

# Algoritmo Snippet-Finder para Descoberta de Padrões Representativos em Séries Temporais

---

# Diferença entre snippet e motif

Um **snippet** é uma subsequência de uma série temporal que é representativa de muitos outros segmentos na série. Ele é selecionado com base em uma medida de similaridade, como a MPdist (Matrix Profile Distance), que leva em consideração a semelhança entre subsequências.

Um **motif** é um par de subsequências de uma série temporal que são altamente similares entre si. Eles são frequentemente usados para detectar padrões repetitivos em dados de séries temporais.

# Diferença entre snippet e motif

Característica	Snippet	Motif
Definição	Subsequência representativa de muitas outras	Pares de subsequências altamente similares
Objetivos	Resumir e representar a série temporal	Detectar padrões repetitivos
Medida de Similaridade	Baseado na similaridade com várias subsequências	Baseado na similaridade entre pares
Aplicação Principal	Resumo e representação de padrões típicos	Identificação de padrões repetitivos
Diversidade	Garante que cada snippet é distinto dos anteriores	Pode encontrar múltiplos motifs similares
Agnosticismo de Domínio	Sim, aplicável sem conhecimento específico	Pode ser ajustado para domínios específicos

# Principais Passos do Algoritmo

1. **Cálculo de Perfis de Distância (MPdist):** Utiliza o MPdist (Matrix Profile Distance) para medir a similaridade entre subsequências de séries temporais, robusto a ruídos, picos e variações lineares.
2. **Seleção de Snippets:**
  - a. Perfis de Distância: Geração de perfis de distância para todas as subsequências não sobrepostas.
  - b. Minimização da Área sob a Curva: Seleção de snippets minimizando a área sob a curva dos perfis de distância combinados.
  - c. Diversidade e Cobertura: Cada novo snippet deve cobrir novas regiões dos dados e ser diverso dos snippets anteriores.
3. **Cálculo de Metadados:** Determina a fração de dados representada por cada snippet e sua localização dentro da série temporal.
  - a. Cobertura: Determina a fração dos dados representada por cada snippet.
  - b. Localização: Identificação da posição de cada snippet na série temporal.

# Método MASS (Mueen's Algorithm for Similarity Search)

O método MASS é um algoritmo para busca de similaridade em séries temporais que utiliza a Transformada Rápida de Fourier (FFT) para acelerar o cálculo da distância entre subsequências de uma série temporal.

No código do snippet finder ele é utilizado para criar os perfis de distâncias.

# Por que o Custo Computacional do MASS é Alto?

Apesar do método utilizado (FFT) ser bem mais rápido  $-O(n \log n)-$  que o cálculo direta das distâncias  $O(n^2 \log n)$ , ainda é necessário calcular a distância para todas as subsequências da série.

O cálculo completo do snippet finder é de complexidade de  $O(n^2 * (n - m / m))$ .

# Código Antigo - Função Principal.

```
def find_snippets(df, series_col, window_size_seconds = 60*5, k=3):
    freq = get_freq(df)
    subseq_size = max(3, int(np.round(window_size_seconds*freq)))
    print(f"Computing {k} snippets for signal with freq: {np.round(freq, 2)}Hz using subsequence size: {subseq_size}...")
    ts = df[series_col].values.astype(float)
    time_idx = df["time"].values
    start = time()
    _, snippets, snippets_profiles, fractions, areas, _ = stumpy.snippets(ts, subseq_size, k)
    end = time()
    print(f"Computation time: {np.round(end-start)}s")
    sorted_args = fractions.argsort()[::-1]

    fractions = fractions[sorted_args]
    snippets = snippets[sorted_args]
    areas = areas[sorted_args]
    snippets_profiles = snippets_profiles[sorted_args]
    curve = snippets_profiles.min(axis=0)

    return subseq_size, snippets, snippets_profiles, fractions, areas, curve
```

1. Calcula a frequência de amostragem.
2. Determina o tamanho da subsequência.
3. Usa o stumpy.snippets para encontrar os snippets.
4. Ordena os snippets com base nas frações de cobertura.

# Sugestão: Fast Summarization of Long Time Series with Graphics Processor

1. Paralelização do Cálculo de Perfis de Distância: O PSF divide o cálculo da similaridade de todas as subsequências em várias etapas, cada uma sendo executada em paralelo.
2. Uso de GPU e CUDA: Acelera o processamento utilizando arquiteturas paralelas, especificamente GPUs com CUDA, para distribuir as tarefas de cálculo intensivo.
3. Desempenho Experimental: Demonstrou um desempenho superior em comparação com o algoritmo original e outras abordagens paralelas diretas em várias séries temporais reais.



