)$:

$$\begin{aligned} f_{n-1} &= f_{n-2} + hf'_{n-2} \\ f'_{n-1} &= f'_{n-2} + hf''_{n-2} \\ &\vdots \\ f^{(r-2)}_{n-1} &= f^{(r-2)}_{n-2} + hf^{(r-1)}_{n-2} \\ f^{(r-1)}_{n-1} &= f^{(r-1)}_{n-2} + h \cdot ODJ_{ekspl}\left(x_{n-2}, f_{n-2}, f'_{n-2}, \dots, f^{(r-2)}_{n-2}, f^{(r-1)}_{n-2}\right) \end{aligned}$$

korak n:

$$\begin{aligned} f_b &= f_{n-1} + hf'_{n-1} \\ f'_b &= f'_{n-1} + hf''_{n-1} \\ &\vdots \\ f^{(r-2)}_b &= f^{(r-2)}_{n-1} + hf^{(r-1)}_{n-1} \\ f^{(r-1)}_b &= f^{(r-1)}_{n-1} + h \cdot ODJ_{ekspl}\left(x_{n-1}, f_{n-1}, f'_{n-1}, \dots, f^{(r-2)}_{n-1}, f^{(r-1)}_{n-1}\right) \end{aligned}$$

Rešenje ODJ je vektor:

$$f = \begin{bmatrix} f_a \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_b \end{bmatrix}$$

Vektor predstavlja vrednosti funkcije u tačkama $a + ih, i \in [0, n]$, tj. numeričko rešenje ODJ r-tog reda sa problemom početne vrednosti (PPV).

Zadatak 3. Napisati Ojlerovu metodu za rešavanje diferencijalne jednačine proizvoljnog reda sa PPV nad proizvoljnim intervalom.
Pokušati prvo ručno jednu iteraciju Ojlerove metode nad funkcijom na intervalu $x \in [0,2\pi]$ sa 5 podintervala za sledeći PPV:

$$f'''(x) = -\cos x$$

$$f(0) = 0$$
$$f'(0) = 1$$
$$f''(0) = 0$$

```
% f(x)"' = cos(x)
% eksplicitni oblik jednačine po 3. izvodu funkcije
% zbog uniformnosti pozivanja moraju kao parametri da se navedu fX, dfX i ddfX
% iako se ne koriste
ddfX = @(x, fX, dfX, ddfX) -cos(x)

% f(0) = 0
% f'(0) = 1
% f"(0) = 0
% PPV mora da ima definisane vrednosti svih izvoda funkcije do reda
% za 1 manjeg od reda jednačine u početnoj tački
nfX0 = [0 1 0];

a = 0;
b = 2*pi;
h = (b - a)/5;
```

1. Napraviti vektor nezavisno promenljive `x` u krajevima podintervala i izračunati red diferencijalne jednačine `order` na osnovu dužine vektora `nfX0` a zatim napraviti matricu praznih vrednosti `fX` funkcije i njenih izvoda. Postaviti prvi element vektora vrednosti funkcije po početnom uslovu:

```
x = a:h:b;
n = length(x);
order = length(nfX0) % red diferencijalne jedna?ine

% prva vrsta čuva vrednosti funkcije
% druga vrsta čuva vrednosti 1. izvoda funkcije, itd.
% prva kolona čuva poznate vrednosti funkcije i njenih izvoda
% druga kolona čuva vrednosti funkcije i njenih izvoda u tački (a + h), itd.
% [f(a)          f(a + h)        f(a + 2h)        ...]
% [f'(a)         f'(a + h)       f"(a + 2h)       ...]
% .
% .
% .
% [f^(r-1)(a)    f^(r-1)(a + h)  f^(r-1)(a + 2h) ...]
fnX = NaN(order, n);
fnX(:, 1) = nfX0';
```

