trabalho_05

August 31, 2022

1 Trabalho 5 - Controle multivariável

Disciplina:

- TE975 - Controle Avançado

Autores:

- Ana Paula da Silva Pelegrini GRR20177221
- Caio Phillipe Mizerkowski GRR20166403

1.0.1 Bibliotecas

As seguintes bibliotecas foram utilizadas, sendo o JADE um algoritmo construído durante a disciplina de Artificial e Aprendizagem de Máquina ministrada no semestre passado pelo professor Leandro Coelho e cujos códigos estão presentes no repositório AIML e os resultados presentes em Evolução diferencial vs algoritmo genetico vs JADE.

[ ]:
```
import numpy as np
from scipy import optimize
import sympy as sp
from jade import JADifferentialEvolution
from pprint import pprint
```

1.0.2 Sistema WoodBerry

O sistema a ser controlado foi implementado em uma classe, composta de todo o necessário para o funcionamento do mesmo.

[ ]:
```
class WoodBerry:
    def __init__(self):
        self.s = sp.symbols("s")
        self.t = sp.symbols("t", positive=True)
        self.N = 200
        self.Sn = self.N**2 / 2

        self.bounds = np.array(
            [
                [-0.2, 0.2],
            ]
```

```
            * 4
        )

        self.active_constrains = False

        self.G = sp.Array(
            [
                [
                    (12.8 * self.pade_exp(-1)) / (16.7 * self.s + 1),
                    (-18.9 * self.pade_exp(-3)) / (21 * self.s + 1),
                ],
                [
                    (6.6 * self.pade_exp(-7)) / (10.9 * self.s + 1),
                    (-19.4 * self.pade_exp(-3)) / (14.4 * self.s + 1),
                ],
            ]
        )

    def pade_exp(self, n):
        """Aproximação de pade para a exponencial"""
        return (1 + 0.5 * n * self.s) / (1 - 0.5 * n * self.s)

    def wood_berry(self, u):
        y = self.G @ u
        return y

    def funcao_objetivo(self, erro):
        """Função ITAE normalizada para facilitar a comparação entre diferentes␣
↪quantidade de amostras"""
        f = (
            sum(k * (np.abs(e1) + np.abs(e2)) for k, (e1, e2) in␣
↪enumerate(erro))
            / self.Sn
        )

        return f

    def inverse_laplace(self, sis):
        """Processo para separar a equação simbolica em frações parciais e␣
↪realizar a inversa de laplace do sistema"""
        sis = sp.N(sp.polys.partfrac.apart(sis, self.s, full=True).doit(), 5)
        for sisf in sis.args:
            v1 = 0
            v2 = 0
            if len(sisf.args):
                v1 = sisf.args[0]
                sisf2 = 1 / sisf.args[1]
```

```
                if len(sisf2.args):
                    v2 = sisf2.args[0]
            yield v1 * sp.exp(-v2 * self.t)

    def controlador(self, c, erro):
        """Controlador PI, feito como PI para reduzir o espaço de buscar em␣
↪razão da demora do processamento"""
        u1 = (c[0] + c[1] * self.s) * erro[0]
        u2 = (c[2] + c[3] * self.s) * erro[1]
        return np.array([u1, u2]).T

    def sistema(self, c, erro):
        u = self.controlador(c, erro)
        y = self.wood_berry(u)
        return [sum(self.inverse_laplace(y0)) for y0 in y]

    def evaluate_sistem(self, x):
        """Função que realiza o processamente e retorna o erro"""
        erro = np.array([1, 1])
        sis = self.sistema(x, erro)
        erros = []
        for k in range(self.N):
            val0 = sis[0] if isinstance(sis[0], int) else sis[0].
↪evalf(subs={self.t: k})
            val1 = sis[1] if isinstance(sis[1], int) else sis[1].
↪evalf(subs={self.t: k})
            erro = np.array(
                [
                    float(val0) - 0.96,
                    float(val1) - 0.05,
                ]
            )
            sis = self.sistema(x, erro)
            erros.append(erro)
        itae = self.funcao_objetivo(erros)
        print(x)
        print(itae)
        # input()
        print()
        return itae

    def run_sistem(self, x):
        """Função que realiza o processamente e retorna o erro"""
        erro = np.array([1, 1])
        sis = self.sistema(x, erro)

        for k in range(self.N):
```

```
            val0 = sis[0] if isinstance(sis[0], int) else sis[0].
↪evalf(subs={self.t: k})
            val1 = sis[1] if isinstance(sis[1], int) else sis[1].
↪evalf(subs={self.t: k})
            erro = np.array(
                [
                    float(val0) - 0.96,
                    float(val1) - 0.05,
                ]
            )
            sis = self.sistema(x, erro)
            yield erro

    def problem(self, x):
        """Função a ser chamada pelo JADE"""
        return self.evaluate_sistem(x)
```

Caso já exista uma lista de coeficientes para ser iniciada no algoritmo, em razão de um processamento anterior que foi interrompido, ela pode ser carregada neste script.