## Atualização do Método:

1. Escolha dos Snippets Utilizando a Área Sob a Curva (Método original utilizado no algoritmo do snippet finder)
2. Utilização do K-means + Facilitação dos Profiles
3. Processo de Identificação de Snippets

Ideia: Combinar o k-means para agrupamento e otimização baseada na área sob a curva para identificar subsequências representativas.

# 1. Escolha dos Snippets Utilizando a Área Sob a Curva


Utilizar a área sob a curva das distâncias mínimas acumuladas entre subsequências e perfis para a escolha das subsequências mais representativas.

## Método:

- Para cada subsequência  $S_i$  na série temporal  $T$ , calcula-se a distância euclidiana para todas as outras subsequências  $S_j$  de  $T$ .
- A matriz de distâncias  $EDmatr$  é construída onde cada elemento  $EDmatr[i][j]$  representa a distância entre  $S_i$  e  $S_j$ .
- O perfil de subsequência  $PBA$  é a distância mínima de cada subsequência em relação a todas as outras.
- O perfil de subsequência invertido  $PAB$  é a distância mínima de cada subsequência em relação a todas as outras quando os índices são invertidos.
- Os perfis combinados  $PABBA$  são formados concatenando  $PAB$  e  $PBA$ .
- Ordenando  $PABBA$ , seleciona-se os  $k$  menores valores para formar os perfis mais representativos.

# Código referente

```
def parallel_get_all_profiles(T, m, subseq_len):  
    n = len(T)  
    profiles = []  
    EDmatr = cp.zeros((m - subseq_len + 1, n - m + 1))  
  
    for i in range(m - subseq_len + 1):  
        for j in range(n - m + 1):  
            subseq = T[j:j + subseq_len]  
            segment = T[i:i + subseq_len]  
            EDmatr[i, j] = cp.linalg.norm(subseq - segment)  
  
    allPBA = cp.min(EDmatr, axis=0)  
    allPAB = cp.zeros(m - subseq_len)  
    for i in range(m - subseq_len):  
        allPAB[i] = cp.min(EDmatr[i, :])  
  
    PABBA = cp.concatenate([allPAB, allPBA])  
    sortedPABBA = cp.sort(PABBA)  
    k = int(0.1 * 2 * m)  
    profiles.append(sortedPABBA[:k])  
  
    return profiles
```



## 2. Utilização do K-means + Facilitação dos profiles

Agrupar as subsequências da série temporal em clusters, onde cada cluster representa um conjunto de subsequências similares e então utilizar os centróides desses clusters para representar os perfis de subsequências.

### **Método:**

- A série temporal  $T$  é dividida em segmentos de tamanho fixo  $m$ .
- O algoritmo MiniBatchKMeans(o MiniBatchKMeans tem maior eficiência computacional e menor uso de memória ao lidar com grandes volumes de dados, além de uma rápida convergência e escalabilidade, sem comprometer a qualidade dos clusters.) é aplicado para agrupar esses segmentos em  $numclusters$ .
- Os centróides dos clusters são então usados para calcular distâncias entre eles, criando uma matriz de distâncias  $D$ .

# Aprofundando o ponto 2:

## 1. Clusterização de Subsequências com K-Means:

- Definição de Subsequências: A série temporal  $T$  é dividida em subsequências de comprimento  $m$ .
- Agrupamento: Utilizamos o MiniBatchKMeans para agrupar essas subsequências em `numclusters` clusters. Cada cluster agrupa subsequências similares entre si.

## 2. Centróides dos Clusters:

- Representação dos Clusters: Cada cluster é representado por um centróide, que é a média das subsequências pertencentes àquele cluster.
- Profiles dos Clusters: As distâncias entre esses centróides podem ser vistas como uma forma compacta de representar os profiles dos clusters. Essas distâncias capturam a similaridade entre diferentes grupos de subsequências.

## 3. Utilização dos Profiles para Seleção de Snippets:

- Cálculo das Distâncias entre Centróides: Após agrupar as subsequências, calculamos as distâncias entre os centróides dos clusters, resultando em uma matriz de distâncias  $D$ .
- Seleção de Snippets: Usamos essa matriz de distâncias para selecionar subsequências representativas (snippets). As distâncias mínimas acumuladas ajudam a identificar os snippets que melhor representam a série temporal.