Rezultat:

```
order =
         3
```

2. Napraviti jedan korak Ojlerove metode definisane u jednačini (3):

```
fnX(1, 2) = fnX(1, 1) + h*fnX(2, 1)
fnX(2, 2) = fnX(2, 1) + h*fnX(3, 1)

% lista argumenata: [x(1) f(1) f'(1) ... f^(r-1)(1)]
args = num2cell([x(1) fnX(:, 1)']);
fnX(order, 2) = fnX(order, 1) + h*dnfX(args{:})
```

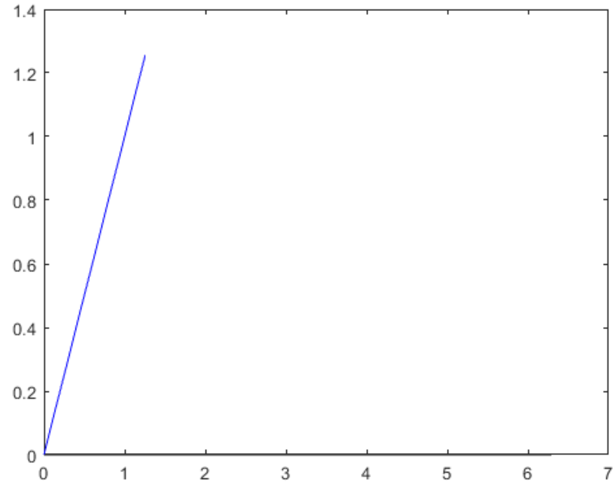
Rezultat:

```
fnX =
         0    1.2566       NaN       NaN       NaN       NaN
    1.0000    1.0000       NaN       NaN       NaN       NaN
         0   -1.2566       NaN       NaN       NaN       NaN
```

3. Nacrtati grafik (`x, fX`) :

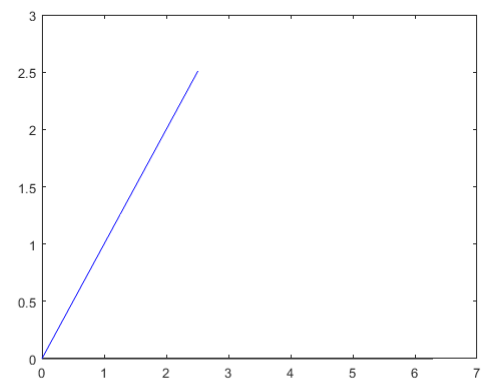
```
plot(x, fnX(1, :), 'blue', [a b], [0 0], 'black')
```

Rezultat:



Ponoviti korake 2 i 3 za ostale podintervale:

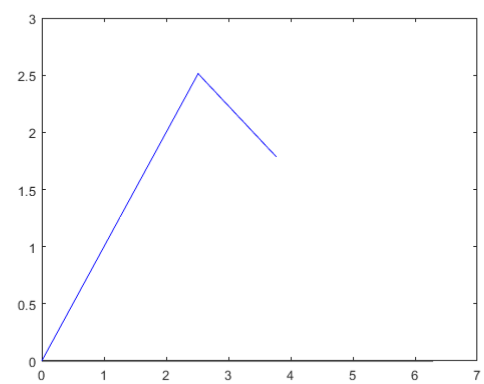
```
fnX(1, 3) = fnX(1, 2) + h*fnX(2, 2)
fnX(2, 3) = fnX(2, 2) + h*fnX(3, 2)

args = num2cell([x(2) fnX(:, 2)']);
fnX(order, 3) = fnX(order, 2) + h*dnfX(args{:})
```



Rezultat:

```
fnX =
         0    1.2566    2.5133       NaN       NaN       NaN
    1.0000    1.0000   -0.5791       NaN       NaN       NaN
         0   -1.2566   -1.6450       NaN       NaN       NaN
```

