demo-01_hw-05

January 12, 2021

1 Analiza i projektiranje računalom - 5. domaća zadaća

U okviru ove zadaće potrebno je ostvariti metode numeričke integracije po postupku ***Runge-Kutta*** 4. reda (*u skripti: str. 7-35*), ***trapeznom*** postupku, ***Eulerovom*** postupku, ***obrnutom Eulerovom*** postupku, i ***prediktorsko-korektorskom*** postupku. Sustav je općenitog oblika $\vec{x} = \vec{A}\vec{x} + \vec{B}r(\vec{t})$. Program treba (iz datoteka) učitavati matrice linearnog sustava diferencijalnih jednadžbi ($\vec{A}$ i $\vec{B}$) te početno stanje $\vec{x}(t = 0)$. Za uporabu trapeznog i obrnutog Eulerovog postupka potrebno je zadani linearni sustav prethodno transformirati u eksplicitni oblik (*skripta 7-24, 25*).

Potrebno je bez prevođenja programa omogućiti zadavanje **željenog koraka integracije** (T) i **vremenskog intervala** za koji se provodi postupak $[0, t_{MAX}]$. Za prediktorsko-korektorski postupak potrebno je omogućiti da se bez prevođenja programa bilo koji dostupni eksplicitni postupak može koristiti kao prediktor, bilo koji implicitni postupak kao korektor te da je moguće definirati koliko puta će se korektor primijeniti. Program treba riješavati sustave svim sljedećim postupcima:

- Eulerovim
- obrnutim Eulerovim
- trapeznim
- Runge-Kutta 4. rede
- PE(CE)^2 (prediktor Euler, korektor obrnuti Euler)
- PECE (prediktor Euler, korektor trapezni postupak)

Prilikom rada potrebno je ispisivati varijable stanja na ekran, no ne u svakoj iteraciji nego svakih nekoliko iteracija (omogućiti da taj broj zadaje korisnik). Osim na ekran, ispis je uputno preusmjeriti i u datoteku. Nakon završetka postupka potrebno je **grafički prikazati kretanje varijabli stanja** za oba postupka izračunavanja (vodoravna os je vrijeme, na uspravnoj su vrijednosti varijabli stanja). Crtanje se može izvesti bilo kojim pomoćnim alatom, npr. čitanjem izračunatih vrijednosti iz datoteke.

1.1 Priprema za izvođenje

[1]:
```
import os

CD_KEY = "--HW05_IN_ROOT"
```

[2]:
```
if (
    CD_KEY not in os.environ
    or os.environ[CD_KEY] is None
    or len(os.environ[CD_KEY]) == 0
```

```
    or os.environ[CD_KEY] == "false"
):
    %cd ..
else:
    print(os.getcwd())

os.environ[CD_KEY] = "true"
```

```
/mnt/data/projekti/faks/AIPR/dz/dz-05
```

1.2 Učitavanje paketa

[3]:
```
import warnings

from matplotlib import pyplot as plt
import numpy as np

from src.constants import SUB
from src.integrators.explicit import EulerIntegrator, RungeKutta4Integrator
from src.integrators.implicit import InverseEulerIntegrator,␣
 ↪TrapezoidalIntegrator
from src.integrators.predictor_corrector import PredictorCorrectorIntegrator
from src.utils import get_state_over_time_graphs, print_diffs
```

[4]:
```
np.set_printoptions(precision=3, suppress=True)
np.random.seed(21051208)
warnings.filterwarnings('ignore')
```

1.3 Zadatci

1.3.1 Zadatak 1

Izračunajte ponašanje sljedećeg sustava:

$$\vec{x_{k+1}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \vec{x_k} \quad \vec{x_0} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

s periodom integracije $T = 0.01$ i $t_{MAX} = 10$ za sve zadane postupke. Sustav predstavlja matematičko njihalo, gdje je x_{0_0} početni odmak od ravnotežnog položaja, a x_{0_1} je početna brzina.

Odgovor

[5]:
```
t1_step = 0.01
t1_interval =(0, 10)

t1_a = np.array([
    [0, 1],
    [-1, 0],
```

```
])
t1_b = np.array([
    [0, 0],
    [0, 0],
])

t1_initial_state = np.array([
    [1],
    [1],
])

t1_steps_to_print = 100
```

[6]:
```
t1_integrators = (
    EulerIntegrator(),
    InverseEulerIntegrator(),
    TrapezoidalIntegrator(),
    RungeKutta4Integrator(),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=InverseEulerIntegrator(),
    ),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=TrapezoidalIntegrator(),
    ),
)

t1_results = dict()
```