# Código Referente

```
def get_clustered_profiles(T, m, num_clusters=100):  
    segments = np.array([T[i: i + m] for i in range(len(T) - m + 1)])  
    kmeans = MiniBatchKMeans(n_clusters=num_clusters, random_state=0)  
    kmeans.fit(segments)  
    centroids = cp.asarray(kmeans.cluster_centers_)  
  
    n = len(centroids)  
    D = cp.zeros((n, n))  
    for i in range(n):  
        for j in range(n):  
            D[i, j] = cp.linalg.norm(centroids[i] - centroids[j])  
  
    return D, kmeans.labels_
```

# 3. Processo de Identificação de Snippets

## Etapas:

- 1. Clusterização das subsequências usando o algoritmo k-means.**
  - As subsequências da série temporal são agrupadas em clusters.
  - Os centróides dos clusters são calculados.
- 2. Cálculo das distâncias entre os centróides dos clusters.**
  - A matriz de distâncias  $D$  é construída, cada elemento  $D[i][j]$  representa a distância entre os centróides  $i$  e  $j$ .
- 3. Seleção dos snippets que minimizam a área sob a curva das distâncias acumuladas.**
  - Inicia-se com uma matriz  $Q$  cheia de valores infinitos.
  - Para cada iteração, seleciona-se o cluster cujo perfil minimiza a área sob a curva da matriz  $D$  acumulada.
  - A subsequência representativa é escolhida como o primeiro índice do cluster selecionado.
  - As distâncias mínimas são atualizadas e o processo é repetido até que os  $k$  snippets sejam selecionados.

# Código Referente

```
def find_snippets_with_psf(df, series_col, window_size_seconds=60*2, k=3):  
    freq = get_freq(df)  
    subseq_size = max(3, int(np.round(window_size_seconds * freq)))  
    ts = df[series_col].values.astype(float)  
    start = time()  
  
    D, labels = get_clustered_profiles(ts, subseq_size, num_clusters=100)  
  
    snippets, snippet_profiles, areas = [], [], []  
    Q = cp.full((1, D.shape[1]), cp.inf)  
  
    for _ in range(k):  
        minimum_area = cp.inf  
        index_min = -1
```

1

```
    total_area = sum(areas)  
    fractions = [area / total_area for area in areas]  
  
    duration = time() - start  
    print(f"Computation time: {round(duration, 2)}secs")  
  
    return subseq_size, snippets, snippet_profiles, fractions, areas
```

