

# راهنمای کامل حل‌کننده ژنتیک سیستم معادلات

## Complete Guide to Genetic Algorithm Equation Solver

محمد مهدی شریف‌بیگی  
MohammadMahdi Sharifbeigy

مستندات فنی الگوریتم ژنتیک  
Technical Documentation of Genetic Algorithm

۱ خرداد ۱۴۰۴

### فهرست مطالب

۱	مقدمه	۳
۱.۱	ویژگی‌های کلیدی	۳
۲	تابع شایستگی	۳
۱.۲	چرا از این تابع شایستگی استفاده شد؟	۳
۲.۲	دلایل انتخاب این روش:	۳
۱.۲.۲	استفاده از خطای مربعی ( $result**2$ )	۳
۲.۲.۲	معکوس خطا ( $1.0 / total\_error$ )	۴
۳.۲.۲	مدیریت استثناء	۴
۴.۲.۲	تشخیص جواب دقیق	۴
۳	روش‌های جایگزین تابع شایستگی	۴
۱.۳	روش خطای خطی (Linear Error)	۴
۲.۳	روش خطای لگاریتمی (Logarithmic Error)	۵
۳.۳	روش وزن‌دار (Weighted Error)	۵
۴.۳	روش نرمالایز شده (Normalized Error)	۶
۵.۳	روش تطبیقی (Adaptive Fitness)	۶
۶.۳	روش ترکیبی چندگانه (Multi-Objective)	۷
۴	جمع‌بندی تابع شایستگی	۸
۱.۴	چرا روش فعلی (خطای مربعی معکوس) بهترین انتخاب است؟	۸
۲.۴	کی از روش‌های دیگر استفاده کنیم؟	۸

۸	<b>۵ عوامل مؤثر در الگوریتم ژنتیک</b>
۸	۱.۵ فهرست عوامل به ترتیب اولویت
۹	۲.۵ روش انتخاب (Selection Strategy)
۹	۱.۲.۵ تأثیر: بالا (اولویت ۱)
۹	۲.۲.۵ روش استفاده شده در کد:
۹	۳.۲.۵ روش‌های جایگزین:
۱۰	<b>۶ مثال‌های کاربردی</b>
۱۰	۱.۶ حل سیستم معادلات خطی
۱۱	۲.۶ حل سیستم معادلات غیرخطی
۱۱	<b>۷ بهینه‌سازی پارامترها</b>
۱۱	۱.۷ راهنمای تنظیم پارامترها
۱۱	۱.۱.۷ اندازه جمعیت (Population Size)
۱۲	۲.۱.۷ نرخ جهش (Mutation Rate)
۱۲	۳.۱.۷ اندازه تورنمنت (Tournament Size)
۱۲	<b>۸ خطاهای رایج و راه‌حل‌ها</b>
۱۲	۱.۸ همگرایی زودرس
۱۲	۲.۸ همگرایی کند
۱۳	<b>۹ مطالعات موردی</b>
۱۳	۱.۹ سیستم معادلات درجه دوم
۱۳	۲.۹ سیستم معادلات با پارامترهای کسری
۱۴	<b>۱۰ نتیجه‌گیری</b>
۱۴	۱.۱۰ نقاط قوت سیستم
۱۴	۲.۱۰ محدودیت‌ها
۱۴	۳.۱۰ پیشنهادات برای توسعه آینده
۱۴	<b>۱۱ منابع و مراجع</b>
۱۵	<b>۱۲ ضمائم</b>
۱۵	۱.۱۲ ضمیمه الف: کد کامل کلاس اصلی
۱۸	۲.۱۲ ضمیمه ب: مثال‌های تست
۱۹	۳.۱۲ ضمیمه ج: جدول پارامترهای بهینه

## ۱ مقدمه

این مستند راهنمای کاملی برای درک و استفاده از حل‌کننده ژنتیک سیستم معادلات ارائه می‌دهد. این سیستم قادر به حل انواع مختلف معادلات خطی و غیرخطی با استفاده از الگوریتم ژنتیک پیشرفته است.

