MSc Statistical Programming 2023 Assessed Practical Assignment

P151

November 23, 2023

0.1 British House Prices

Figure 1 below displays the evolution of House Prices in England over time.

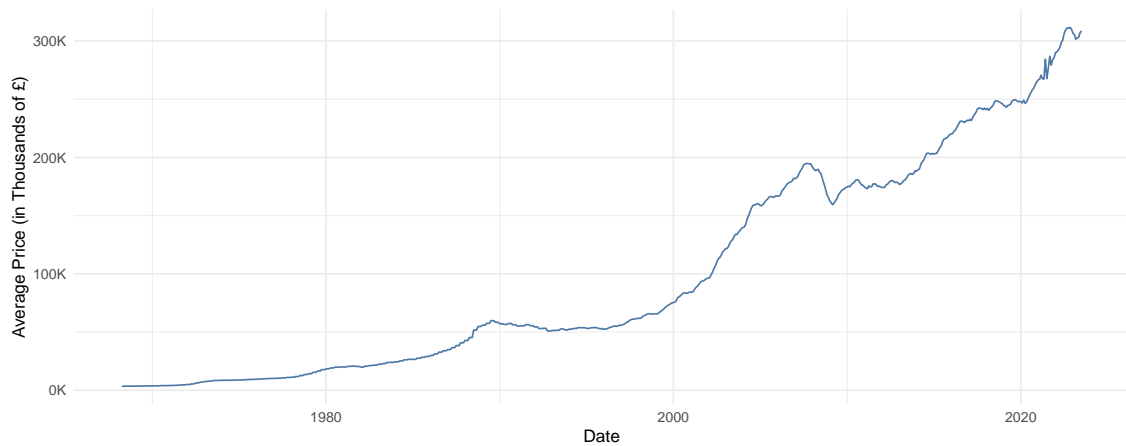Figure 1: Average House Price in England Over Time

Figure 2 below additionally displays the evolution of House Prices in the Oxford Regions as well as England as a whole over time.

Figure 2: Comparison of Average House Prices between England and Oxford Regions
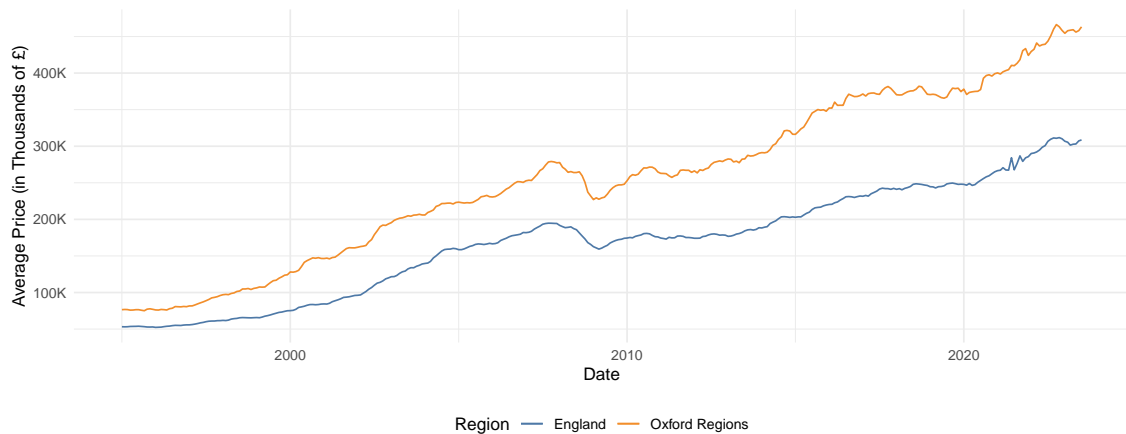
Table 1 below shows that of the ten regions with the highest median of the ratio $\frac{regional\ average\ house\ price}{average\ house\ price\ in\ England}$. Perhaps unsurprisingly, we can see that the ten most expensive regions by this metric are geographically concentrated in London and its suburbs, and that no Oxford Regions are included.