3

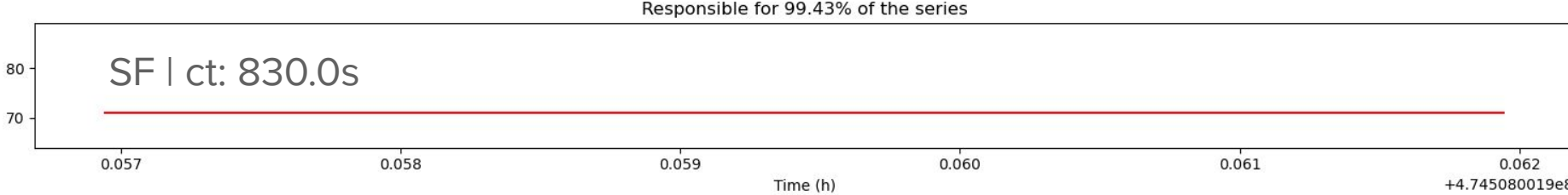
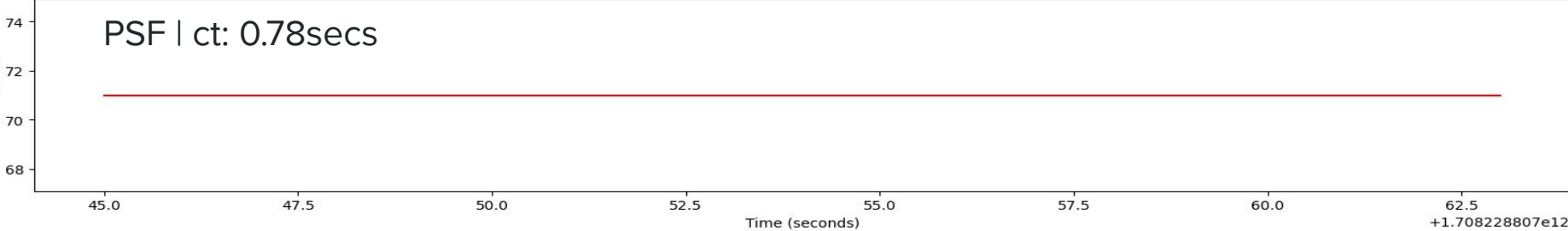
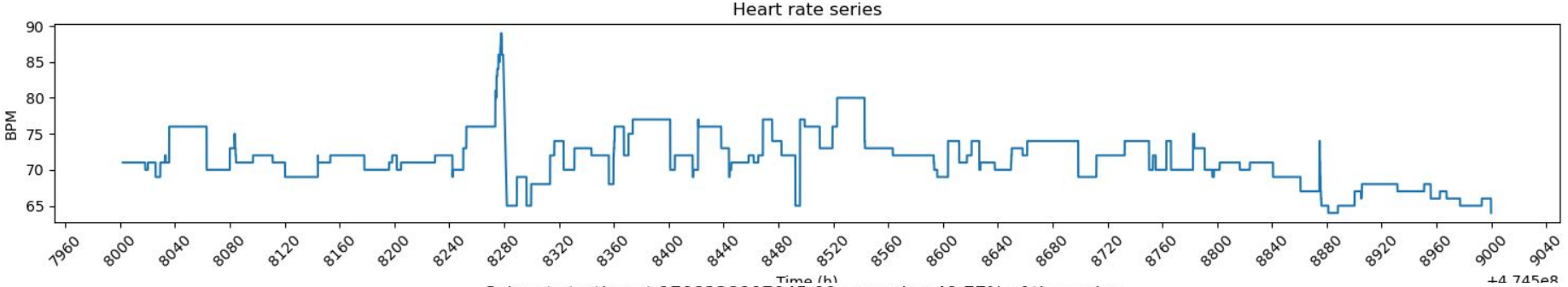
```
    for _ in range(k):  
        minimum_area = cp.inf  
        index_min = -1  
  
        for i in range(D.shape[0]):  
            profile_area = cp.sum(cp.minimum(D[i, :], Q))  
            if profile_area < minimum_area:  
                minimum_area = profile_area  
                index_min = i  
  
        if index_min == -1:  
            break  
  
    Q = cp.minimum(D[index_min, :], Q)  
    indices = np.where(labels == index_min)[0]  
    if indices.size > 0:  
        representative_idx = indices[0] * subseq_size  
        snippets.append((representative_idx, minimum_area.get()))  
        snippet_profiles.append(D[index_min, :])  
        areas.append(minimum_area.get())
```