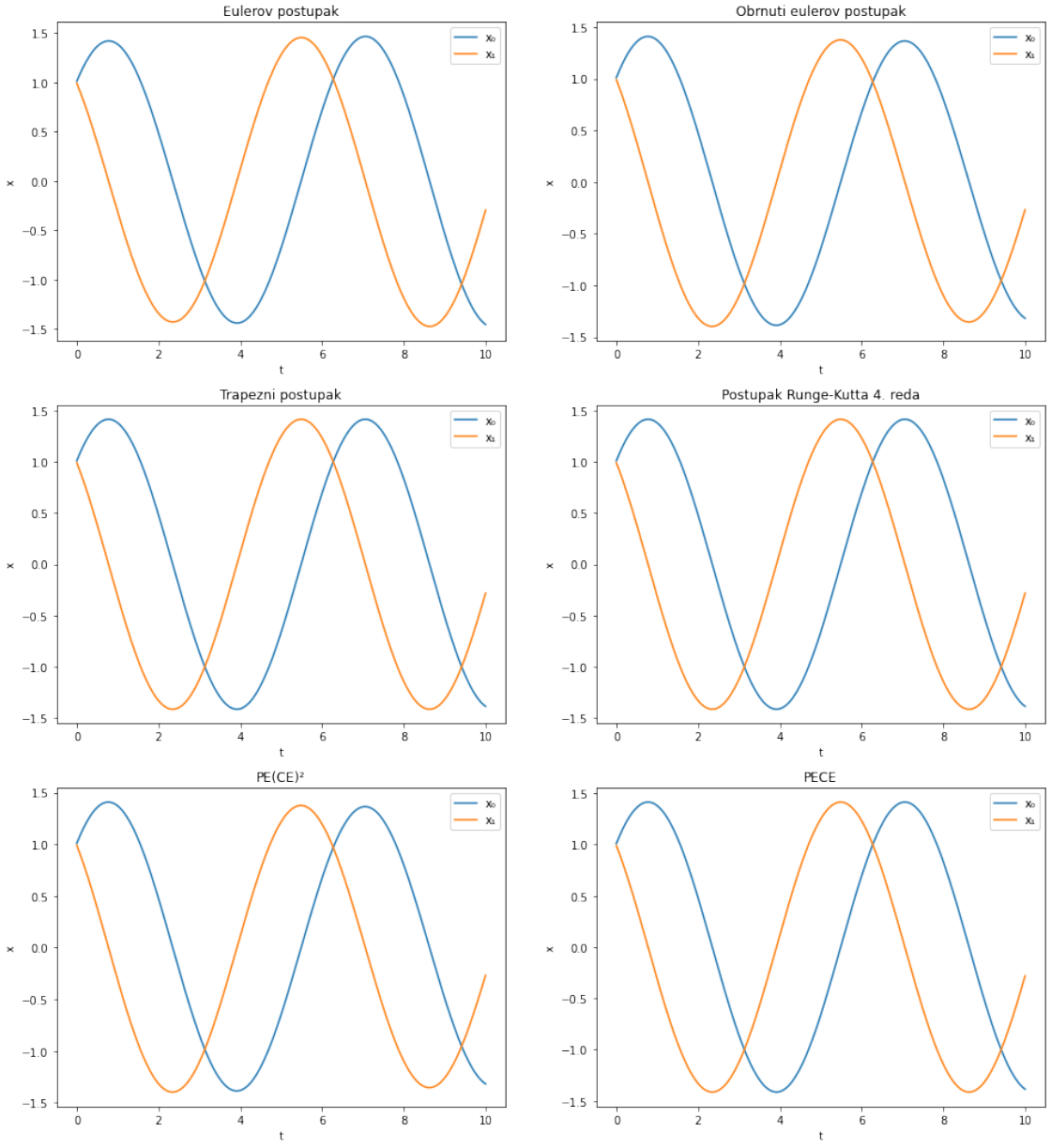
[7]:
```
for integrator in t1_integrators[:-2]:
    t1_results[integrator.name] = integrator(
        step=t1_step,
        interval=t1_interval,
        initial_state=t1_initial_state,
        time_function=None,
        steps_to_print=t1_steps_to_print,
        a=t1_a,
        b=t1_b
    )
```

```
x_1000(t = 10.0) = [[-1.454 -0.311]]: : 1001it [00:00, 30358.34it/s]
x_1000(t = 10.0) = [[-1.316 -0.281]]: : 1001it [00:00, 23676.59it/s]
x_1000(t = 10.0) = [[-1.383 -0.295]]: : 1001it [00:00, 17091.04it/s]
x_1000(t = 10.0) = [[-1.383 -0.295]]: : 1001it [00:00, 12913.49it/s]
```

[8]:
```
for integrator, n_corrector_repeats in zip(t1_integrators[-2:], (2, 1)):
    t1_results[integrator.name] = integrator(
        step=t1_step,
        interval=t1_interval,
        initial_state=t1_initial_state,
        time_function=None,
        steps_to_print=t1_steps_to_print,
        a=t1_a,
        b=t1_b,
        n_corrector_repeats=n_corrector_repeats
    )
```

```
x_1000(t = 10.0) = [[-1.316 -0.281]]: : 1001it [00:00, 17171.07it/s]
x_1000(t = 10.0) = [[-1.383 -0.295]]: : 1001it [00:00, 19951.62it/s]
```

[9]:
```
t1_fig, t1_ax = get_state_over_time_graphs(t1_results)
```



Analitičko rješenje sustava je

$$\vec{x} = \begin{bmatrix} x_{0_0} \cos t + x_{0_1} \sin t \\ x_{0_1} \cos t - x_{0_0} \sin t \end{bmatrix}$$

Za svaki postupak izračunajte kumulativnu pogrešku koju svaki od postupaka napravi tijekom izvođenja, na način da zbrojite apsolutnu razliku dobivenog i stvarnog rješenja u svakoj točki integracije, a zbroj prikažete na kraju izvođenja programa.