Table 1: Ten Highest Median Ratios of Average House Prices

	Region Name	Median Ratio	Initial Price	Final Price	% Increase	Oxford?
1	Kensington and Chelsea	4.59	182694.83	1344669.00	636.02	FALSE
2	City of Westminster	3.24	133025.28	954356.00	617.42	FALSE
3	Camden	2.92	120932.89	828389.00	585.00	FALSE
4	Hammersmith and Fulham	2.79	124902.86	798537.00	539.33	FALSE
5	City of London	2.56	91448.98	905489.00	890.16	FALSE
6	Richmond upon Thames	2.51	109326.12	770107.00	604.41	FALSE
7	Elmbridge	2.34	106524.00	678279.00	536.74	FALSE
8	Islington	2.32	92516.49	710181.00	667.63	FALSE
9	Wandsworth	2.15	88559.04	632895.00	614.66	FALSE
10	Mole Valley	1.98	101899.76	563083.00	452.59	FALSE

Figure 3 below compares the month-on-month percentage increase in House Prices in England to monthly inflation, defined as the percentage increase in the monthly Consumer Price Index over the period in which data for both is available. We can see that like in most countries, the increase in the average price of a house has drastically outpaced the general cost-of-living increase since the year 2000.

Figure 3: Percentage Increase in House Prices compared to Inflation Rate
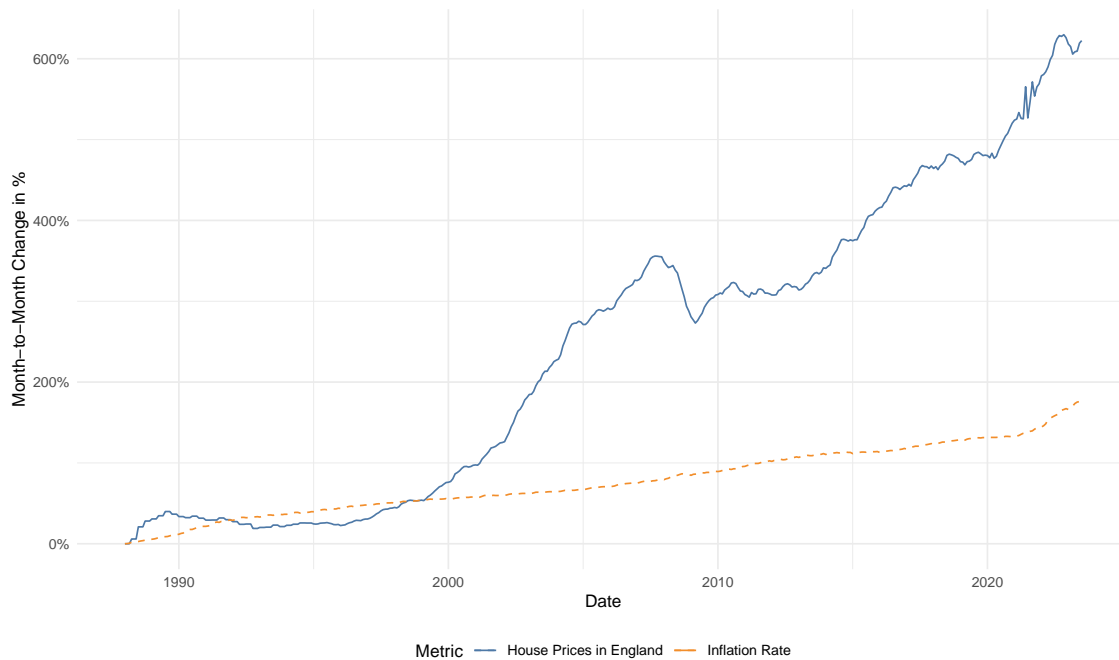
1 Chromosome Painting

