دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش ۹

پیاده‌سازی یک سیستم فازی

نام: مریم علیپور حاجی آقا

شماره دانشجویی:۹۶۱۲۰۳۷

استاد: دکتر قطعی

سامانه کنترل فازی:

سامانه کنترل فازی یک سامانه کنترل مبتنی بر منطق فازی است - سامانه‌ای ریاضی که مقادیر ورودی آنالوگ را به عنوان متغیرهای منطقی تحلیل می‌کند، یعنی برروی مقادیری پیوسته در بازه بین ۰ و ۱. برخلاف منطق کلاسیک یا دیجیتال، که بر روی مقادیر گسسته ۱ یا ۰ (درست یا نادرست) عمل می‌کند.

منطق فازی به طور گسترده در سامانه‌های کنترلی استفاده می‌شود. اصطلاح "فازی" یا مبهم به این واقعیت اشاره دارد که منطق حاکم بر سامانه می‌تواند با مفاهیمی برخورد کند که نمی‌توانند به عنوان "درست" یا "نادرست" بیان شوند، بلکه به عنوان "تا حدی درست" بیان می‌شوند.

اگر چه رویکردهای دیگری مانند الگوریتم ژنتیک و شبکه‌های عصبی می‌توانند در بسیاری از موارد مانند منطق فازی عمل کنند، اما منطق فازی این مزیت را دارد که راه حل مسئله را می‌توان به گونه‌ای که اپراتورهای انسانی آن را درک کرده‌اند، ارائه و پیاده‌سازی کرد. به این ترتیب از تجربه آن‌ها برای طراحی کنترل کننده، بدون داشتن مدل ریاضی دقیقی برای سامانه، استفاده نمود. این خصوصیت باعث می‌شود تا مکانیزه کردن کارهایی که قبلاً توسط انسان با موفقیت انجام شده‌اند، آسان‌تر شود.

کاربردها:

1. علاقه به سامانه‌های فازی توسط «سیجی یاسونوبو» و «سوجی میاموتو» از هیتاچی آغاز شد، زمانی که در سال ۱۹۸۵ شبیه سازی‌هایی را ارائه دادند که امکان سنجی سامانه‌های کنترل فازی برای متروی سندای را نشان می‌داد. ایده آن‌ها به تصویب رسید و سامانه‌های فازی برای کنترل شتاب، ترمز و توقف قطار شهری، همزمان با افتتاح خط نامبوک در ۱۹۸۷ استفاده شد.
2. در سال ۱۹۸۷، «تاکشی یاماکاوا» در آزمایش «آونگ واژگون» استفاده از کنترل فازی را از طریق مجموعه‌ای از تراشه‌های اختصاصی و ساده منطقی فازی نشان داد. این یک مسئله کلاسیک کنترل است، که در آن وسیله نقلیه سعی می‌کند با حرکت به جلو و عقب، آونگ واژگون را به صورت باثبات نگه دارد. در ادامه یاماکاوا با نصب یک لیوان حاوی آب و حتی قرار دادن یک موش زنده در بالای آونگ، این مسئله را پیچیده‌تر کرد و این سامانه با موفقیت در هر دو مورد ثبات خود را حفظ کرد. سرانجام یاماکاوا به منظور بهره‌برداری از حق ثبت اختراعات خود در این زمینه، اقدام به تاسیس آزمایشگاه تحقیقات سامانه‌های فازی خود نمود.
3. مهندسان ژاپنی متعاقباً طیف گسترده‌ای از سامانه‌های فازی را برای کاربردهای صنعتی و خانگی توسعه دادند. در سال ۱۹۸۸، ژاپن آزمایشگاه مهندسی بین المللی فازی (LIFE)، که یک همکاری مشترک بین ۴۸ شرکت برای پیگیری تحقیقات فازی است، را تأسیس کرد. شرکت خودروسازی فولکس واگن تنها عضو خارجی در این توافق بود که یک محقق را برای مدت سه سال به آن اعزام کرد.
4. بسیاری از کالاهای مصرفی ژاپنی دارای سامانه‌های فازی هستند. جاروبرقی ماتسوشیتا از میکروکنترلرهای الگوریتم فازی برای خواندن اطلاعات از سنسورهای گرد و غبار استفاده می‌کند و بر این اساس قدرت مکش را تنظیم می‌کند. ماشین لباسشویی هیتاچی از کنترلرهای فازی برای حسگرهای وزن، نوع پارچه و چرک استفاده کرده و به طور خودکار چرخه یا سیکل شستشو را برای استفاده بهینه از انرژی، آب و مواد شوینده تنظیم می‌کنند.

