In [1]:

```
%matplotlib inline
```

In [78]:

```
import numpy as np
import matplotlib.pyplot as plt

class Solow_Swan():

    def __init__(self, n=0.02,  # population growth rate
                       s=0.20,  # savings rate
                       δ=0.10,   # depreciation rate
                       α=0.36,   # share of labor
                       z=1.00,   # technology constant
                       k=0.01):  # initial capital stock

        self.n, self.s, self.δ, self.α, self.z = n, s, δ, α, z
        self.k = k

    def steady_state(self):
        # Unpack parameters
        n, s, δ, α, z = self.n, self.s, self.δ, self.α, self.z
        # Compute and return steady-state level of capital per worker, output per worker, and
capital per output
        capital_per_worker_steady_state = ((s * z) / (δ + n))**(1 / (1 - α))
        output_per_worker_steady_state = (z * (s / (δ + n))**α)**(1 / (1 - α))
        capital_per_output_steady_state = α / (δ + n)
        return capital_per_worker_steady_state, output_per_worker_steady_state,
capital_per_output_steady_state

    def random_process(self, t):
        # Generate and return random process of length t
        process = np.zeros(t)
        for i in range(t):
            process[i] = np.random.randn()
        return process

    def log_deviation_coefficients(self):
        # Unpack parameters
        n, δ, α = self.n, self.δ, self.α
        # Compute and return coefficients ρ1 and ρ2
        ρ1 = (1 + α * n - δ * (1 - α)) / (1 + n)
        ρ2 = (δ + n) / (1 + n)
        return ρ1, ρ2

    def output_per_worker(self, random_process):
        # Unpack parameters
        α, z, k = self.α, self.z, self.k
        # Generate random process of length t for level of technology
        technology_process = z * np.exp(random_process)
        # Generate and return output per worker
        return technology_process * k**α

    def capital_per_worker(self, random_process):
        # Unpack parameters
        n, s, δ, α, z = self.n, self.s, self.δ, self.α, self.z
        # Update capital per worker
        return (s * self.output_per_worker(random_process) + (1 - δ) * self.k) / (1 + n)

    def capital_per_worker_update(self, random_process):
        # Update current state of capital per worker
        self.k =  self.capital_per_worker(random_process)

    def capital_per_output(self, random_process):
        # Generate and return capital per output
        return self.capital_per_worker(random_process) / self.output_per_worker(random_process)

    def capital_per_worker_log_deviation(self, t, random_process):
        # Unpack parameter
        k = self.k
        # Unpack steady-state level of capital per worker
```

```
        ρ1, ρ2 = self.log_deviation_coefficients()
        # Generate addends for summation of log deviations of capital per worker from steady state
of length t
        if t == 0:
            capital_per_worker_log_deviation_addend = np.log(k / capital_per_worker_steady_state)
        else:
            capital_per_worker_log_deviation_addend = np.zeros(t)
            for i in range(t-1):
                capital_per_worker_log_deviation_addend[i] = ρ2 * ρ1**i * random_process[(t-1)-i]
        # Generate and return log deviation of output per worker from steady state
        capital_per_worker_log_deviation = np.sum(capital_per_worker_log_deviation_addend)
        return capital_per_worker_log_deviation

    def output_per_worker_log_deviation(self, t, random_process):
        #Unpack parameter
        α = self.α
        # Unpack coefficients ρ1 and ρ2
        ρ1, ρ2 = self.log_deviation_coefficients()
        # Generate and return log deviation of output per worker from steady state
        output_per_worker_log_deviation = random_process[t] + α *
self.capital_per_worker_log_deviation(t, random_process)
        return output_per_worker_log_deviation

    def sequence(self, t):
        # Unpack parameters
        n, s, δ, α, z = self.n, self.s, self.δ, self.α, self.z
        # Unpack coefficients ρ1 and ρ2
        ρ1, ρ2 = self.log_deviation_coefficients()
        # Generate time series of length t for capital per worker, output per worker, and capital
per worker
        capital_per_worker_path = np.zeros(t)
        output_per_worker_path = np.zeros(t)
        capital_per_output_path = np.zeros(t)
        # Generate time series of length t for log deviations of capital per worker and output per
worker from steady state
        capital_per_worker_log_deviation_path = np.zeros(t)
        output_per_worker_log_deviation_path = np.zeros(t)
        # Generate random process of length t for level of technology
        random_process = self.random_process(t)
        technology_process = z * np.exp(random_process)
        for i in range(t):
            # Compute current level of capital per output, output per worker, and capital per outpu
t
            capital_per_worker_path[i] = self.k
            output_per_worker_path[i] = self.output_per_worker(random_process[i])
            capital_per_output_path[i] = capital_per_worker_path[i] / output_per_worker_path[i]
            # Update current state of capital per worker
            self.capital_per_worker_update(random_process[i])
            # Compute current level of log deviations of capital per output, output per worker, and
capital per output from steady state
            capital_per_worker_log_deviation_path[i] = self.capital_per_worker_log_deviation(i, ran
dom_process)
            output_per_worker_log_deviation_path[i] = self.output_per_worker_log_deviation(i, rando
m_process)
        return capital_per_worker_path, output_per_worker_path, capital_per_output_path, capital_pe
r_worker_log_deviation_path, output_per_worker_log_deviation_path, random_process
```