1.1 Implementation of `forward` Algorithm

```r
# Function to calculate emission probability (eq. 3)
emission_probability <- function(observed, reference, error = 0.1) {
  return ((1 - error)^(observed == reference) * error^(observed != reference))
}

# matrices alpha and beta , both with K rows and T columns
forward <- function(haps, hap, error = 0.1) {
  K <- nrow(haps)
  T <- ncol(haps)

  # Initialize alpha matrix
  alpha <- matrix(0, nrow = K, ncol = T)

  # Compute initial and emission probabilities
  pi <- 1 / K   # (eq. 1)
  for (k in 1:K) { # (eq. 4)
    alpha[k, 1] <- pi * emission_probability(hap[1], haps[k, 1], error)
  }

  # Induction step to compute (eq. 5)
  for (t in 2:T) {
    for (k in 1:K) {
      transition_sum <- 0
      for (i in 1:K) {
A_ik <- ifelse(i == k, (1 - 0.999) / K + 0.999, (1 - 0.999) / K) # (eq. 2)
transition_sum <- transition_sum + alpha[i, t - 1] * A_ik
}
      b_kt <- emission_probability(hap[t], haps[k, t], error)
      alpha[k, t] <- transition_sum * b_kt
    }
  }

  return(alpha)
}
```

1.2 Unit Test for `forward` Algorithm Implementation

```r
test_that("alpha matrix has expected form when haps and hap always match", {
  K <- 10  # Number of rows in haps
  T <- 15  # Number of columns in haps = length of hap
  e <- 0.1  # Error rate (= default value)

  # Create haps and hap such that they always match (both all 0)
```

```
  haps <- matrix(0, nrow = K, ncol = T)
  hap <- rep(0, T)

  # Run the forward function
  alpha <- forward(haps, hap, error = e)

  # Check first column
  expected_first_column <- rep((1 - e) / K, K)
  expect_equal(alpha[, 1], expected_first_column)

  # Check all other columns
  for (t in 2:T) {
    expect_equal(alpha[, t], alpha[, t - 1] * (1 - e), tolerance = 1e-5)
  }
})

## Test passed
```

1.3   Implementation of `backward` Algorithm

```
backward <- function(haps, hap, error = 0.1) {
  K <- nrow(haps)
  T <- ncol(haps)

  # Initialize beta matrix (eq. 6)
  beta <- matrix(0, nrow = K, ncol = T)
  beta[, T] <- 1   # Set last column to 1

  # Induction step (eq. 7)
  for (t in (T-1):1) {
    for (k in 1:K) {
      sum_transition <- 0
      for (i in 1:K) { # (eq. 2)
        A_ki <- ifelse(k == i, (1 - 0.999) / K + 0.999, (1 - 0.999) / K)
        b_i_t_plus_1 <- emission_probability(hap[t+1], haps[i, t+1], error)
        sum_transition <- sum_transition +
            A_ki * b_i_t_plus_1 * beta[i, t+1]
      }
      beta[k, t] <- sum_transition
    }
  }

  return(beta)
}
```

1.4 Implementation of `gamma` Function

```r
gamma <- function(haps, hap, error = 0.1) {
  K <- nrow(haps)
  T <- ncol(haps)

  # Compute alpha and beta matrices
  alpha <- forward(haps, hap, error)
  beta <- backward(haps, hap, error)

  # Compute normalization factor (denominator)
  norm_factor <- sum(alpha[, T])

  # Initialize gamma matrix
  gamma_matrix <- matrix(0, nrow = K, ncol = T)

  # Update gamma values (eq. 8)
  for (t in 1:T) {
    for (k in 1:K) {
      gamma_matrix[k, t] <- (alpha[k, t] * beta[k, t]) / norm_factor
    }
  }

  return(gamma_matrix)
  }
}
```

1.5 Unit Test for `gamma` Function Implementation

```r
test_that("column sums of gamma matrix sum to 1", {
  set.seed(42)  # For reproducibility
  K <- 10  # Number of rows in haps
  T <- 15  # Number of columns in haps = length of hap
  e <- 0.1  # Error rate (= default value)

  # Create random haps and hap with 0 or 1 entries
  haps <- matrix(sample(0:1, K * T, replace = T), nrow = K, ncol = T)
  hap <- sample(0:1, T, replace = T)

  # Run the gamma function
  gamma_matrix <- gamma(haps, hap, error = e)

   # Check total sum
  total_sum <- sum(gamma_matrix)
  expect_equal(total_sum, 1, tolerance = 1e-5)
})

## Test passed
```

1.6 Computational Complexity of `forward` and `backward` Algorithms

1. The `forward` algorithm has a time complexity of $\mathcal{O}(K^2 \cdot T)$ as for each time step t, it iterates over K states, and within each state again iterates over K states to compute the transition probabilities.