معماری سیستم:

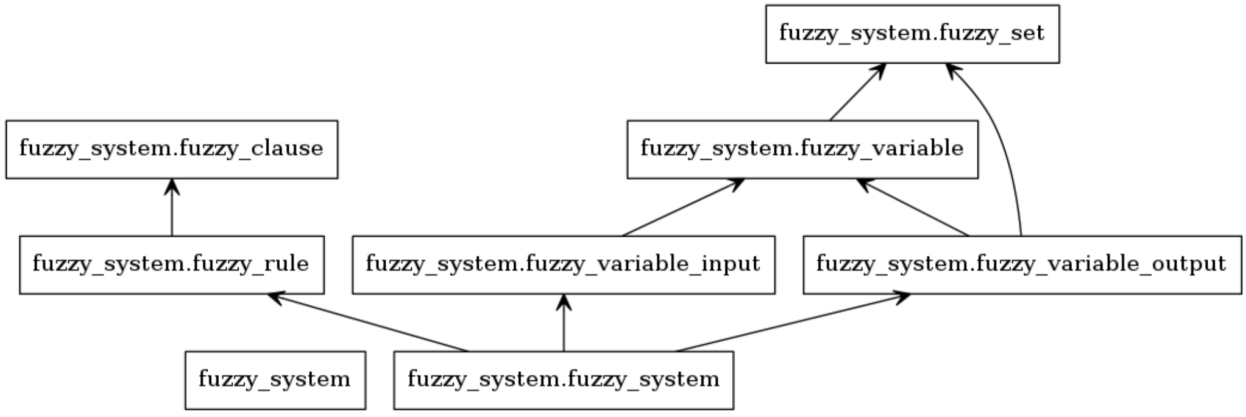
نمودار زیر ساختار برنامه را نشان می دهد. این طرح بر اساس چندین ملاحظه در سیستم های استنتاج فازی ساخته شده است که برخی از این موارد عبارتند از:

1. یک سیستم استنتاج فازی به متغیرهای ورودی و خروجی و مجموعه ای از قوانین فازی نیاز دارد.
2. اگر سیستم استنتاج فازی از نوع ممدانی باشد ، هر دو متغیر ورودی و خروجی شامل مجموعه ای از مجموعه های فازی خواهند بود.
3. متغیرهای ورودی و خروجی بسیار مشابه هستند ، اما با قوانین فازی از آنها متفاوت استفاده می شود. در حین اجرا ، متغیرهای ورودی از مقادیر ورودی به سیستم برای فازی کردن مجموعه های خود استفاده می کنند ، یعنی آنها درجه تعلق آن مقدار ورودی را به همه مجموعه های فازی متغیر تعیین می کنند. هر قانون تا حدودی به متغیرهای خروجی کمک می کند. مجموع این سهم خروجی سیستم را تعیین می کند.
4. قوانین فازی فرم زیر را دارند.

```
if {antecedent clauses} then {consequent clauses}
```

بنابراین یک قانون شامل چندین بند از نوع پیشین و برخی بندها از نوع متعلقه خواهد بود. بندها به شکل زیر خواهد بود:

```
{variable name} is {set name}
```



ما در بخشهای زیر درباره جزئیات اجرای کلاسهای توسعه یافته برای این سیستم بحث خواهیم کرد:

FuzzySet class

- name: نام مجموعه
- Minimum value: حداقل مقدار مجموعه

- maximum value: حداکثر مقدار مجموعه
- resolution: تعداد مراحل بین حداقل و حداکثر مقدار