### ۱.۱ ویژگی‌های کلیدی

- حل سیستم‌های معادلات خطی و غیرخطی
- استفاده از استراتژی‌های متنوع جهش و ترکیب
- پشتیبانی از تابع شایستگی تطبیقی
- قابلیت راه‌اندازی مجدد خودکار
- تبدیل اعداد اعشاری به کسرهای دقیق

## ۲ تابع شایستگی

### ۱.۲ چرا از این تابع شایستگی استفاده شد؟

```

1 def fitness_function(self, chromosome, equations):
2     total_error = 0
3     for eq in equations:
4         try:
5             result = eq(*chromosome)
6             total_error += result**2 #
7         except:
8             total_error += 1e6
9
10    if total_error < 1e-15:
11        return float('inf')
12
13    return 1.0 / total_error
14
```

Listing 1: اصلی شایستگی تابع

### ۲.۲ دلایل انتخاب این روش:

#### ۱.۲.۲ استفاده از خطای مربعی ( $result**2$ )

- دلیل: خطاهای بزرگ بیشتر تنبیه می‌شوند
- مزیت: همگرایی سریع‌تر به جواب دقیق
- مثال: اگر دو کروموزوم خطاهای ۱.۰ و ۱۰.۰ داشته باشند:

- روش خطی:  $1.0 \text{ vs } 0.0$  (تفاوت  $10$  برابری)
- روش مربعی:  $0.0 \text{ vs } 0.0001$  (تفاوت  $100$  برابری)

### ۲.۲.۲ معکوس خطا ( $1.0 / \text{total\_error}$ )

- دلیل: تبدیل مسئله minimization به maximization
- مزیت: الگوریتم ژنتیک معمولاً برای بیشینه‌سازی طراحی شده
- نتیجه: خطای کمتر = شایستگی بیشتر

### ۳.۲.۲ مدیریت استثناء

- جریمه سنگین ( $1e6$ ) برای کروموزوم‌های نامعتبر
- جلوگیری از تقسیم بر صفر، سرریز عددی، و خطاهای محاسباتی

### ۴.۲.۲ تشخیص جواب دقیق

- اگر خطا کمتر از  $1e-15$  باشد، شایستگی  $\text{inf}$  می‌شود
- نتیجه: توقف زودهنگام در صورت یافتن جواب دقیق

## ۳ روش‌های جایگزین تابع شایستگی

### ۱.۳ روش خطای خطی (Linear Error)

```

1 def linear_fitness(self, chromosome, equations):
2     total_error = 0
3     for eq in equations:
4         total_error += abs(eq(*chromosome))
5     return 1.0 / (1.0 + total_error)
6

```

Listing 2: خطای شایستگی تابع

#### مزایا:

- ساده‌تر در پیاده‌سازی
- حساسیت کمتر به نویز
- پایداری عددی بهتر

#### معایب:

- همگرایی کندتر
- تمایز کمتر بین جواب‌های خوب و عالی
- ممکن است در جواب‌های تقریبی گیر کند

### ۲.۳ روش خطای لگاریتمی (Logarithmic Error)

```

1  def log_fitness(self, chromosome, equations):
2  total_error = 0
3  for eq in equations:
4  error = abs(eq(*chromosome))
5  if error > 0:
6  total_error += math.log(1 + error)
7  return 1.0 / (1.0 + total_error)
8

```

Listing 3: لگاریتمی شایستگی تابع

#### مزایا:

- کاهش اثر خطاهای بزرگ
- تعادل بهتر بین دقت و پایداری
- مناسب برای مسائل با مقیاس‌های مختلف

#### معایب:

- پیچیدگی محاسباتی بیشتر
- ممکن است دقت نهایی کمتری داشته باشد
- نیاز به تنظیم پارامترهای اضافی

### ۳.۳ روش وزن‌دار (Weighted Error)

```

1  def weighted_fitness(self, chromosome, equations, weights=None):
2  if weights is None:
3  weights = [1.0] * len(equations)
4
5  total_error = 0
6  for eq, weight in zip(equations, weights):
7  error = eq(*chromosome)
8  total_error += weight * (error ** 2)
9
10 return 1.0 / (1.0 + total_error)
11

```

Listing 4: وزن‌دار شایستگی تابع

#### مزایا:

- امکان اولویت‌بندی معادلات مختلف
- انعطاف‌پذیری بالا
- کنترل بهتر بر فرآیند همگرایی

**معایب:**

- نیاز به تعیین وزن‌های مناسب
- پیچیدگی بیشتر در تنظیم
- ممکن است به برخی معادلات بی‌توجهی کند