2

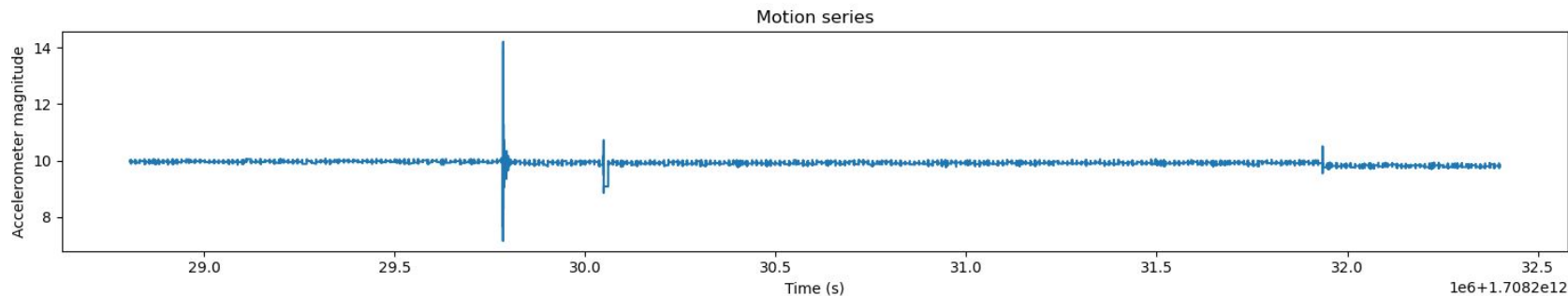


# Comparação PSF X SF Default

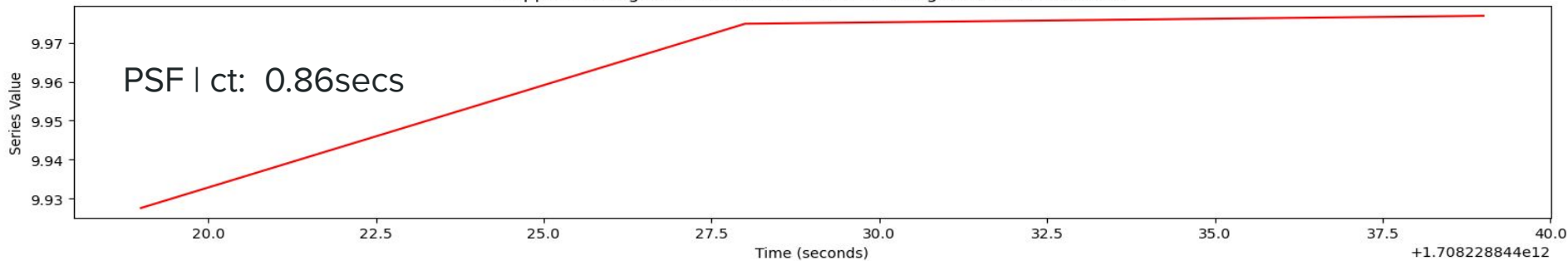
data 2024-02-18 01-00-00.csv



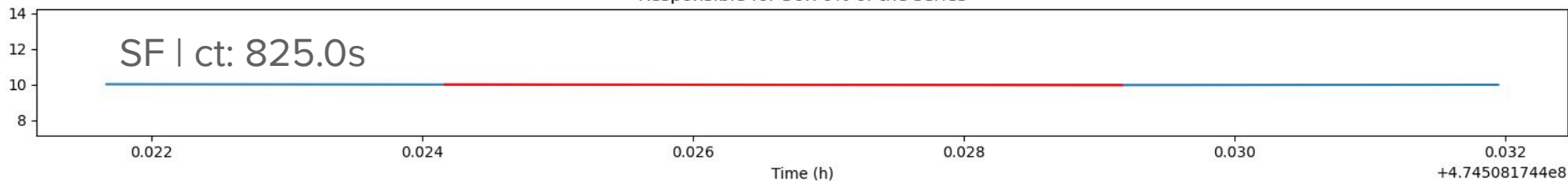