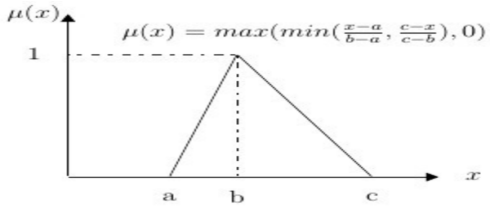
بنابراین، می‌توان یک مجموعه فازی را با استفاده از دو آرایه numpy نشان داد. یکی که مقادیر دامنه را نگه دارد و دیگری که مقادیر درجه عضویت را در خود داشته باشد. در ابتدا، تمام مقادیر درجه عضویت روی صفر تنظیم می‌شوند. می‌توان ادعا کرد که اگر حداقل و حداکثر مقادیر همراه با وضوح مجموعه موجود باشد، آرایه numpy دامنه مورد نیاز نیست زیرا مقادیر مربوطه را می‌توان محاسبه کرد. گرچه این کاملا درست است، اما یک آرایه دامنه در این نمونه پروژه ترجیح داده شده است تا کد خواناتر و ساده تر باشد.

```
def create_triangular(cls, name, domain_min, domain_max, res, a, b, c):
  t1fs = cls(name, domain_min, domain_max, res)

  a = t1fs._adjust_domain_val(a)
  b = t1fs._adjust_domain_val(b)
  c = t1fs._adjust_domain_val(c)

  t1fs._dom = np.round(np.maximum(np.minimum((t1fs._domain-a)/(b-a),
(c-t1fs._domain)/(c-b)), 0), t1fs._precision)
```

در متن یک متغیر فازی، همه مجموعه‌ها از حداقل، حداکثر و تفکیک‌پذیری یکسانی برخوردار خواهند بود.
از آنجا که با یک دامنه گسسته سروکار داریم، لازم است هر مقداری برای تنظیم یا بازیابی درجه عضویت به نزدیکترین مقدار در آرایه دامنه استفاده شود.

```
def _adjust_domain_val(self, x_val):
  return self._domain[np.abs(self._domain-x_val).argmin()]
```

کلاس شامل روش‌هایی است که به موجب آن می‌توان مجموعه‌ای از یک شکل داده شده را با توجه به تعداد متغیر پارامترها ساخت. به عنوان مثال، در مورد یک مجموعه مثلثی، سه پارامتر ارائه شده است، دو پارامتر که مجموعه‌های مجموعه را تعریف می‌کند و دیگری برای راس. با استفاده از این سه پارامتر می‌توان مجموعه ای مثلثی ساخت که در شکل زیر مشاهده می‌شود.



Triangular Fuzzy Set equation

از آنجا که مجموعه‌ها براساس آرایه‌های numpy بنا شده‌اند، معادله فوق می‌تواند مستقیماً به کد ترجمه شود، همانطور که در زیر مشاهده می‌شود. مجموعه‌هایی با اشکال مختلف را می‌توان با استفاده از روشی مشابه ساخت.

```
def create_triangular(cls, name, domain_min, domain_max, res, a, b, c):
  t1fs = cls(name, domain_min, domain_max, res)

  a = t1fs._adjust_domain_val(a)
  b = t1fs._adjust_domain_val(b)
  c = t1fs._adjust_domain_val(c)

  t1fs._dom = np.round(np.maximum(np.minimum((t1fs._domain-a)/(b-a),
(c-t1fs._domain)/(c-b)), 0), t1fs._precision)
```