**۴.۳ روش نرمالایز شده (Normalized Error)**

```

1  def normalized_fitness(self, chromosome, equations):
2      errors = []
3      for eq in equations:
4          errors.append(abs(eq*chromosome))
5
6      #
7      mean_error = sum(errors) / len(errors)
8      if mean_error == 0:
9          return float('inf')
10
11     normalized_error = sum(e / mean_error for e in errors)
12     return 1.0 / (1.0 + normalized_error)
13

```

Listing 5: شده نرمال‌ایز شایستگی تابع

**مزایا:**

- تعادل بهتر بین معادلات مختلف
- کاهش اثر معادلات با مقیاس بزرگ
- عملکرد یکنواخت‌تر

**معایب:**

- محاسبات اضافی برای نرمالایز
- ممکن است اطلاعاتی از دست برود
- پیچیدگی در پیاده‌سازی

**۵.۳ روش تطبیقی (Adaptive Fitness)**

```

1  def adaptive_fitness(self, chromosome, equations, generation):
2      total_error = 0
3
4      #
5      adaptive_power = 2.0 + (generation / 1000.0)
6
7      for eq in equations:
8          error = abs(eq*chromosome)

```

```

9      total_error += error ** adaptive_power
10
11      return 1.0 / (1.0 + total_error)
12

```

Listing 6: تطبیقی شایستگی تابع

**مزایا:**

- تطبیق خودکار با پیشرفت الگوریتم
- شروع با جستجوی گسترده، ادامه با جستجوی دقیق
- بهینه‌سازی مراحل مختلف حل

**معایب:**

- پیچیدگی زیاد
- نیاز به تنظیم پارامترهای بیشتر
- ممکن است پیش‌بینی‌ناپذیر باشد

**۶.۳ روش ترکیبی چندگانه (Multi-Objective)**

```

1      def multi_objective_fitness(self, chromosome, equations):
2          # :1
3          total_error = sum(eq(*chromosome)**2 for eq in equations)
4
5          # :2          )          (
6          complexity = sum(abs(x) for x in chromosome)
7
8          # :3
9          fraction_score = sum(1 for x in chromosome
10             if abs(x - round(x*4)/4) < 1e-3)
11
12         #
13         fitness = (1.0 / (1 + total_error)) * 0.7 + \
14             (1.0 / (1 + complexity)) * 0.2 + \
15             fraction_score * 0.1
16
17         return fitness
18

```

Listing 7: چندهدفه شایستگی تابع

**مزایا:**

- در نظر گیری چندین معیار همزمان
- تولید جواب‌های عملی‌تر
- انعطاف‌پذیری بسیار بالا

**معایب:**

- پیچیدگی زیاد در طراحی و تنظیم
- ممکن است همگرایی آهسته‌تر داشته باشد
- نیاز به دانش تخصصی برای تنظیم وزن‌ها

**۴ جمع‌بندی تابع شایستگی****۱.۴ چرا روش فعلی (خطای مربعی معکوس) بهترین انتخاب است؟**

۱. سادگی و کارایی: پیاده‌سازی ساده با عملکرد عالی
۲. همگرایی سریع: خطاهای بزرگ شدیداً تنبیه می‌شوند
۳. دقت بالا: قادر به یافتن جواب‌های بسیار دقیق
۴. پایداری: مدیریت مناسب حالات استثنایی

**۲.۴ کی از روش‌های دیگر استفاده کنیم؟**

- خطای خطی: برای مسائل حساس به نویز
  - وزن‌دار: وقتی برخی معادلات اهمیت بیشتری دارند
  - نرمالایز: برای معادلات با مقیاس‌های مختلف
  - تطبیقی: برای مسائل خیلی پیچیده
  - چندهدفه: وقتی کیفیت جواب اهمیت دارد
- این انتخاب بستگی به نوع مسئله، دقت مورد نیاز، و منابع محاسباتی در دسترس دارد.