# Comparação PSF X SF Default



Snippet starting at 1708228844319.00s covering 40.48% of the series

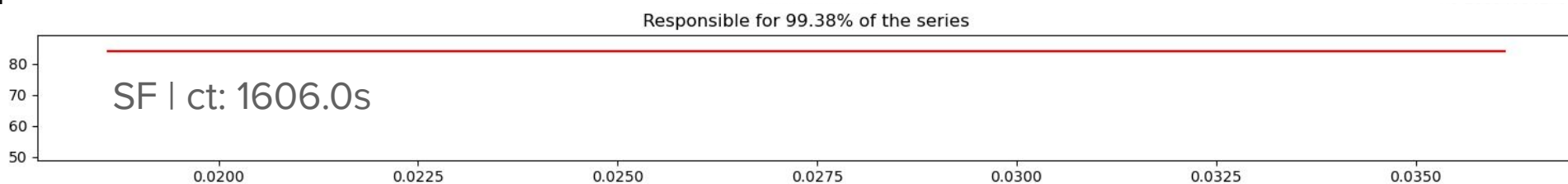
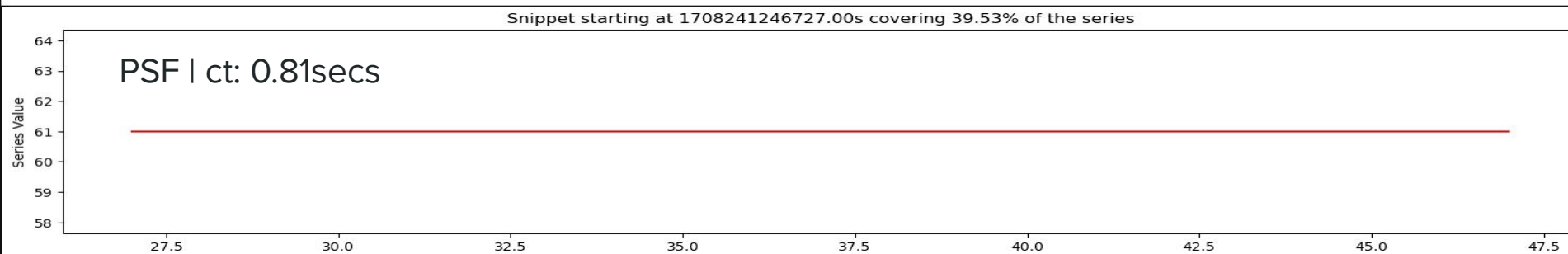
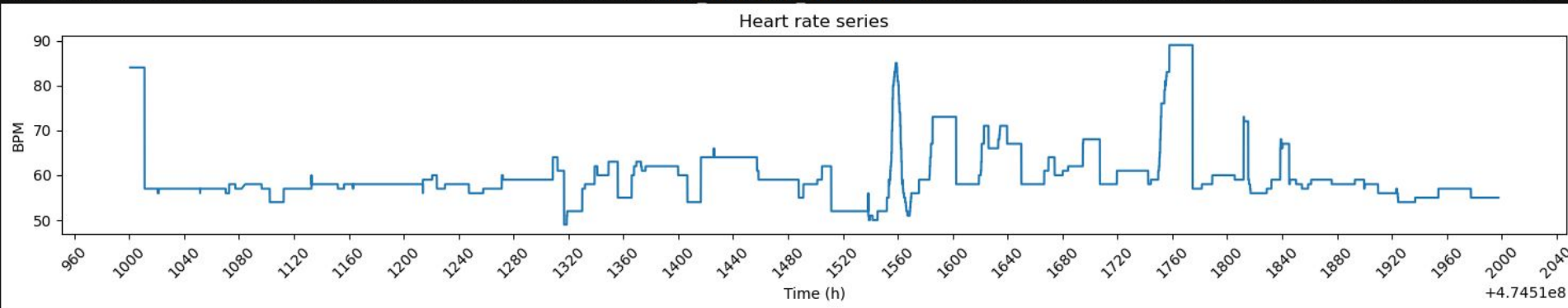