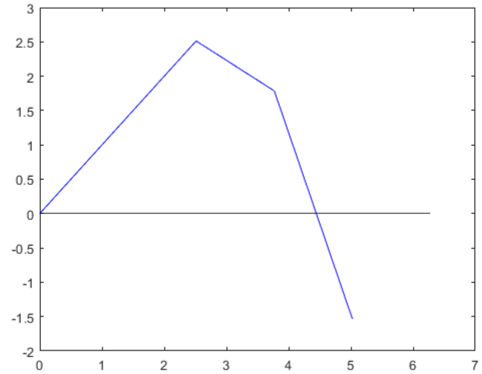
```
fnX(1, 4) = fnX(1, 3) + h*fnX(2, 3)
fnX(2, 4) = fnX(2, 3) + h*fnX(3, 3)

args = num2cell([x(3) fnX(:, 3)']);
fnX(order, 4) = fnX(order, 3) + h*dnfX(args{:})
```



Rezultat:

```
fnX =
         0    1.2566    2.5133    1.7855       NaN       NaN
    1.0000    1.0000   -0.5791   -2.6463       NaN       NaN
         0   -1.2566   -1.6450   -0.6283       NaN       NaN
```

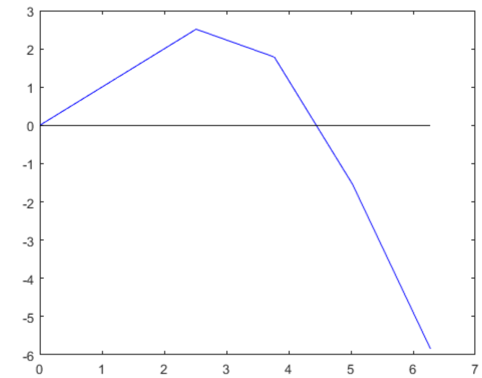
```
fnX(1, 5) = fnX(1, 4) + h*fnX(2, 4)
fnX(2, 5) = fnX(2, 4) + h*fnX(3, 4)

args = num2cell([x(4) fnX(:, 4)']);
fnX(order, 5) = fnX(order, 4) + h*dnfX(args{:})
```



Rezultat:

```
fnX =
         0    1.2566    2.5133    1.7855   -1.5399       NaN
    1.0000    1.0000   -0.5791   -2.6463   -3.4358       NaN
         0   -1.2566   -1.6450   -0.6283    0.3883       NaN
```

```
fnX(1, 6) = fnX(1, 5) + h*fnX(2, 5)
fnX(2, 6) = fnX(2, 5) + h*fnX(3, 5)

args = num2cell([x(5) fnX(:, 5)']);
fnX(order, 6) = fnX(order, 5) + h*dnfX(args{:})
```



Rezultat:

```
fnX =
         0    1.2566    2.5133    1.7855   -1.5399   -5.8575
    1.0000    1.0000   -0.5791   -2.6463   -3.4358   -2.9478
         0   -1.2566   -1.6450   -0.6283    0.3883    0.0000
```

Uporediti korake:

```
fnX(1, 2) = fnX(1, 1) + h*fnX(2, 1)
fnX(2, 2) = fnX(2, 1) + h*fnX(3, 1)

args = num2cell([x(1) fnX(:, 1)']);
fnX(order, 2) = fnX(order, 1) + h*dnfX(args{:})


fnX(1, 4) = fnX(1, 3) + h*fnX(2, 3)
fnX(2, 4) = fnX(2, 3) + h*fnX(3, 3)

args = num2cell([x(3) fnX(:, 3)']);
fnX(order, 4) = fnX(order, 3) + h*dnfX(args{:})


fnX(1, 5) = fnX(1, 4) + h*fnX(2, 4)
fnX(2, 5) = fnX(2, 4) + h*fnX(3, 4)

args = num2cell([x(4) fnX(:, 4)']);
fnX(order, 5) = fnX(order, 4) + h*dnfX(args{:})


fnX(1, 6) = fnX(1, 5) + h*fnX(2, 5)
fnX(2, 6) = fnX(2, 5) + h*fnX(3, 5)

args = num2cell([x(5) fnX(:, 5)']);
fnX(order, 6) = fnX(order, 5) + h*dnfX(args{:})
```