**۵ عوامل مؤثر در الگوریتم ژنتیک****۱.۵ فهرست عوامل به ترتیب اولویت**

۱. روش انتخاب (Selection Strategy)
۲. عملگر ترکیب (Crossover Operator)
۳. استراتژی جهش (Mutation Strategy)
۴. مدیریت جمعیت (Population Management)
۵. پارامترهای کنترلی (Control Parameters)
۶. نحوه ایجاد جمعیت اولیه (Initialization Strategy)
۷. معیارهای توقف (Termination Criteria)



## ۲.۵ روش انتخاب (Selection Strategy)

### ۱.۲.۵ تأثیر: بالا (اولویت ۱)

دلیل: تعیین‌کننده نحوه انتشار ژن‌های خوب در جمعیت

### ۲.۲.۵ روش استفاده شده در کد:

```

1 def selection(self, ranked_population):
2     tournament = random.sample(ranked_population, self.tournament_size)
3     if random.random() < 0.9:
4         return max(tournament, key=lambda x: x[1])[0]
5     else:
6         sorted_tournament = sorted(tournament, key=lambda x: x[1], reverse=True)
7         return sorted_tournament[1][0] if len(sorted_tournament) > 1 else
8         sorted_tournament[0][0]

```

Listing 8: Tournament Selection

نوع: Selection Tournament با اندازه ۵

### ۳.۲.۵ روش‌های جایگزین:

#### ۱.۱ Selection Wheel Roulette

```

1 def roulette_wheel_selection(self, ranked_population):
2     total_fitness = sum(fitness for _, fitness in ranked_population)
3     pick = random.uniform(0, total_fitness)
4     current = 0
5     for chromosome, fitness in ranked_population:
6         current += fitness
7         if current > pick:
8             return chromosome
9

```

Listing 9: Roulette Wheel Selection

مزایا:

- احتمال انتخاب متناسب با شایستگی
- حفظ تنوع بهتر
- پیاده‌سازی ساده

معایب:

- حساس به scaling مقادیر شایستگی
- ممکن است افراد ضعیف زیاد انتخاب شوند
- مشکل با شایستگی‌های منفی

## ۲.۱. Selection Rank-Based

```

1  def rank_selection(self, ranked_population):
2  n = len(ranked_population)
3  # rank fitness
4  ranks = list(range(n, 0, -1)) # = n = 1
5  total_rank = sum(ranks)
6  pick = random.uniform(0, total_rank)
7  current = 0
8  for i, (chromosome, _) in enumerate(ranked_population):
9  current += ranks[i]
10 if current > pick:
11 return chromosome
12

```

Listing 10: Rank-Based Selection

## مزایا:

- مستقل از مقیاس شایستگی
- کنترل بهتر فشار انتخاب
- جلوگیری از dominance زودرس

## معایب:

- از دست دادن اطلاعات دقیق شایستگی
- محاسبات بیشتر برای رتبه‌بندی
- ممکن است همگرایی کندتر باشد

## ۶ مثال‌های کاربردی

## ۱.۶ حل سیستم معادلات خطی

```

1  # :
2  # 2x + 3y = 7
3  # x - y = 1
4
5  equations_text = [
6  "2*x + 3*y - 7",
7  "x - y - 1"
8  ]
9
10 \begin{lstlisting}
11 \begin{lstlisting}[language=Python, caption = [
12 # :
13 # 2x + 3y = 7
14 # x - y = 1
15
16 equations_text = [
17 "2*x + 3*y - 7",

```

```

18     "x - y - 1"
19 ]
20
21 solver = EnhancedGeneticEquationSolver(
22     pop_size=300,
23     generations=1000,
24     mutation_rate=0.15
25 )
26
27 result = solve_equations(equations_text, solver)
28 print(f"    : x = {result[0]}, y = {result[1]}")
29

```

Listing 11: مت‌غیر دو خطی سیستم مثال

## ۲.۶ حل سیستم معادلات غیرخطی

```

1  #      :
2  #  $x^2 + y^2 = 25$ 
3  #  $x + y = 7$ 
4
5  equations_text = [
6  "x**2 + y**2 - 25",
7  "x + y - 7"
8  ]
9
10 solver = EnhancedGeneticEquationSolver(
11     pop_size=500,
12     generations=3000,
13     mutation_rate=0.2,
14     adaptive_mutation=True
15 )
16
17 result = solve_equations(equations_text, solver)
18 print(f"    : x = {result[0]}, y = {result[1]}")
19