Odgovor

[10]:
```
t1_true_solutions = list()

for t in np.arange(t1_interval[0], t1_interval[1] + t1_step, t1_step):
    t1_true_solutions.append([
        [t1_initial_state[0][0] * np.cos(t) + t1_initial_state[1][0] * np.
 ↪sin(t)],
        [t1_initial_state[1][0] * np.cos(t) - t1_initial_state[0][0] * np.
 ↪sin(t)]
    ])

t1_true_solutions = np.array(t1_true_solutions)
```

[11]:
```
t1_diffs = dict()

for key, value in t1_results.items():
    t1_diffs[key] = np.zeros(shape=t1_true_solutions[0].shape)

    for state, true_solution in zip(value["states"], t1_true_solutions):
        t1_diffs[key] += abs(state["x"] - true_solution)
```

[12]:
```
print_diffs(t1_diffs, n_decimals=6)
```

```
Eulerov postupak:
        Δx  = 22.429695
        Δx  = 23.648003

Obrnuti eulerov postupak:
        Δx  = 21.728698
        Δx  = 22.824658

Trapezni postupak:
        Δx  = 0.038718
        Δx  = 0.036793

Postupak Runge-Kutta 4. reda:
        Δx  = 0.000000
        Δx  = 0.000000

PE(CE)²:
        Δx  = 21.732969
        Δx  = 22.829150

PECE:
        Δx  = 0.077510
        Δx  = 0.073529
```

Želimo li npr. dobiti sustav s prigušenjem, $\vec{A}_{2,2}$ treba postaviti na negativnu vrijednost.

1.3.2 Zadatak 2

Izračunajte ponašanje sljedećeg sustava:

$$\vec{x_{k+1}} = \begin{bmatrix} 0 & 1 \\ -200 & -102 \end{bmatrix} \vec{x_k} \quad \vec{x_0} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Sustav predstavlja fizikalno njihalo s prigušenjem (zadatak s predavanja). Isprobajte rješavanje s periodom integracije $T = 0.1$ i $t_{MAX} = 1$ za sve zadane postupke i obratite pažnju na numeričku stabilnost (uz zadane početne uvjete)!

Odgovor

[13]:
```
t2_step = 0.1
t2_interval =(0, 1)

t2_a = np.array([
    [0, 1],
    [-200, -102],
])
t2_b = np.array([
    [0, 0],
    [0, 0],
])

t2_initial_state = np.array([
    [1],
    [-2],
])

t2_steps_to_print = 1
```

[14]:
```
t2_integrators = (
    EulerIntegrator(),
    InverseEulerIntegrator(),
    TrapezoidalIntegrator(),
    RungeKutta4Integrator(),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=InverseEulerIntegrator(),
    ),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=TrapezoidalIntegrator(),
    ),
)

t2_results = dict()
```