[ ]:
```
with open("./trabalho_05_jade.txt") as f:
    results = f.read()
results = list(set(results.split("\n\n")))
results = [res.splitlines() for res in results]
results = [
    (
        list(
            map(
                float,
                list(
                    pesos.replace("[", "")
                    .replace("]", "")
                    .strip()
                    .replace("   ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .split(" ")
                ),
            )
        ),
        float(valor),
    )
    for pesos, valor in results
    if float(valor) < 3
]
```

```
X0 = np.concatenate(
    [
        np.array([pesos for pesos, valor in results]),
        np.random.uniform(low=-0.2, high=0.2, size=(60 - len(results), 4)),
    ]
)
```

1.0.3 Controle Descentralizado

[ ]:
```
N = 1
G = 100
config = {"case1": {"mutation": 0.6, "recombination": 0.8}}


def run(algorithm, problem, N, G, config):
    res = dict()
    for case in config:
        res[case] = []
        for i in range(N):
            alg = algorithm(
                problem.problem, problem.bounds, seed=i, G=G, **config[case]
            )
            alg.run()
            res[case].append(alg)
    return res

wb = WoodBerry()
res_jade = run(algorithm=JADifferentialEvolution, problem=wb, N=N, G=G,␣
 ↪config=config)
```

1.0.4 Controle Centralizado

Criando subclasse com o controlador modificado para ser um controlador centralizado

[ ]:
```
class WoodBerry2(WoodBerry):
    def __init__(self):
        super().__init__()
        self.bounds = np.array(
            [
                [-0.2, 0.2],
            ]
            * 6
        )

    def controlador(self, c, erro):
```

```
        """Controlador PI, feito como PI para reduzir o espaço de buscar em␣
↪razão da demora do processamento."""
        u1 = (c[0] + c[1] * self.s) * erro[0] + c[4] * erro[1]
        u2 = (c[2] + c[3] * self.s) * erro[1] + c[5] * erro[0]
        return np.array([u1, u2]).T
```

Adicionando os dois coeficientes extras necessários Kp12 e Kp21 e iniciando-os numa distribuição normal em torno do zero, usando os valores da processo anterior como chute inicial.

[ ]:
```
with open("./trabalho_05_jade.txt") as f:
    results = f.read()
results = list(set(results.split("\n\n")))
results = [res.splitlines() for res in results]
results = [
    (
        list(
            map(
                float,
                list(
                    pesos.replace("[", "")
                    .replace("]", "")
                    .strip()
                    .replace("   ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .replace("  ", " ")
                    .split(" ")
                )
                + [np.random.normal(0, 0.01), np.random.normal(0, 0.01)],
            )
        ),
        float(valor),
    )
    for pesos, valor in results
    if float(valor) < 3
]

X0 = np.concatenate(
    [
        np.array([pesos for pesos, valor in results]),
        np.random.uniform(low=-0.2, high=0.2, size=(60 - len(results), 6)),
    ]
)
```

[ ]:
```
N = 1
G = 100
```

```
config = {"case1": {"mutation": 0.6, "recombination": 0.8}}


def run(algorithm, problem, N, G, config):
    res = dict()
    for case in config:
        res[case] = []
        for i in range(N):
            alg = algorithm(
                problem.problem, problem.bounds, seed=i, G=G, **config[case]
            )
            alg.run()
            res[case].append(alg)
    return res


wb2 = WoodBerry2()
res_jade = run(algorithm=JADifferentialEvolution, problem=wb2, N=N, G=G,␣
 ↪config=config)
```

1.0.5 Resultados

Após a implementação das rotinas, não conseguimos os resultados esperados. O motivo provavel é algum erro na implementação do sistema Wood Berry que não fomos capazes de identificar em tempo hábil para a entrega dos resultados.