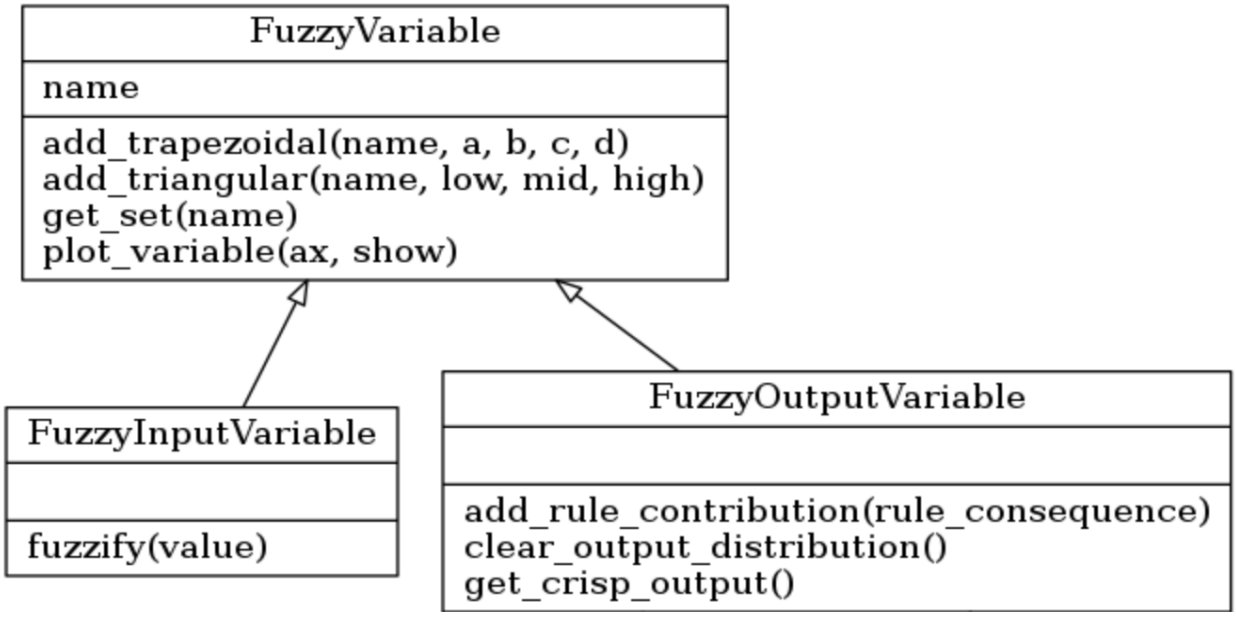
کلاس FuzzySet همچنین شامل عملگرهای اتحادیه، تقاطع و نفی است که برای انجام استنباط لازم است. همه روش‌های عملگر با نتیجه عملیاتی که انجام شده است، یک مجموعه فازی جدید را برمی‌گردانند.

```
def union(self, f_set):

        result = FuzzySet(f'({self._name}) union ({f_set._name})',
self._domain_min, self._domain_max, self._res)
        result._dom = np.maximum(self._dom, f_set._dom)

        return result
```

سرانجام، ما توانایی بدست آوردن نتیجه واضح از مجموعه فازی را با استفاده از روش مرکز ثقل که در مقاله قبلی با جزئیات به آن اشاره شده است، پیاده سازی کردیم. ذکر این نکته مهم است که تعداد زیادی روش فازی سازی در ادبیات موجود است. هنوز هم، چون روش مرکز ثقل بسیار محبوب است، در این اجرا استفاده می‌شود.

```
def cog_defuzzify(self):

  num = np.sum(np.multiply(self._dom, self._domain))
  den = np.sum(self._dom)

  return num/den
```

Fuzzy Variable classes



FuzzyVariable Classes

همانطور که قبلاً بحث شد، متغیرها می‌توانند از نوع ورودی یا خروجی باشند، تفاوت در محاسبه استنتاج فازی تأثیر دارد.
FuzzyVariable به مجموعه‌ای از مجموعه‌ها گفته می‌شود که در یک دیکشنری پایتون نگهداری می‌شوند و نام مجموعه را به عنوان کلید در اختیار دارند. روش هایی برای افزودن FuzzySets به متغیر در دسترس هستند، جایی که چنین مجموعه‌هایی محدودیت و وضوح متغیر را می‌گیرند.
برای متغیرهای ورودی، fuzzification با بازیابی درجه عضویت همه مجموعه‌های متغیر برای یک مقدار دامنه مشخص انجام می‌شود. درجه عضویت در مجموعه ذخیره می‌شود زیرا در هنگام ارزیابی آنها توسط قوانین الزامی است.

```
def fuzzify(self, val):

    # get dom for each set and store it -
    # it will be required for each rule
    for set_name, f_set in self._sets.items():
        f_set.last_dom_value = f_set[val]
```

متغیرهای خروجی در نهایت نتیجه تکرار استنتاج فازی را ایجاد می‌کنند. این بدان معنی است که برای سیستم‌های نوع ممدانی، همانطور که ما در اینجا در حال ساخت آن هستیم، متغیرهای خروجی اتحادیه سهم فازی را از همه قوانین حفظ می‌کنند و متعاقباً این نتیجه را از بین می‌برند تا یک مقدار واضح را بدست آورند که می تواند در برنامه‌های زندگی واقعی مورد استفاده قرار گیرد. بنابراین، متغیرهای خروجی به یک ویژگی FuzzySet اضافی نیاز دارند که توزیع خروجی