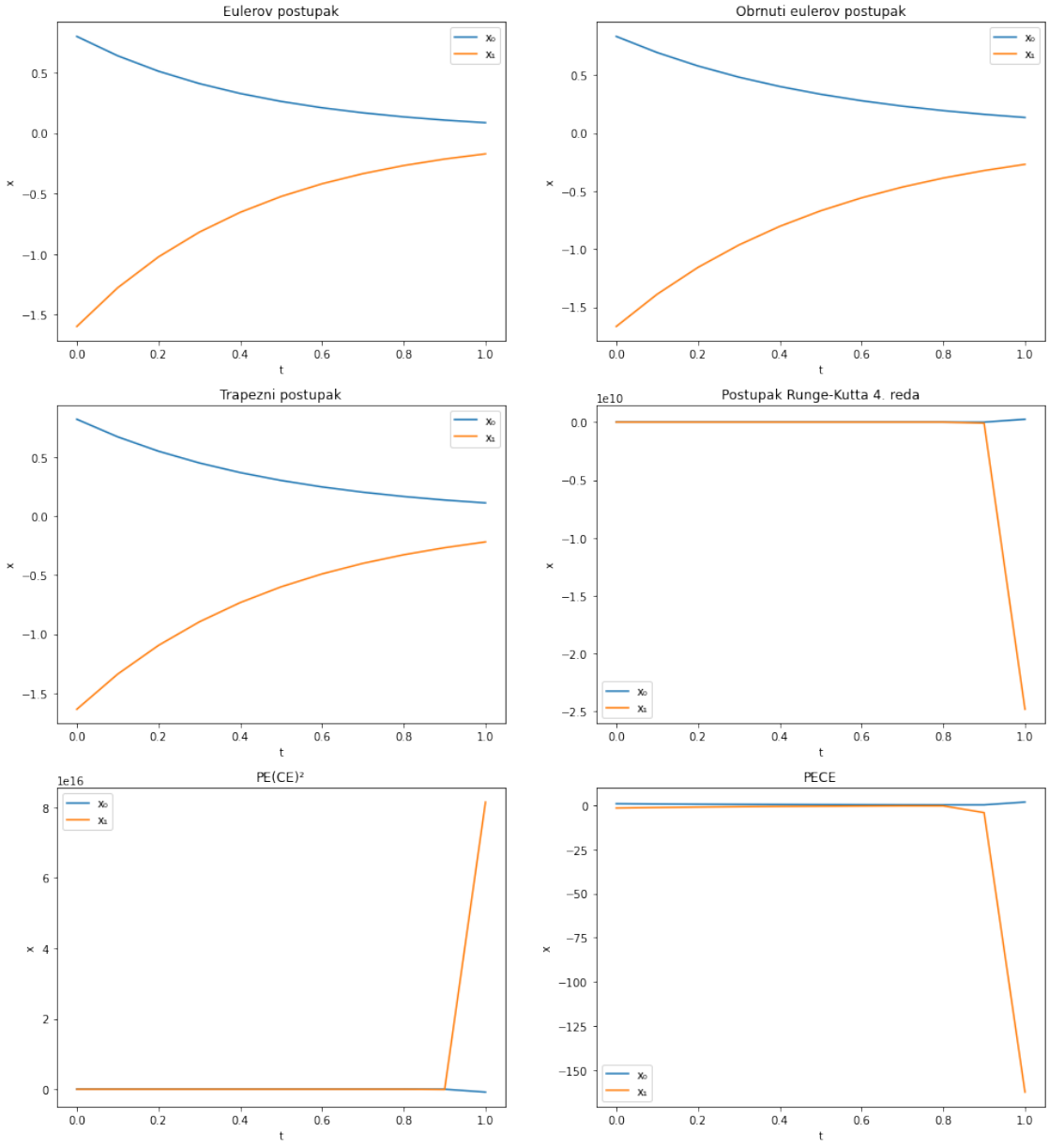
[15]:
```
for integrator in t2_integrators[:-2]:
    t2_results[integrator.name] = integrator(
        step=t2_step,
        interval=t2_interval,
        initial_state=t2_initial_state,
        time_function=None,
        steps_to_print=t2_steps_to_print,
        a=t2_a,
        b=t2_b
    )
```

```
x_10(t = 1.0) = [[ 0.107 -0.215]]: : 11it [00:00, 676.92it/s]
x_10(t = 1.0) = [[ 0.162 -0.323]]: : 11it [00:00, 1272.33it/s]
x_10(t = 1.0) = [[ 0.134 -0.269]]: : 11it [00:00, 870.81it/s]
x_10(t = 1.0) = [[   852109.551 -85210941.861]]: : 11it [00:00, 810.32it/s]
```

[16]:
```
for integrator, n_corrector_repeats in zip(t2_integrators[-2:], (2, 1)):
    t2_results[integrator.name] = integrator(
        step=t2_step,
        interval=t2_interval,
        initial_state=t2_initial_state,
        time_function=None,
        steps_to_print=t2_steps_to_print,
        a=t2_a,
        b=t2_b,
        n_corrector_repeats=n_corrector_repeats
    )
```

```
x_10(t = 1.0) = [[ 8.961e+11 -8.961e+13]]: : 11it [00:00, 886.51it/s]
x_10(t = 1.0) = [[ 0.177 -4.232]]: : 11it [00:00, 1193.66it/s]
```

[17]:
```
t2_fig, t2_ax = get_state_over_time_graphs(t2_results)
```



Komentar

Vidimo da su ovdje stabilni bili jedino Eulerov, obrnuti Eulerov i trapezni postupak.

Usporedbom rezultata odredite prikladni korak integracije za Runge-Kutta postupak.

Odgovor

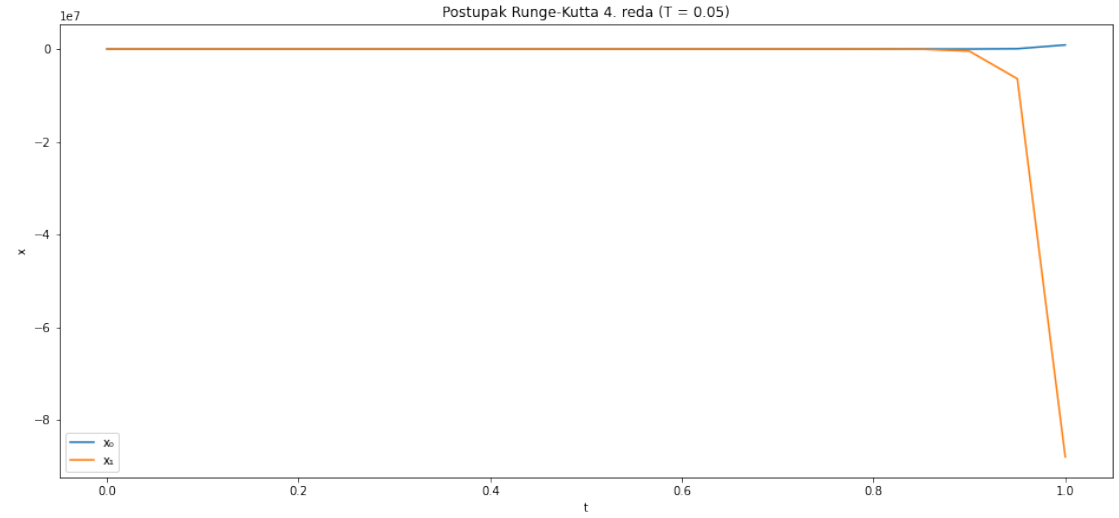
Čini se da je potreban nešto finiji korak integracije.