2. The `backward` algorithm has an identical time complexity of $\mathcal{O}(K^2 \cdot T)$ as it performs the same iterations, just in a different oder.

Figures 4 and 5 below show benchmark results for both the forward and backward algorithms, executed 10 times each for all $K \times T$ with $K, L = 5, \ldots, 20$. The fitted lines for the median execution time for each value of K and T, respectively, look polynomial in K and linear in T.
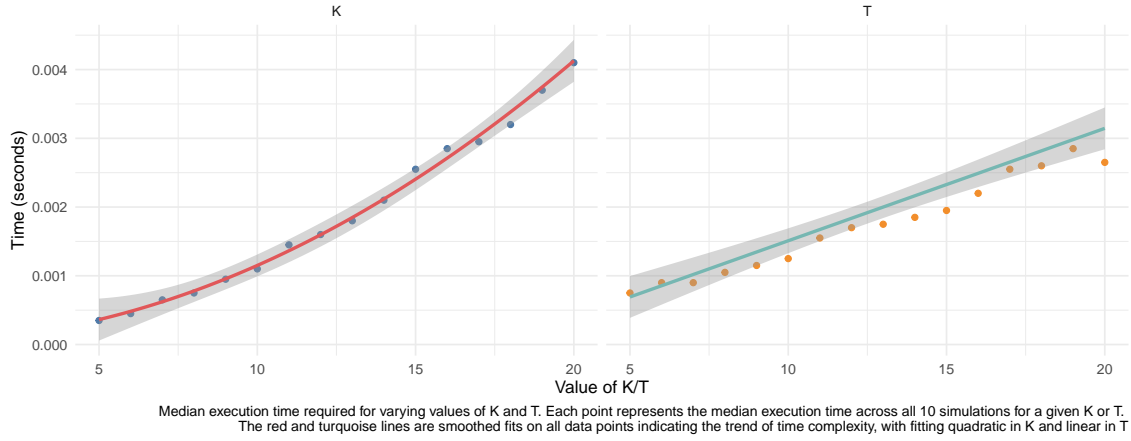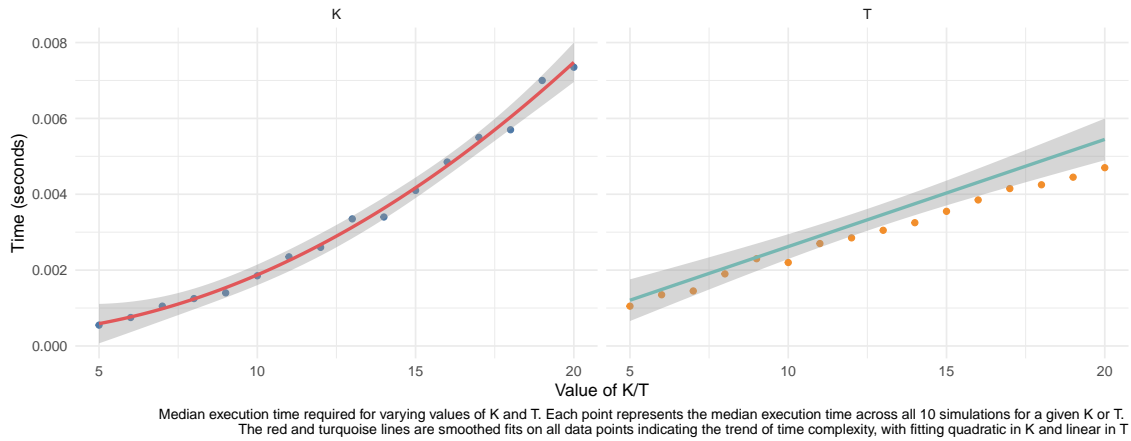
Figure 4: Benchmark Results for `forward` Algorithm

Median execution time required for varying values of K and T. Each point represents the median execution time across all 10 simulations for a given K or T.
The red and turquoise lines are smoothed fits on all data points indicating the trend of time complexity, with fitting quadratic in K and linear in T