را برای آن متغیر نگه دارد، جایی که سهم حاصل از هر قانون است و با استفاده از عملگر union set اضافه می‌شود.
سپس می‌توان با فراخوانی روش مرکز ثقل برای مجموعه توزیع خروجی، نتیجه رفع فازی را بدست آورد.

```
class FuzzyOutputVariable(FuzzyVariable):

    def __init__(self, name, min_val, max_val, res):
        super().__init__(name, min_val, max_val, res)
        self._output_distribution = FuzzySet(name, min_val, max_val, res)

    def add_rule_contribution(self, rule_consequence):
        self._output_distribution = self._output_distribution.union(rule_consequence)

    def get_crisp_output(self):
        return self._output_distribution.cog_defuzzify()
```

Fuzzy Rules classes

کلاس FuzzyClause به دو ویژگی نیاز دارد. یک متغیر فازی و یک مجموعه فازی به طوری که دستور

```
variable is set
```

می تواند ایجاد شود. از بندها برای اجرای عباراتی استفاده می شود که می توانند با هم زنجیر شوند و قسمتهای قبلی و متعاقب قانون را تشکیل دهند.
FuzzyClause وقتی به عنوان یک جمله پیشین استفاده می شود ، آخرین درجه عضویت مجموعه را برمی گرداند ، که در مرحله fuzzification همانطور که قبلاً مشاهده کردیم محاسبه می شود.
این قانون با استفاده از عملگر min، مقادیر درجه عضویت را از بندهای مختلف پیشین ترکیب می کند و فعال سازی قاعده را بدست می آورد که سپس همراه با بندهای بعدی برای بدست آوردن سهم قانون در متغیرهای خروجی استفاده می شود. این عملیات یک فرآیند دو مرحله ای است:

- مقدار فعال سازی با FuzzySet متعاقباً با استفاده از عملگر min ترکیب می شود که به عنوان آستانه مقادیر درجه عضویت FuzzySet عمل خواهد کرد.
- FuzzySet حاصل با FuzzySets به دست آمده از سایر قوانین با استفاده از عملگر اتحادیه ترکیب می شود و توزیع خروجی آن متغیر را بدست می آورد.

```
def evaluate_antecedent(self):
        return self._set.last_dom_value

def evaluate_consequent(self, activation):
        self._variable.add_rule_contribution(self._set.min_scalar(activation))
```

بنابراین کلاس FuzzyRule به دو ویژگی نیاز دارد:

- لیستی حاوی بندهای پیشین
- لیستی شامل بندهای متعاقب آن

در حین اجرای FuzzyRule، روشی که در بالا توضیح داده شد انجام می‌شود. FuzzyRule همه کارها را با استفاده از FuzzyClauses متناسب با هم هماهنگ می‌کند.

```
def evaluate(self):
        # rule activation initialize to 1 as min operator will be performed
        rule_activation = 1
        # execute all antecedent clauses, keeping the minimum of the returned doms to
determine the activation
        for ante_clause in self._antecedent:
                rule_activation = min(ante_clause.evaluate_antecedent(), rule_activation)

        # execute consequent clauses, each output variable will update its
output_distribution set
        for consequent_clause in self._consequent:
                consequent_clause.evaluate_consequent(rule_activation)
```

کلاس سیستم فازی - جمع آوری همه آنها با هم:

در بالاترین سطح این معماری، ما سیستم FuzzySystem را داریم که کلیه فعالیت‌های بین FuzzyVariables و FuzzyRules را هماهنگ می‌کند. از این رو سیستم شامل متغیرهای ورودی و خروجی است که با استفاده از نام متغیرها به عنوان کلید و لیستی از قوانین، در دیکشنری‌های پایتون ذخیره می‌شوند.
یکی از چالش‌های ارائه شده در این مرحله روشی است که کاربر نهایی برای افزودن قوانین استفاده می‌کند، که باید ایده‌آل جزئیات پیاده‌سازی کلاس‌های FuzzyClause باشد. روشی که اجرا شد شامل ارائه دو فرهنگ لغت پایتون است که شامل بندهای پیشین و متعاقب آن قانون در قالب زیر است.

```
variable name : set name
```