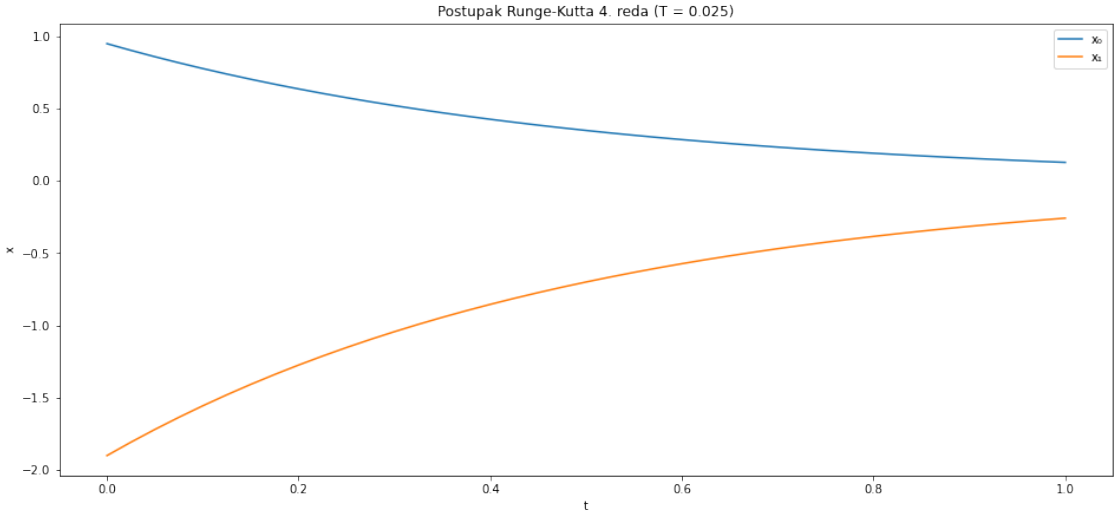
[18]:
```
t2_step_candidates = [(0.1 / (2 ** x)) for x in range(1, 4)]

t2_rk_integrator = [x for x in t2_integrators if x.name ==␣
↪"RungeKutta4Integrator"][0]
t2_rk_results = dict()
```

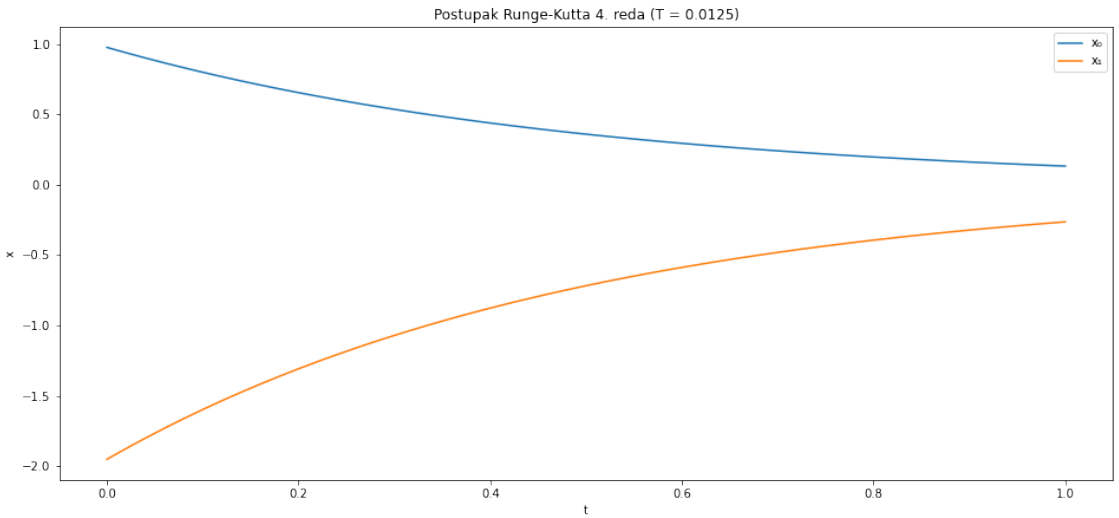
[19]:
```
for step in t2_step_candidates:
    t2_rk_results[step] = t2_rk_integrator(
        step=step,
        interval=t2_interval,
        initial_state=t2_initial_state,
        time_function=None,
        steps_to_print=t2_steps_to_print,
        a=t2_a,
        b=t2_b
    )
```

```
x_20(t = 1.0) = [[   64137.261 -6413712.818]]: : 21it [00:00, 967.09it/s]
x_40(t = 1.0) = [[ 0.135 -0.271]]: : 41it [00:00, 644.24it/s]
x_80(t = 1.0) = [[ 0.135 -0.271]]: : 81it [00:00, 762.39it/s]
```