Figure 5: Benchmark Results for `backward` Algorithm

Median execution time required for varying values of K and T. Each point represents the median execution time across all 10 simulations for a given K or T.
The red and turquoise lines are smoothed fits on all data points indicating the trend of time complexity, with fitting quadratic in K and linear in T

1.7 Implementation of `forward2` Algorithm

```r
forward2 <- function(haps, hap, error = 0.1) {
  K <- nrow(haps)
  T <- ncol(haps)
  alpha <- matrix(0, nrow = K, ncol = T)
  pi <- 1 / K

  for (k in 1:K) {
    alpha[k, 1] <- pi * emission_probability(hap[1], haps[k, 1], error)
  }

  for (t in 2:T) {
    phi <- sum(alpha[, t - 1]) * (1 - 0.999) / K
    for (k in 1:K) {
      A_ik <- ifelse(k == k, (1 - 0.999) / K + 0.999, (1 - 0.999) / K)
      b_kt <- emission_probability(hap[t], haps[k, t], error)
      alpha[k, t] <- (phi + 0.999 * alpha[k, t - 1]) * b_kt
    }
  }

  return(alpha)
}
```

```r
test_that("forward and forward2 produce the same outputs", {
  set.seed(42)  # For reproducibility
  K <- 10  # Number of rows in haps
  T <- 15  # Number of columns in haps = length of hap
  e <- 0.1  # Error rate (= default value)

  # Generate example data
  haps <- matrix(sample(0:1, K * T, replace = TRUE), nrow = K)
  hap <- sample(0:1, T, replace = TRUE)

  # Run both functions
  alpha_forward <- forward(haps, hap, e)
  alpha_forward2 <- forward2(haps, hap, e)

  # Check if the outputs are equal
  expect_equal(alpha_forward, alpha_forward2, tolerance = 1e-5)
})
```
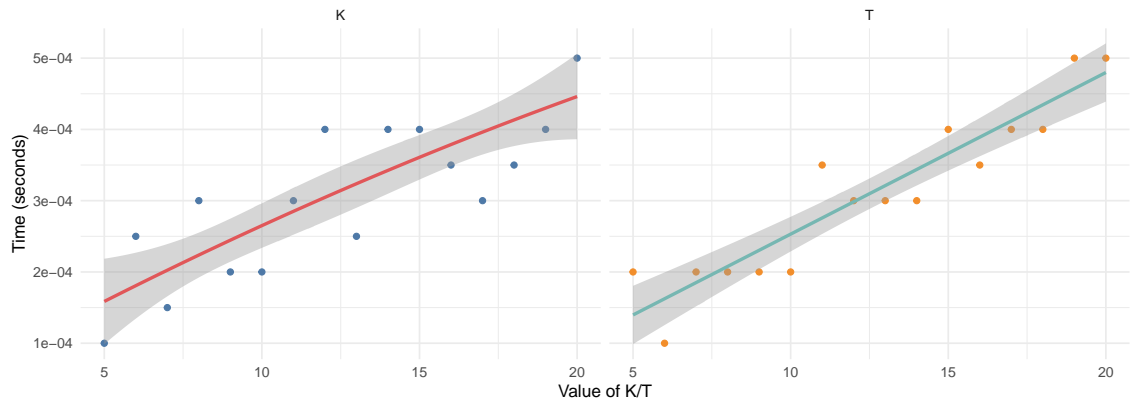
```
## Test passed
```

The `forward2` algorithm has a time complexity of $\mathcal{O}(K \cdot T)$ as it computes the initial probabilities ($\mathcal{O}(K)$) and then for each time step t, it sums over all K states ($\mathcal{O}(K)$) and computes equation (14) ($\mathcal{O}(1)$). Since the main loop runs for each time step T, and within each time step, two $\mathcal{O}(K)$ operations are performed (one for summing alpha and one for the loop over K), the overall complexity of the forward2 function is $\mathcal{O}(K \cdot T)$

Figure 6: Benchmark Results for `forward2` Algorithm

Median execution time required for varying values of K and T. Each point represents the median execution time across all 10 simulations for a given K or T.
The red and turquoise lines are smoothed fits on all data points indicating the trend of time complexity, with fitting quadratic in K and linear in T