Proširiti indekse:

```
fnX(1, 2) = fnX(1, 2 - 1) + h*fnX(1 + 1, 2 - 1)
fnX(2, 2) = fnX(2, 2 - 1) + h*fnX(2 + 1, 2 - 1)

args = num2cell([x(2 - 1) fnX(:, 2 - 1)']);
fnX(order, 2) = fnX(order, 2 - 1) + h*dnfX(args{:})


fnX(1, 4) = fnX(1, 4 - 1) + h*fnX(1 + 1, 4 - 1)
fnX(2, 4) = fnX(2, 4 - 1) + h*fnX(2 + 1, 4 - 1)

args = num2cell([x(4 - 1) fnX(:,4 - 1)']);
fnX(order, 4) = fnX(order, 4 - 1) + h*dnfX(args{:})


fnX(1, 5) = fnX(1, 5 - 1) + h*fnX(1 + 1, 5 - 1)
fnX(2, 5) = fnX(2, 5 - 1) + h*fnX(2 + 1, 5 - 1)

args = num2cell([x(5 - 1) fnX(:,5 - 1)']);
fnX(order, 5) = fnX(order, 5 - 1) + h*dnfX(args{:})


fnX(1, 6) = fnX(1, 6 - 1) + h*fnX(1 + 1, 6 - 1)
fnX(2, 6) = fnX(2, 6 - 1) + h*fnX(2 + 1, 6 - 1)

args = num2cell([x(6 - 1) fnX(:,6 - 1)']);
fnX(order, 6) = fnX(order, 6 - 1) + h*dnfX(args{:})
```

4. Primetiti šta je promenljivo. Koraci 2 i 3 se mogu zapisati jednom *for* petljom, pri čemu promenljivi indeksi zavise od indeksa *for* petlje:

```
fnX = NaN(order, n);
fnX(:, 1) = nfX0';
for it = 2:n
    fnX(1, it) = fnX(1, it - 1) + h*fnX(1 + 1, it - 1);
    fnX(2, it) = fnX(2, it - 1) + h*fnX(2 + 1, it - 1);

    args = num2cell([x(it - 1) fnX(:, it - 1)']);
    fnX(order, it) = fnX(order, it - 1) + h*dnfX(args{:});

    plot(x, fnX(1, :), 'blue', [a b], [0 0], 'black')
end
```

5. Promenljivi indeksi 1. dimenzije matrice se takođe mogu zameniti jednom *for* petljom, pri čemu oni zavise od indeksa *for* petlje:

```
for it = 2:n
    for itOrder = 1:order - 1
        fnX(itOrder, it) = fnX(itOrder, it - 1) + h*fnX(itOrder + 1, it - 1);
    end
    args = num2cell([x(it - 1) fnX(:, it - 1)']);
    fnX(order, it) = fnX(order, it - 1) + h*dnfX(args{:})

    plot(x, fnX(1, :), 'blue', [a b], [0 0], 'black')
end
```

6. Sada je moguće definisati funkciju koja sadrži prethodni algoritam. Na početku funkcije zatvoriti eventualno postojeći prethodni grafik:

```
function [fX, fnX] = eulerN(a, b, h, nfX0, dnfX, plotSpeed)
    close
    .
    .
    .
end
```

7. Matrica `fnX` će biti vraćena kao 2. povratna vrednost (korisnik može da je traži po potrebi). Samo 1. vrsta matrice (vrednosti funkcije u svim tačkama) će biti vraćena kao 1. povratna vrednost:

```
function [fX, fnX] = eulerN(a, b, h, nfX0, dnfX, plotSpeed)
    .
    .
    .
    fX = fnX(1, :); % prva vrsta čuva vrednosti funkcije
end
```