[20]:
```
t2_fig_rk, t2_ax_rk = plt.subplots(
    len(t2_step_candidates),
    1,
    figsize=(16, len(t2_step_candidates * 8))
)

for i, (step, results) in enumerate(t2_rk_results.items()):
    x_axis = [x["t"] for x in results["states"]]
    y_axis = np.array([x["x"] for x in results["states"]])
    y_axis = y_axis.reshape(len(y_axis), -1).T

    t2_ax_rk[i].set_title(f"Postupak Runge-Kutta 4. reda (T = {step})")
    t2_ax_rk[i].set_xlabel("t")
    t2_ax_rk[i].set_ylabel("x")

    for index, y in enumerate(y_axis):
        t2_ax_rk[i].plot(x_axis, y, label=f"x{str(index).translate(SUB)}")

    t2_ax_rk[i].legend()
```


Postupak Runge-Kutta 4. reda (T = 0.05)
1e7
0
−2
−4
−6
−8
x
x₀
x₁
0.0
0.2
0.4
0.6
0.8
1.0
t


Postupak Runge-Kutta 4. reda (T = 0.025)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x
x₀
x₁
0.0
0.2
0.4
0.6
0.8
1.0
t


Postupak Runge-Kutta 4. reda (T = 0.0125)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x
x₀
x₁
0.0
0.2
0.4
0.6
0.8
1.0
t

Komentar

Vidimo da smo postigli stabilnost već za $T = 0.025$.

Odgovor

1.3.3 Zadatak 3

Izračunajte ponašanje sljedećeg sustava:

$$\vec{x_{k+1}} = \begin{bmatrix} 0 & -2 \\ 1 & -3 \end{bmatrix} \vec{x_k} + \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \vec{x_0} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Isprobajte rješavanje s periodom integracije $T = 0.01$ i $t_{MAX} = 10$ za sve zadane postupke.

Odgovor

[21]:
```
t3_step = 0.01
t3_interval =(0, 10)

t3_a = np.array([
    [0, -2],
    [1, -3],
])
t3_b = np.array([
    [2, 0],
    [0, 3],
])
t3_time_function = lambda t: np.array([
    [1],
    [1],
])

t3_initial_state = np.array([
    [1],
    [3],
])

t3_steps_to_print = 100
```

[22]:
```
t3_integrators = (
    EulerIntegrator(),
    InverseEulerIntegrator(),
    TrapezoidalIntegrator(),
    RungeKutta4Integrator(),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=InverseEulerIntegrator(),
    ),
```

```
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=TrapezoidalIntegrator(),
    ),
)

t3_results = dict()
```

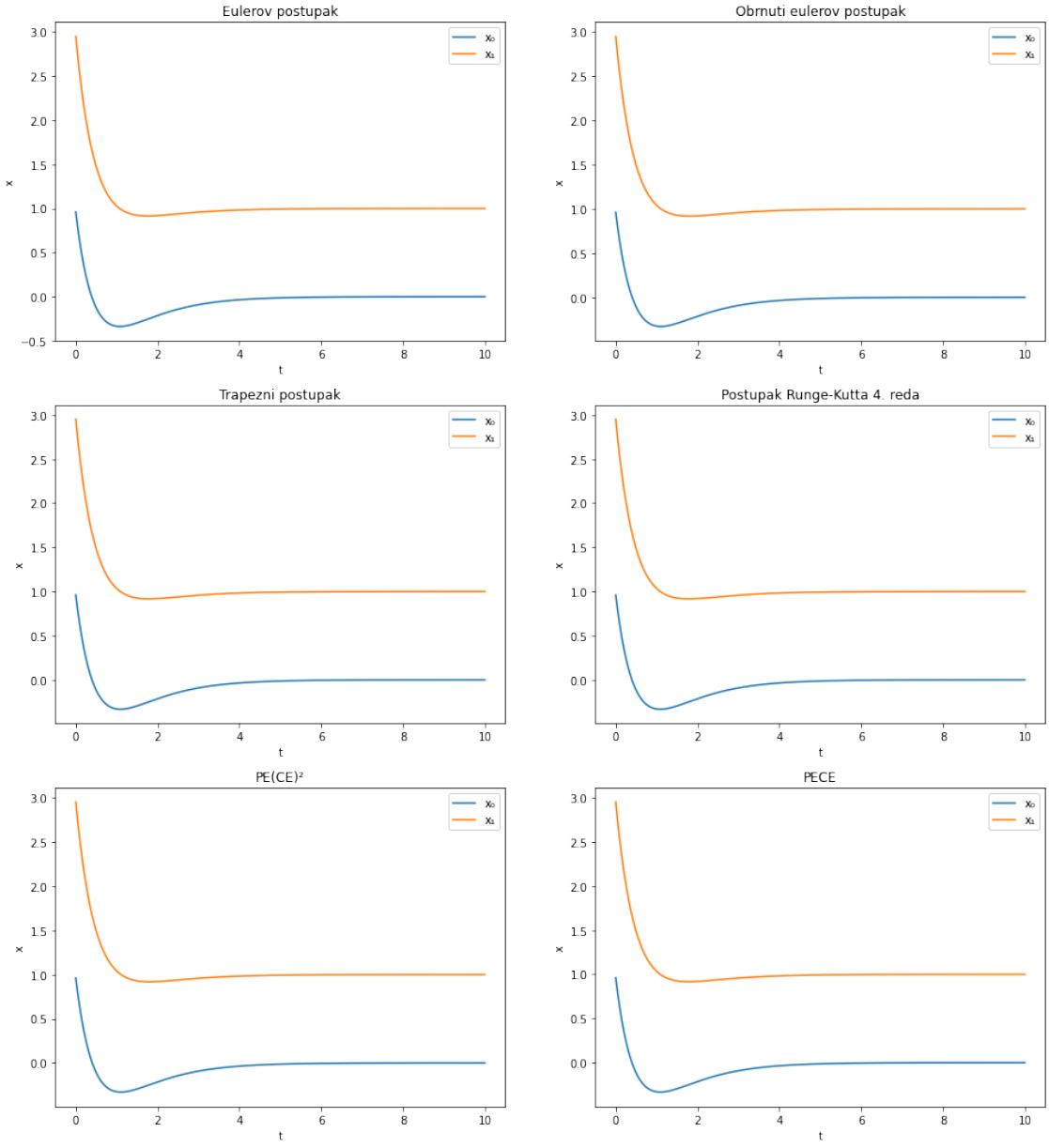
[23]:
```
for integrator in t3_integrators[:-2]:
    t3_results[integrator.name] = integrator(
        step=t3_step,
        interval=t3_interval,
        initial_state=t3_initial_state,
        time_function=t3_time_function,
        steps_to_print=t3_steps_to_print,
        a=t3_a,
        b=t3_b
    )
```

```
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 12600.72it/s]
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 16641.42it/s]
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 18499.74it/s]
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 8098.85it/s]
```