```

Listing 12: غیرخطی سیستم مثال

## ۷ بهینه‌سازی پارامترها

### ۱.۷ راهنمای تنظیم پارامترها

#### ۱.۱.۷ اندازه جمعیت (Population Size)

- مسائل ساده: ۱۰۰-۳۰۰
- مسائل متوسط: ۳۰۰-۵۰۰
- مسائل پیچیده: ۵۰۰-۱۰۰۰

**۲.۱.۷ نرخ جهش (Mutation Rate)**

- شروع: ۳.۰-۲.۰ (اکتشاف بالا)
- میانه: ۲.۰-۱.۰ (تعادل)
- پایان: ۱.۰-۰.۵ (بهره‌برداری)

**۳.۱.۷ اندازه تورنمنت (Tournament Size)**

- کوچک (۳-۲): تنوع بیشتر، همگرایی کندتر
- متوسط (۶-۴): تعادل مناسب
- بزرگ (۱۰-۷): همگرایی سریع، خطر optimum محلی

**۸ خطاهای رایج و راه‌حل‌ها****۱.۸ همگرایی زودرس****علائم:**

- توقف بهبود در نسل‌های اولیه
- کاهش شدید تنوع جمعیت

**راه‌حل‌ها:**

- افزایش اندازه جمعیت
- کاهش اندازه تورنمنت
- افزایش نرخ جهش
- استفاده از mechanism restart

**۲.۸ همگرایی کند****علائم:**

- بهبود بسیار آهسته fitness
- عدم رسیدن به جواب در تعداد نسل‌های تعیین شده

**راه‌حل‌ها:**

- افزایش اندازه تورنمنت
- کاهش نرخ جهش
- بهبود تابع شایستگی
- استفاده از elitism بیشتر

## ۹ مطالعات موردی

### ۱.۹ سیستم معادلات درجه دوم

مسئله:

$$x^2 + y^2 = 13 \quad (۱)$$

$$x + y = 5 \quad (۲)$$

تحلیل:

- دو جواب دارد: (۲,۳) و (۳,۲)
- نیاز به تنوع بالا برای یافتن هر دو جواب
- تابع شایستگی مربعی مناسب است

پارامترهای بهینه:

```

1 solver = EnhancedGeneticEquationSolver(
2     pop_size=400,
3     generations=2000,
4     mutation_rate=0.25,
5     tournament_size=4,
6     elite_size=40
7 )
8

```

### ۲.۹ سیستم معادلات با پارامترهای کسری

مسئله:

$$\frac{1}{2}x + \frac{1}{3}y = 1 \quad (۳)$$

$$\frac{1}{4}x - \frac{1}{6}y = 0 \quad (۴)$$

تحلیل:

- جواب کسری دارد
- نیاز به precision بالا
- استفاده از fraction\_precision=True

پارامترهای بهینه:

```

1 solver = EnhancedGeneticEquationSolver(
2     pop_size=300,
3     generations=1500,
4     mutation_rate=0.15,
5     fraction_precision=True,
6     adaptive_mutation=True
7 )
8

```

## ۱۰ نتیجه‌گیری

### ۱.۱۰ نقاط قوت سیستم

۱. انعطاف‌پذیری: قابلیت حل انواع مختلف معادلات
۲. مقاومت: عدم گیر کردن در optimum محلی
۳. دقت: یافتن جواب‌های بسیار دقیق
۴. سازگاری: تطبیق خودکار با انواع مسائل
۵. کارایی: استفاده از تکنیک‌های بهینه‌سازی

### ۲.۱۰ محدودیت‌ها

۱. زمان محاسبه: نیاز به زمان بیشتر نسبت به روش‌های مستقیم
۲. تنظیم پارامتر: نیاز به دانش برای تنظیم بهینه
۳. تضمین همگرایی: عدم تضمین یافتن جواب در زمان محدود
۴. پیچیدگی: درک و debug کردن سخت‌تر از روش‌های کلاسیک

### ۳.۱۰ پیشنهادات برای توسعه آینده

۱. Processing Parallel: موازی‌سازی محاسبات
۲. Methods Hybrid: ترکیب با روش‌های عددی کلاسیک
۳. Learning Machine: یادگیری خودکار پارامترها
۴. Multi-Objective: حل همزمان چندین هدف
۵. Handling Constraint: مدیریت بهتر محدودیت‌ها