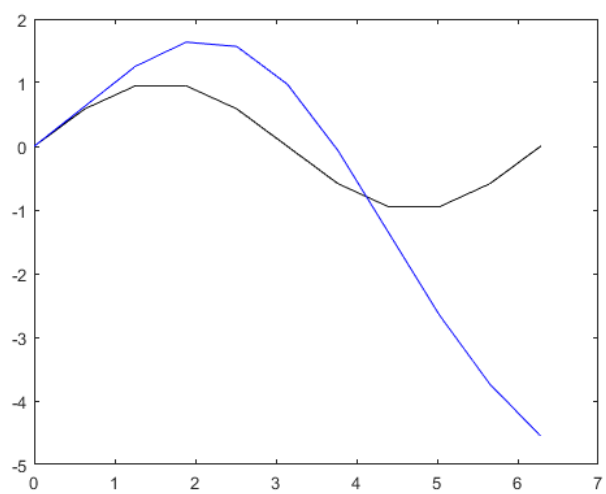
8. Pauzirati algoritam u zavisnosti od tražene brzine iscrtavana postupka:

```
    .
    .
    .
    for it = 2:n
        .
        .
        .
        pause(1/plotSpeed)
    end
    fX = fnX(1, :);
end
```

9. Testirati funkciju `eulerN` na primeru:

```
ddfX = @(x, fX, dfX, ddfX) -cos(x)
nfX0 = [0 1 0];

a = 0;
b = 2*pi;
h = (b - a)/10;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x); % rešenje: f(x) = sin(x)
fXEuler = eulerN(a, b, h, nfX0, ddfX, 1.0);

plot(x, fXTrue, 'black', x, fXEuler, 'blue')
```

Rezultat:



Slika 5. Rešenje diferencijalne jednačine sa 10 koraka

Što je red jednačine veći, veće su i greške!

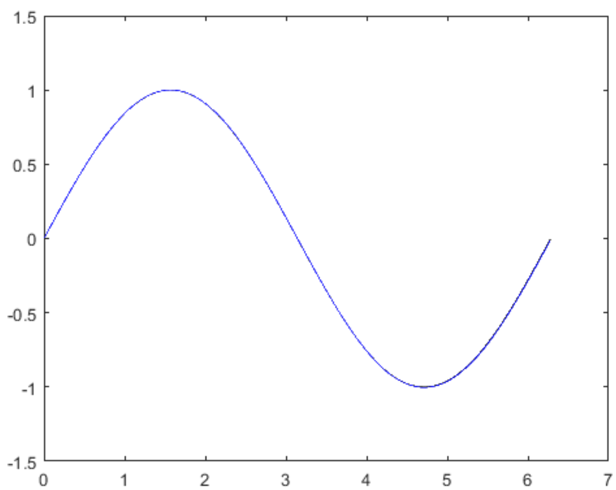
10. Napraviti 2 podfunkcije u funkciji `eulerN`. Ako je prosleđena brzina iscrtavanja postupka 0 ili manja, pozvati varijantu funkcije bez naredbi za iscrtavanje i pauziranje algoritma:

```
function [fX, fnX] = eulerN(a, b, h, nfX0, dnfX, plotSpeed)
    if plotSpeed <= 0
        [fX, fnX] = eulerNWithoutPlot(a, b, h, nfX0, dnfX);
        return
    end
    [fX, fnX] = eulerNWithPlot(a, b, h, nfX0, dnfX, plotSpeed);
end
```

11. Testirati funkciju `euler` na primeru:

```
ddfX = @(x, fX, dfX, ddfX) -cos(x)
nfX0 = [0 1 0];

a = 0;
b = 2*pi;
h = (b - a)/10000;

x = a:h:b;
fXTrue = feval(@(x) sin(x), x);
fXEuler = eulerN(a, b, h, nfX0, ddfX, 0.0);

plot(x, fXTrue, 'black', x, fXEuler, 'blue')
```

Rezultat:



Slika 6. Rešenje diferencijalne jednačine sa 10000 koraka

* **Zadatak 4**. Napisati RK4 metodu za rešavanje diferencijalne jednačine proizvoljnog reda sa PPV nad proizvoljnim intervalom.