[24]:
```
for integrator, n_corrector_repeats in zip(t3_integrators[-2:], (2, 1)):
    t3_results[integrator.name] = integrator(
        step=t3_step,
        interval=t3_interval,
        initial_state=t3_initial_state,
        time_function=t3_time_function,
        steps_to_print=t3_steps_to_print,
        a=t3_a,
        b=t3_b,
        n_corrector_repeats=n_corrector_repeats
    )
```

```
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 10923.15it/s]
x_1000(t = 10.0) = [[-0.  1.]]: : 1001it [00:00, 12259.82it/s]
```

[25]:
```
t3_fig, t3_ax = get_state_over_time_graphs(t3_results)
```



1.3.4 Zadatak 4

Izračunajte ponašanje sljedećeg sustava:

$$\vec{x_{k+1}} = \begin{bmatrix} 1 & -5 \\ 1 & -7 \end{bmatrix} \vec{x_k} + \begin{bmatrix} 5 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} t \\ t \end{bmatrix} \quad \vec{x_0} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

Isprobajte rješavanje s periodom integracije $T = 0.01$ i $t_{MAX} = 10$ za sve zadane postupke.

Odgovor

[26]:
```
t4_step = 0.01
t4_interval =(0, 1)

t4_a = np.array([
    [1, -5],
    [1, -7],
])
t4_b = np.array([
    [5, 0],
    [0, 3],
])
t4_time_function = lambda t: np.array([
    [t],
    [t],
])

t4_initial_state = np.array([
    [-1],
    [3],
])

t4_steps_to_print = 100
```

[27]:
```
t4_integrators = (
    EulerIntegrator(),
    InverseEulerIntegrator(),
    TrapezoidalIntegrator(),
    RungeKutta4Integrator(),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=InverseEulerIntegrator(),
    ),
    PredictorCorrectorIntegrator(
        predictor=EulerIntegrator(),
        corrector=TrapezoidalIntegrator(),
    ),
)

t4_results = dict()
```

[28]:
```
for integrator in t4_integrators[:-2]:
    t4_results[integrator.name] = integrator(
        step=t4_step,
        interval=t4_interval,
        initial_state=t4_initial_state,
        time_function=t4_time_function,
        steps_to_print=t4_steps_to_print,
```

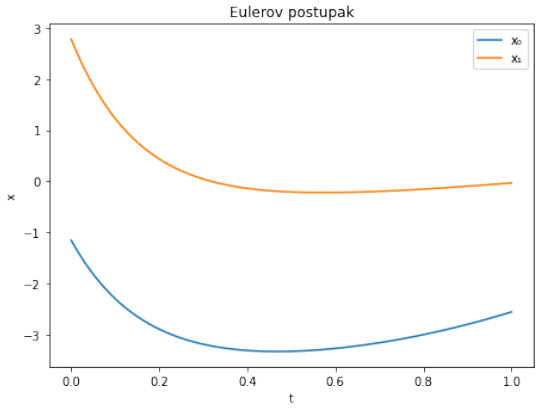
```
        a=t4_a,
        b=t4_b
    )
```