## ۱۱ منابع و مراجع

۱. E. D. Goldberg, (۱۹۸۹). Genetic Algorithms and Optimization, Search, in Algorithms Genetic Learning chine
۲. H. J. Holland, (۱۹۹۲). Systems Artificial and Natural in Adaptation
۳. Z. Michalewicz, (۱۹۹۶). Evolution = Structures Data + Algorithms Genetic Programs
۴. C. A. C. Coello, (۲۰۰۲). tech- constraint-handling numerical and Theoretical niques
۵. K. Deb, (۲۰۰۱). Algorithms Evolutionary using Optimization Multi-Objective

## ۱۲ ضمایم

## ۱.۱۲ ضمیمه الف: کد کامل کلاس اصلی

```

1  import random
2  import math
3  from fractions import Fraction
4
5  class EnhancedGeneticEquationSolver:
6  def __init__(self, pop_size=500, generations=5000, mutation_rate=0.2,
7  elite_size=50, var_range=(-100, 100), tournament_size=5,
8  stagnation_limit=200, adaptive_mutation=True,
9  fraction_precision=True):
10
11  self.pop_size = pop_size
12  self.generations = generations
13  self.base_mutation_rate = mutation_rate
14  self.current_mutation_rate = mutation_rate
15  self.elite_size = elite_size
16  self.var_range = var_range
17  self.tournament_size = tournament_size
18  self.stagnation_limit = stagnation_limit
19  self.adaptive_mutation = adaptive_mutation
20  self.fraction_precision = fraction_precision
21
22  #
23  self.generation_stats = []
24  self.best_fitness_history = []
25  self.restart_count = 0
26
27  def fitness_function(self, chromosome, equations):
28  total_error = 0
29  for eq in equations:
30  try:
31  result = eq(*chromosome)
32  total_error += result**2
33  except:
34  total_error += 1e6
35
36  if total_error < 1e-15:
37  return float('inf')
38
39  return 1.0 / total_error
40
41  def initialize_population(self, var_count):
42  population = []
43  for _ in range(self.pop_size):
44  chromosome = []
45  for _ in range(var_count):
46  strategy = random.choice(['integer', 'small', 'fraction', 'decimal'])
47
48  if strategy == 'integer':
49  value = random.randint(-10, 10)
50  elif strategy == 'small':
51  value = random.uniform(-2, 2)
52  elif strategy == 'fraction':

```

```

53     value = random.choice([0.5, 0.25, 0.75, 1.5, 2.5, -0.5, -0.25])
54     else: # decimal
55         value = random.uniform(self.var_range[0], self.var_range[1])
56
57     chromosome.append(value)
58     population.append(chromosome)
59     return population
60
61     def evaluate_population(self, population, equations):
62         evaluated = []
63         for chromosome in population:
64             fitness = self.fitness_function(chromosome, equations)
65             evaluated.append((chromosome, fitness))
66         return sorted(evaluated, key=lambda x: x[1], reverse=True)
67
68     def selection(self, ranked_population):
69         tournament = random.sample(ranked_population, self.tournament_size)
70         if random.random() < 0.9:
71             return max(tournament, key=lambda x: x[1])[0]
72         else:
73             sorted_tournament = sorted(tournament, key=lambda x: x[1], reverse=True)
74             return sorted_tournament[1][0] if len(sorted_tournament) > 1 else
sorted_tournament[0][0]
75
76     def crossover(self, parent1, parent2):
77         strategy = random.choice(['uniform', 'arithmetic', 'single_point'])
78
79         if strategy == 'uniform':
80             child = [p1 if random.random() < 0.5 else p2 for p1, p2 in zip(parent1,
parent2)]
81         elif strategy == 'arithmetic':
82             alpha = random.random()
83             child = [alpha * p1 + (1 - alpha) * p2 for p1, p2 in zip(parent1, parent2)]
84         else: # single_point
85             point = random.randint(1, len(parent1) - 1)
86             child = parent1[:point] + parent2[point:]
87
88         return child
89
90     def mutation(self, chromosome, generation, max_generations):
91         if random.random() > self.current_mutation_rate:
92             return chromosome
93
94         mutated = chromosome.copy()
95         gene_index = random.randint(0, len(mutated) - 1)
96
97         strategy = random.choices(
98             ['random', 'small_change', 'fraction', 'zero', 'sign_flip'],
99             weights=[0.3, 0.4, 0.2, 0.05, 0.05],
100             k=1
101         )[0]
102
103         if strategy == 'random':
104             mutated[gene_index] = random.uniform(self.var_range[0], self.var_range[1])
105         elif strategy == 'small_change':
106             change = random.gauss(0, 0.1)
107             mutated[gene_index] += change
108         elif strategy == 'fraction':

