پاسخ تمرین چهارم

بخش نظری

سوال اول (فصل ششم)

الف) متدهای مولد اعداد شبه‌تصادفی باید چه ویژگی‌هایی داشته باشند؟ (۵ مورد)

پاسخ. باید ویژگی‌های زیر را داشته باشند:

1. سریع باشند؛ زیرا در یک فرایند شبیه‌سازی ممکن است به میلیون‌ها عدد رندوم نیاز باشد.
2. در سیستم‌ها و زبان‌های مختلف، قابل اجرا (با همان عملکرد) باشند.
3. به اندازه‌ی کافی چرخه‌ی آن‌ها بزرگ باشد.
4. سری اعداد قابل تکرار باشند (با مهیا کردن شرایط اولیه‌ی یکسان)؛ برای رفع خطا مفید است.
5. مهم‌تر از همه، باید بتوانند از نظر استقلال و یکنواختی اعداد تولید شده، به شرایط ایده‌آل نزدیک باشند.

ب) بررسی کنید آیا هریک از مولدهای تولید اعداد تصادفی زیر می‌توانند بیشترین طول دوره (cycle length) را به‌دست آورند؟ در این حالت محدودیت‌های X_0 برای به‌دست آمدن این دوره را ذکر کنید.

- $a = 4591,\ m = 256,\ c = 247$

پاسخ. خیر؛ بیش‌ترین طول دوره اینجا همان 256 است (زیرا m توانی از 2 است و c صفر نیست) و زمانی به‌دست می‌آید که m و c نسبت به هم اول باشند و a=4k+1 که مورد دوم برقرار نیست (X_0 محدودیتی ندارد).

- $a = 6507,\ m = 1024,\ c = 0$

پاسخ. بله؛ اینجا بیش‌ترین طول دوره برابر m/4 یا 256 است (زیرا m توانی از 2 است و c صفر است) و زمانی برقرار است که a=8k+3 یا a=8k+5 که برقرار است (باقی‌مانده a به 8 برابر 3 است)؛ هم‌چنین X_0 باید فرد باشد.

بخش عملی

سوال اول (فصل ششم)

الف) با روش Combined Linear Congruential و پارامترهای داده شده، ۱۰۰ عدد تصادفی تولید کنید.

$$m_1 = 100,\ c_1 = 43,\ a_1 = 23,\ X_1 = 13$$
$$m_2 = 99,\ c_2 = 47,\ a_2 = 27,\ X_2 = 17$$

پاسخ. نمونه کد:

```
def sample(count):
    for _ in range(count):
        X_1['i'] = (X_1['a'] * X_1['i']) % X_1['m']
        X_2['i'] = (X_2['a'] * X_2['i']) % X_2['m']
        x = (X_1['i'] - X_2['i']) % (X_1['m'] - 1)
        R.append(x / X_1['m'] if x > 0 else (X_1['m'] - 1) / X_1['m'])
```

ب) یکنواختی توزیع اعداد تولید شده را با استفاده از آزمون **Kolmogorov-Smirnov** و پارامترهای $N = 20, \alpha = 0.05$ بررسی کنید.

پاسخ. نمونه کد:

```
def kolmogorov_smirnov_test(n, d_alpha=0.294):
    test_data = np.array(sorted(R[:n]))
    seq = np.linspace(0, 1, num=n + 1)  # [0 0.05 0.1 ... 0.95 1]
    d_plus = max(seq[1:] - test_data)
    d_minus = max(test_data - seq[:-1])
    d = max(d_plus, d_minus)
    print(f'D = {d}, D_alpha = {d_alpha}')
    print(f'Kolmogorov test {"did not reject" if d ⩽ d_alpha else "rejected"} H0')
```

پ) یکنواختی توزیع اعداد تولید شده را با استفاده از تست **Chi-Square** و پارامترهای $n = 20, \alpha = 0.05$ بررسی کنید.

پاسخ. نمونه کد:

```
def chi_square_test(n, chi_alpha=30.1):
    seq = np.linspace(0, 1, num=n + 1)
    observed = np.array([np.sum(
    (seq[i] ⩽ np.array(R)) * (np.array(R) < seq[i + 1])) for i in range(n)])
    expected = N / n
    chi_value = np.sum((observed - expected) ** 2 / expected)
    print(f'Chi-Square {"didn't reject" if chi_value ⩽ chi_alpha else "rejected"} H0')
    print(f'(Chi_squared value = {chi_value}, Chi_alpha = {chi_alpha})')
```

ت) میزان استقلال اعداد تولیدشده را با استفاده از مقادیر زیر و $m = 7$ بررسی کنید:

- $i = 3, \alpha = 0.02$
- $i = 7, \alpha = 0.02$

پاسخ. نمونه کد

```
def auto_correlation_test(i, lag, z_alpha=2.326):
    m = int((N - i) / lag - 1)
    rho = sum([R[i + k * lag] * R[i + (k + 1) * lag] for k in range(m + 1)]) \
          / (m + 1) - 0.25
    sigma = np.sqrt(13 * m + 7) / (12 * (m + 1))
    z_0 = rho / sigma
    print(f'Independence test {"did not reject" if -z_alpha ⩽ z_0 ⩽ z_alpha else
"rejected"} H0', end='\t')
    print(f'(i = {i} ⇒ Z_0 value = {z_0}, Z_0.01 = {z_alpha})')
```