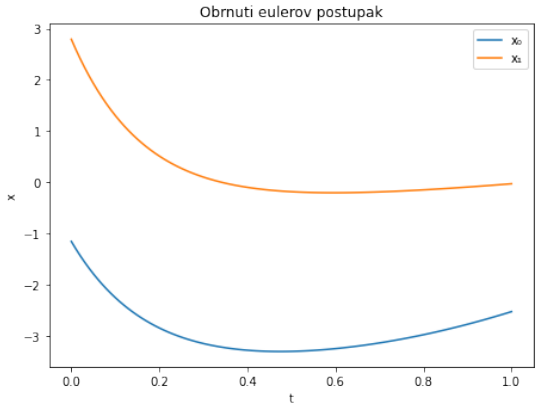
```
x_100(t = 1.0) = [[-2.588 -0.04 ]]: : 101it [00:00, 19094.24it/s]
x_100(t = 1.0) = [[-2.546 -0.032]]: : 101it [00:00, 11270.81it/s]
x_100(t = 1.0) = [[-2.567 -0.036]]: : 101it [00:00, 11787.00it/s]
x_100(t = 1.0) = [[-2.567 -0.036]]: : 101it [00:00, 7124.89it/s]
```

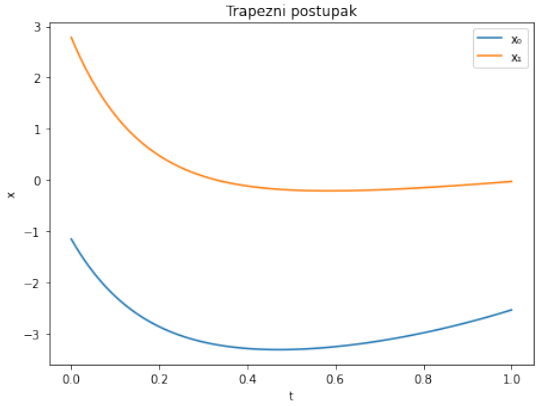
[29]:
```
for integrator, n_corrector_repeats in zip(t4_integrators[-2:], (2, 1)):
    t4_results[integrator.name] = integrator(
        step=t4_step,
        interval=t4_interval,
        initial_state=t4_initial_state,
        time_function=t4_time_function,
        steps_to_print=t4_steps_to_print,
        a=t4_a,
        b=t4_b,
        n_corrector_repeats=n_corrector_repeats
    )
```

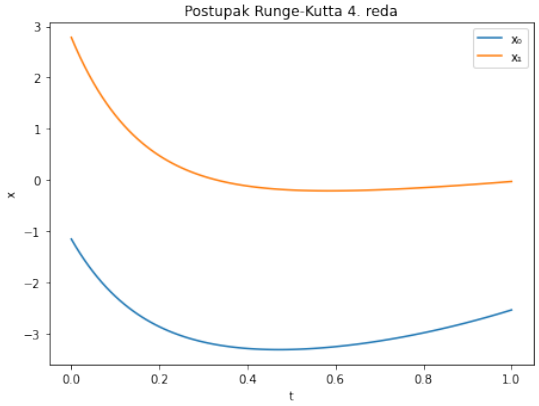
```
x_100(t = 1.0) = [[-2.588 -0.042]]: : 101it [00:00, 10347.71it/s]
x_100(t = 1.0) = [[-2.567 -0.036]]: : 101it [00:00, 11254.04it/s]
```

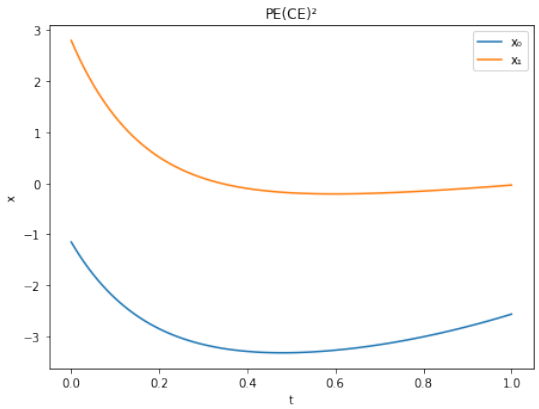
[30]:
```
t4_fig, t4_ax = get_state_over_time_graphs(t4_results)
```

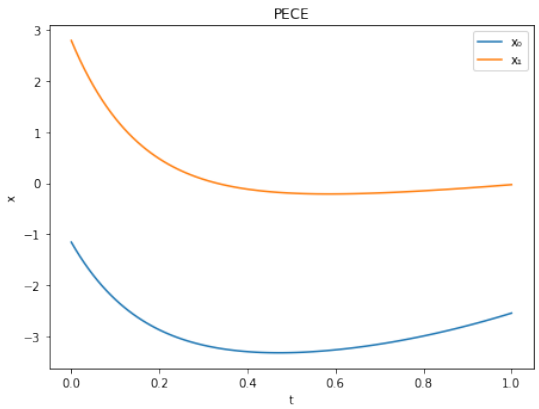

Eulerov postupak
x₀
x₁
x
t


Obrnuti eulerov postupak
x₀
x₁
x
t


Trapezni postupak
x₀
x₁
x
t


Postupak Runge-Kutta 4. reda
x₀
x₁
x
t


PE(CE)²
x₀
x₁
x
t


PECE
x₀
x₁
x
t