```



```

109 mutated[gene_index] = random.choice([0.5, 0.25, 0.75, 1.0, 2.0, -0.5, -1.0])
110 elif strategy == 'zero':
111 mutated[gene_index] = random.uniform(-0.1, 0.1)
112 elif strategy == 'sign_flip':
113 mutated[gene_index] = -mutated[gene_index]
114
115 # Update mutation rate
116 if self.adaptive_mutation:
117 progress = min(1.0, generation / (max_generations * 0.7))
118 self.current_mutation_rate = self.base_mutation_rate * (1.0 - 0.6 * progress
)
119
120 return mutated
121
122 def solve(self, equations, max_time=300):
123 var_count = len(equations)
124 population = self.initialize_population(var_count)
125
126 best_solution = None
127 best_fitness = 0
128 stagnation_counter = 0
129
130 for generation in range(self.generations):
131 # Evaluate population
132 ranked_population = self.evaluate_population(population, equations)
133 current_best_fitness = ranked_population[0][1]
134
135 # Check for improvement
136 if current_best_fitness > best_fitness:
137 best_fitness = current_best_fitness
138 best_solution = ranked_population[0][0].copy()
139 stagnation_counter = 0
140 else:
141 stagnation_counter += 1
142
143 # Check termination conditions
144 if best_fitness > 1e10: # Found exact solution
145 break
146
147 if stagnation_counter >= self.stagnation_limit:
148 # Restart with new population
149 population = self.initialize_population(var_count)
150 self.restart_count += 1
151 stagnation_counter = 0
152 continue
153
154 # Create next generation
155 new_population = []
156
157 # Elitism
158 for i in range(self.elite_size):
159 new_population.append(ranked_population[i][0])
160
161 # Generate offspring
162 while len(new_population) < self.pop_size:
163 parent1 = self.selection(ranked_population)
164 parent2 = self.selection(ranked_population)
165 child = self.crossover(parent1, parent2)

```

```

166     child = self.mutation(child, generation, self.generations)
167     new_population.append(child)
168
169     population = new_population
170
171     return best_solution, best_fitness
172

```

Listing 13: EnhancedGeneticEquationSolver کامل کلاس

## ۲.۱۲ ضمیمه ب: مثال‌های تست

```

1  def test_linear_2var():
2      equations = [
3          "2*x + 3*y - 7",
4          "x - y - 1"
5      ]
6      return equations
7
8  def test_nonlinear_2var():
9      equations = [
10         "x**2 + y**2 - 25",
11         "x + y - 7"
12     ]
13     return equations
14
15  def test_complex_3var():
16      equations = [
17         "x + y + z - 6",
18         "2*x - y + z - 1",
19         "x + 2*y - z - 2"
20     ]
21     return equations
22
23  def solve_equations(equation_strings, solver):
24      # Parse equations
25      equations = []
26      for eq_str in equation_strings:
27          # Convert to lambda function
28          variables = ['x', 'y', 'z', 'w'][:len(equation_strings)]
29          eq_lambda = eval(f"lambda {', '.join(variables)}: {eq_str}")
30          equations.append(eq_lambda)
31
32      # Solve
33      solution, fitness = solver.solve(equations)
34
35      return solution
36

```

Listing 14: مختلف تست نتوابع

## ۳.۱۲ ضمیمه ج: جدول پارامترهای بهینه

نوع مسئله	اندازه جمعیت	نرخ جهش	تعداد نسل	اندازه تورنمنت
خطی ساده	۳۰۰-۲۰۰	۲.۰-۱۵.۰	۱۰۰۰-۵۰۰	۴-۳
خطی پیچیده	۵۰۰-۳۰۰	۲۵.۰-۲.۰	۲۰۰۰-۱۰۰۰	۵-۴
غیرخطی ساده	۶۰۰-۴۰۰	۳.۰-۲.۰	۳۰۰۰-۱۵۰۰	۶-۴
غیرخطی پیچیده	۸۰۰-۵۰۰	۳۵.۰-۲۵.۰	۳۰۰۰-۵۰۰۰	۷-۵
کسری	۵۰۰-۳۰۰	۲.۰-۱۵.۰	۲۰۰۰-۱۰۰۰	۵-۳