یک روش کاربر پسندتر، ارائه قاعده به عنوان یک رشته و سپس تجزیه آن رشته برای ایجاد قانون است، اما به نظر می‌رسد این یک سربار اضافی غیر ضروری برای یک برنامه نمایش است.

```
def add_rule(self, antecedent_clauses, consequent_clauses):
        '''
        adds a new rule to the system.
        TODO: add checks
        Arguments:
        ----------
        antecedent_clauses -- dict, having the form {variable_name:set_name, ...}
        consequent_clauses -- dict, having the form {variable_name:set_name, ...}
        '''
        # create a new rule
        # new_rule = FuzzyRule(antecedent_clauses, consequent_clauses)
        new_rule = FuzzyRule()

        for var_name, set_name in antecedent_clauses.items():
            # get variable by name
            var = self.get_input_variable(var_name)
            # get set by name
            f_set = var.get_set(set_name)
            # add clause
            new_rule.add_antecedent_clause(var, f_set)

        for var_name, set_name in consequent_clauses.items():
            var = self.get_output_variable(var_name)
            f_set = var.get_set(set_name)
            new_rule.add_consequent_clause(var, f_set)

        # add the new rule
        self._rules.append(new_rule)
```

افزودن قانون جدید به FuzzySystem

با توجه به این ساختار، جایی که مراحل زیر انجام می شود ، می توان با چند خط کد، اجرای فرایند استنتاج را به دست آورد.

1. مجموعه‌های توزیع خروجی تمام متغیرهای خروجی پاک می‌شوند.

2. مقادیر ورودی به سیستم به متغیرهای ورودی مربوطه منتقل می‌شود تا هر مجموعه در متغیر بتواند درجه عضویت آن را برای آن مقدار ورودی تعیین کند.
3. اجرای قوانین فازی انجام می‌شود، به این معنی که مجموعه‌های توزیع خروجی تمام متغیرهای خروجی اکنون حاوی اتحادیه مشارکت‌های هر قانون هستند.
4. مجموعه‌های توزیع خروجی برای به دست آوردن نتیجه واضح با استفاده از یک دستگاه دفعی مرکز گرانش از فازی خارج می‌شوند.

```
# clear the fuzzy consequences as we are evaluating a new set of inputs.
  # can be optimized by comparing if the inputs have changes from the previous
  # iteration.
  self._clear_output_distributions()

  # Fuzzify the inputs. The degree of membership will be stored in
  # each set
  for input_name, input_value in input_values.items():
    self._input_variables[input_name].fuzzify(input_value)

  # evaluate rules
  for rule in self._rules:
    rule.evaluate()

  # finally, defuzzify all output distributions to get the crisp outputs
  output = {}
  for output_var_name, output_var in self._output_variables.items():
    output[output_var_name] = output_var.get_crisp_output()

  return output
```

به عنوان نکته آخر ، سیستم استنباط فازی که در اینجا پیاده سازی شده است شامل توابع اضافی برای ترسیم مجموعه ها و متغیرهای فازی و به دست آوردن اطلاعات در مورد اجرای مرحله استنتاج است.

مثالی از استفاده از کتابخانه:

در این بخش، ما به استفاده از سیستم استنتاج فازی خواهیم پرداخت. به طور خاص، ما مطالعه موردی سرعت فن را که در مقاله قبلی این مجموعه طراحی شده بود، اجرا خواهیم کرد.
یک سیستم فازی با در نظر گرفتن متغیرهای ورودی و خروجی و طراحی مجموعه‌های فازی برای توضیح آن متغیر آغاز می‌شود.
متغیرها به یک حد پایین و بالا نیاز دارند و همانطور که با مجموعه‌های فازی گسسته، با وضوح سیستم کار خواهیم کرد.
بنابراین یک تعریف متغیر به شرح زیر است:

```
temp = FuzzyInputVariable('Temperature', 10, 40, 100)
```

که در آن متغیر "Temperature" بین 10 تا 40 درجه است و در 100 سطل گسسته می‌شود.
مجموعه‌های فازی تعریف شده برای متغیر بسته به شکل آن‌ها به پارامترهای مختلفی نیاز دارند. به عنوان مثال، در مورد مجموعه‌های مثلثی، سه پارامتر لازم است، دو پارامتر برای اندام تحتانی و فوقانی دارای درجه عضویت 0 و دیگری برای راس که درجه عضویت آن 1 است. تعریف مجموعه مثلثی برای بنابراین، دما می‌تواند به صورت زیر باشد:

```
temp.add_triangular('Cold', 10, 10, 25)
```