In [79]:

```
solow = Solow_Swan()
```

In [98]:

```
t = 150
x = list(range(t))
```

In [99]:

```
solow.steady_state()
```

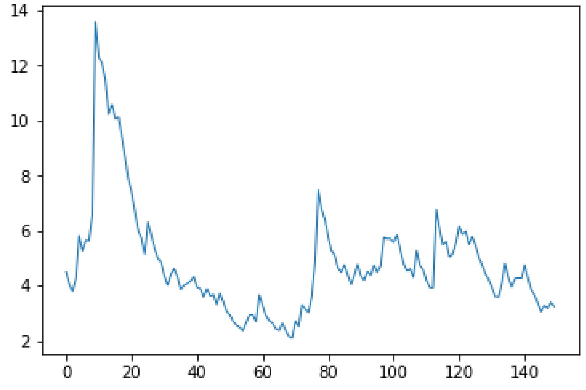
Out[99]:

In [100]:

```
k, y, ky, k_deviation, y_deviation, random_process = solow.sequence(t)
```

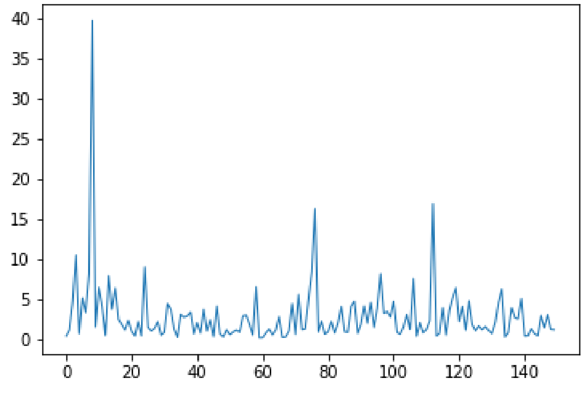
In [101]:

```
fig, ax = plt.subplots()
ax.plot(x, k, linewidth=1)
plt.show()
```



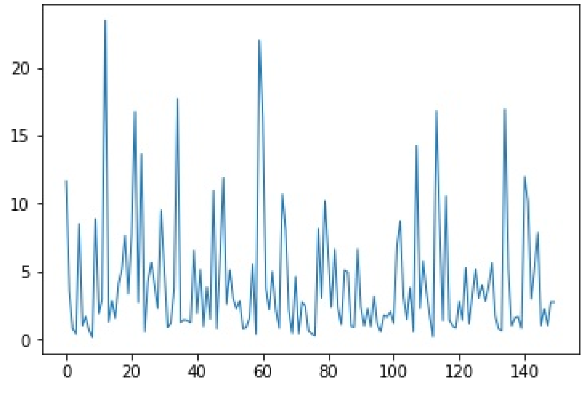
In [102]:

```
fig, ax = plt.subplots()
ax.plot(x, y, linewidth=1)
plt.show()
```



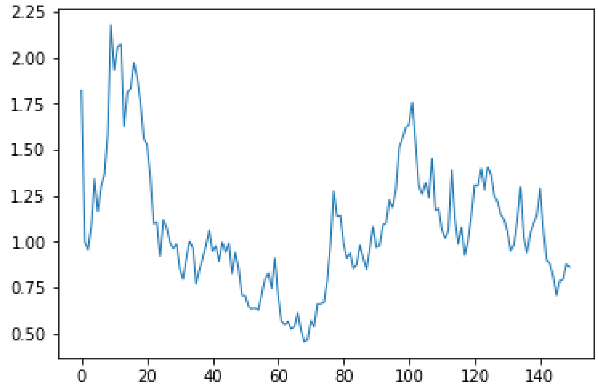
In [103]:

```
fig, ax = plt.subplots()
ax.plot(x, ky, linewidth=1)
plt.show()
```



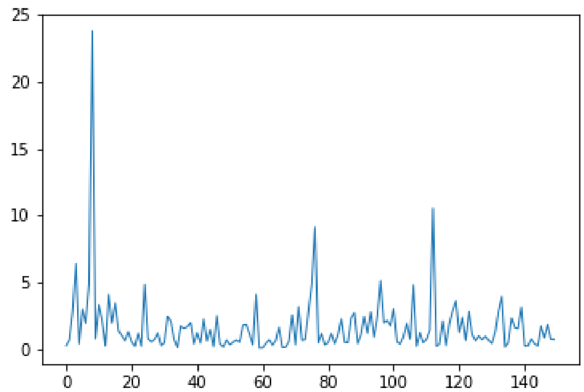
In [104]:

```
ax.plot(x, np.exp(x_deviation), linewidth=1)
plt.show()
```



In [105]:

```
fig, ax = plt.subplots()
ax.plot(x, np.exp(y_deviation), linewidth=1)
plt.show()
```



In [106]:

```
fig, ax = plt.subplots()
ax.plot(x, random_process, linewidth=1)
plt.show()
```