جایی که مجموعه ای به نام "Cold" در 10 و 25 و اوج در 10 درجه دارای افراط است. در سیستم ما ، دو متغیر ورودی "Temperature" و "Humidity" و یک متغیر خروجی واحد "Speed" را در نظر گرفتیم. هر متغیر با سه مجموعه فازی توصیف شده است. تعریف متغیر خروجی "Speed" به شرح زیر است:

```
motor_speed = FuzzyOutputVariable('Speed', 0, 100, 100)
motor_speed.add_triangular('Slow', 0, 0, 50)
motor_speed.add_triangular('Moderate', 10, 50, 90)
motor_speed.add_triangular('Fast', 50, 100, 100)
```

همانطور که قبلاً مشاهده کردیم، سیستم فازی موجودی است که شامل این متغیرها و قوانین فازی خواهد بود. بنابراین متغیرها باید به سیستم زیر به سیستم اضافه شوند:

```
system = FuzzySystem()
system.add_input_variable(temp)
system.add_input_variable(humidity)
system.add_output_variable(motor_speed)
```

قوانین فازی:

یک سیستم فازی قوانین فازی را برای عملکرد فرم اجرا می‌کند:

```
If x1 is S and x2 is M then y is S
```

که در آن اگر بخش از قانون شامل چندین بند پیشین است و بخش پس از آن شامل چندین بند متعاقب است. برای ساده نگه‌داشتن موارد، قوانینی را در نظر می‌گیریم که از هر متغیر ورودی به یک بند پیشین احتیاج دارد و فقط با یک جمله 'and' به هم پیوند خورده‌اند ممکن است عباراتی با 'or' پیوند داده شوند و عبارات همچنین می‌توانند حاوی عملگرهایی در مجموعه ها مانند 'not' باشند.
ساده‌ترین راه برای افزودن یک قانون مبهم به سیستم ما ارائه لیستی از بندهای پیشین و بندهای متعاقب آن است. یکی از روش‌های انجام این کار استفاده از فرهنگ لغت پایتون است که حاوی آن است.

```
Variable:Set
```

ورودی‌های مجموعه بندها. از این رو قانون فوق می‌تواند به شرح زیر اجرا شود:

```
system.add_rule( { 'Temperature':'Cold',  'Humidity':'Wet' },{
'Speed':'Slow'})
```

اجرای سیستم شامل ورودی مقادیر برای همه متغیرهای ورودی و بدست آوردن مقادیر مقادیر خروجی در ازای آن است. بازهم این امر با استفاده از فرهنگ نامه هایی که نام متغیرها را به عنوان کلید در اختیار ما قرار می‌دهد، حاصل می‌شود.

```
output = system.evaluate_output({
'Temperature':18,
'Humidity':60  })
```

سیستم یک فرهنگ لغت را که حاوی نام متغیرهای خروجی است به عنوان کلید و نتیجه نامفهوم را به عنوان مقادیر برمی‌گرداند.

نتیجه‌گیری:

در این گزارش، ما به بررسی عملی یک سیستم استنتاج فازی پرداخته‌ایم. در حالی که کتابخانه ارائه شده در اینجا به کارهای بیشتری احتیاج دارد تا بتواند در پروژه های واقعی از جمله اعتبار سنجی و استفاده از استثناها مورد استفاده قرار گیرد، اما می‌تواند به عنوان پایه ای برای پروژه‌هایی باشد که به کنفرانس فازی نیاز دارند. در فایل یک نمونه پیوست شده است.

منابع:

https://blog.faradars.org/%DA%A9%D9%86%D8%AA%D8%B1%D9%84-%D9%81%D8%A7%D8%B2%DB%8C/

https://fa.wikipedia.org/wiki/%D8%B3%D8%A7%D9%85%D8%A7%D9%86%D9%87_%DA%A9%D9%86%D8%AA%D8%B1%D9%84_%D9%81%D8%A7%D8%B2%DB%8C

https://towardsdatascience.com/fuzzy-inference-system-implementation-in-python-8af88d1f0a6e