Responsible for 36.76% of the series

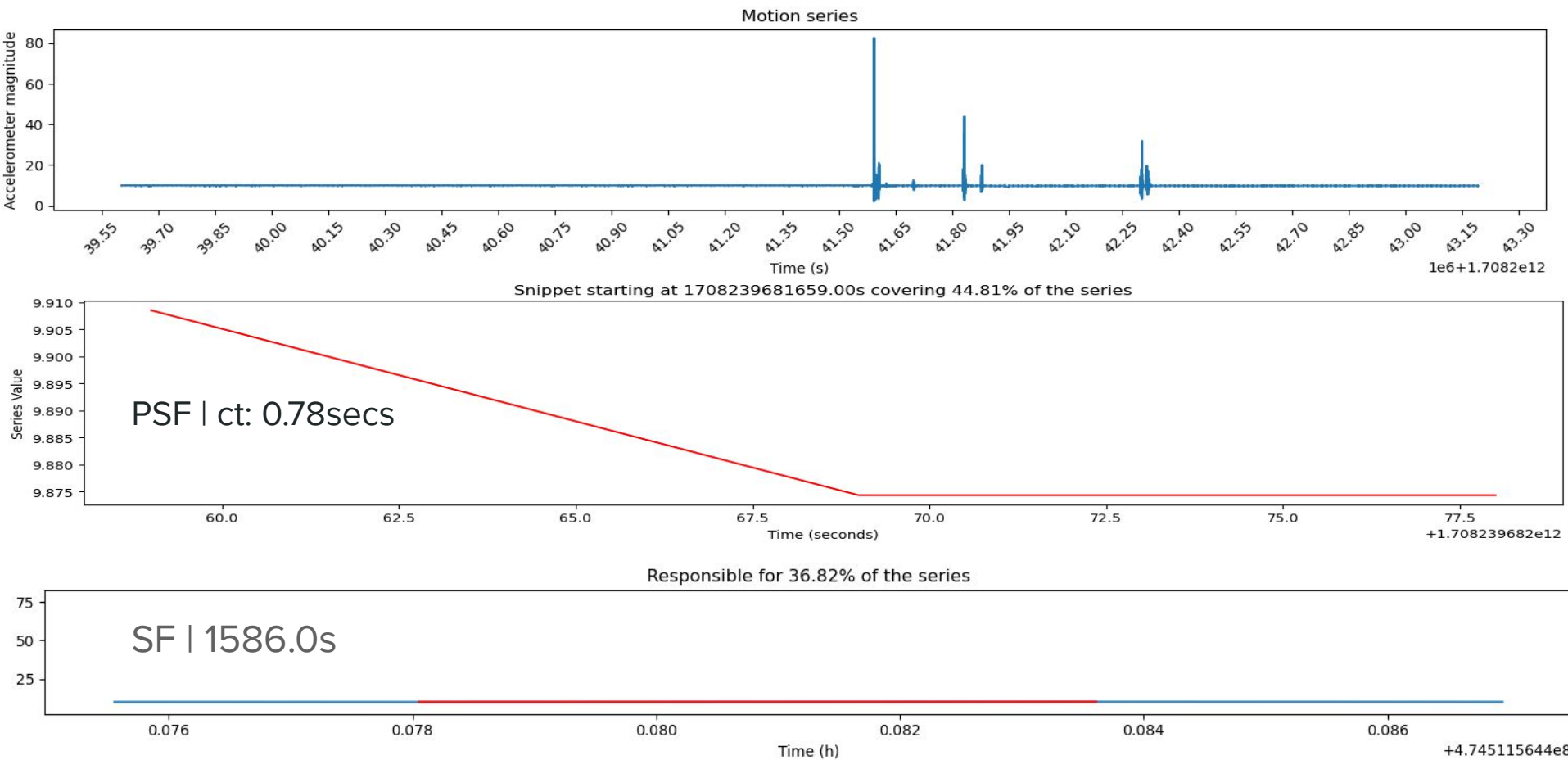


# Comparação PSF X SF Default

data 2024-02-18 04-00-00.csv



# Comparação PSF X SF Default

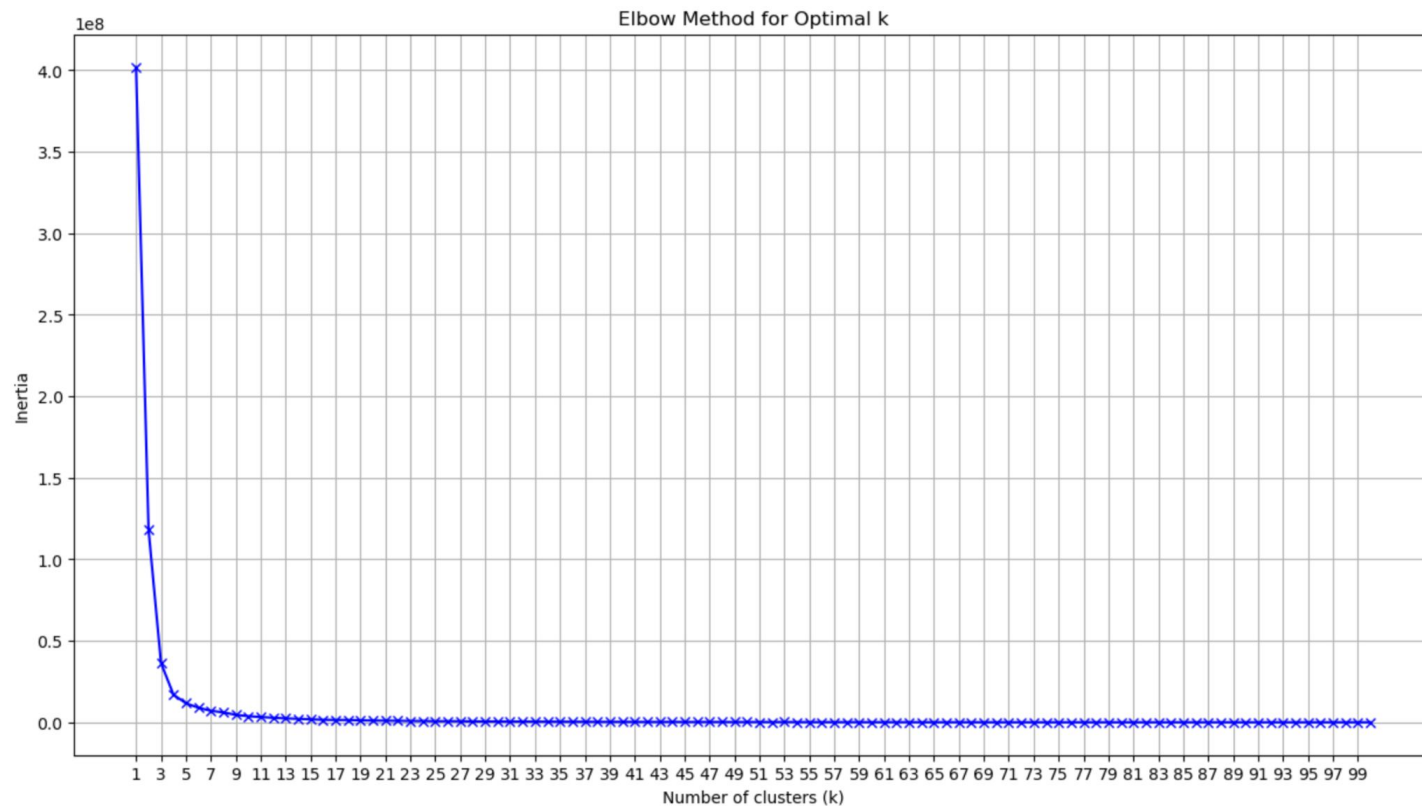


# Últimas atualizações (02/06/2024)

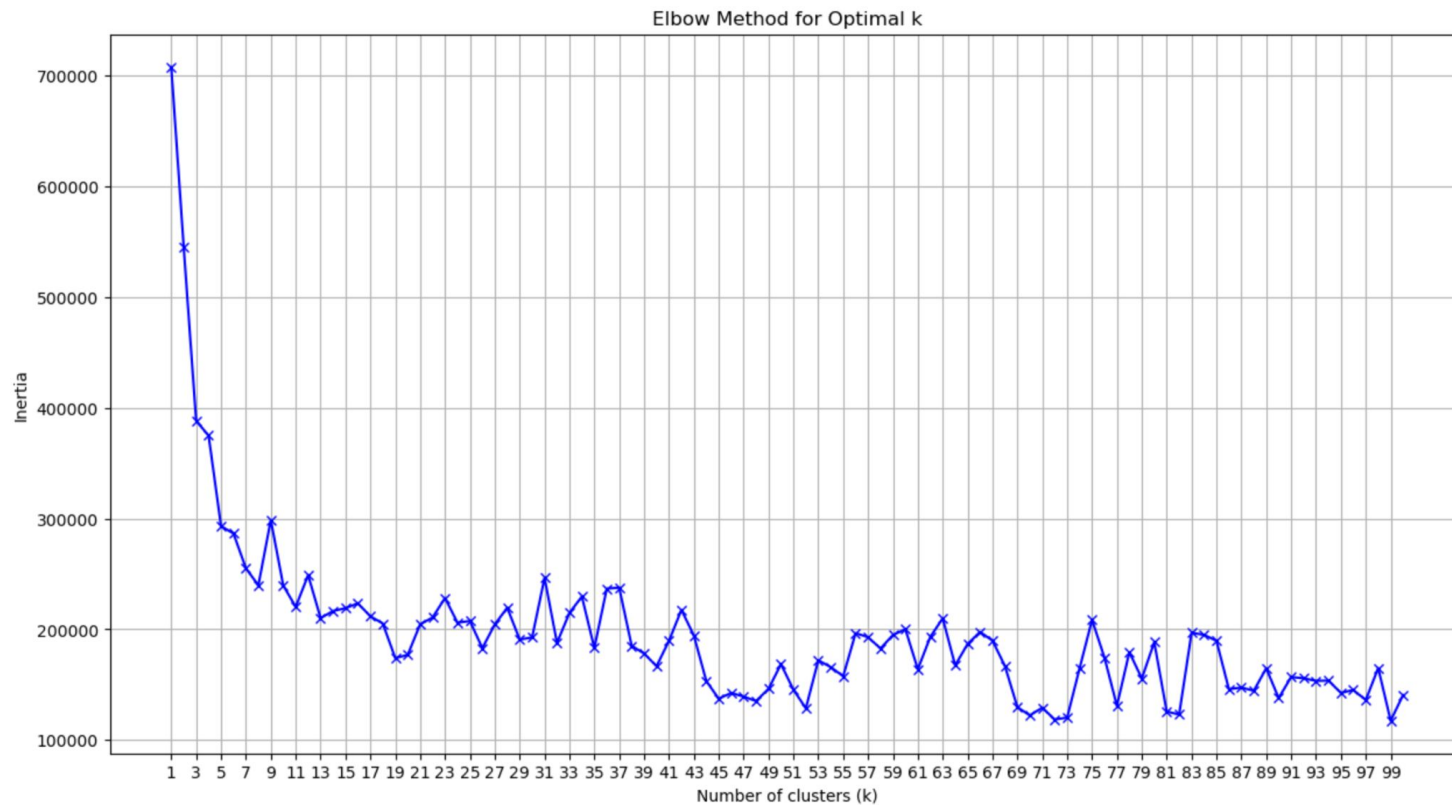
Visto que a implementação com o MiniBatchKmeans foi promissora, os próximos passos foram:

1. Estudo do valor de K via Elbow Method
2. Verificar se há alguma implementação do K-medoids em batch, assim como, minibatchkmeans. Caso não, utilizar o snippet mais próximo do centróide como o centróide de fato.

# Estudo do valor de K (Heart Rate)

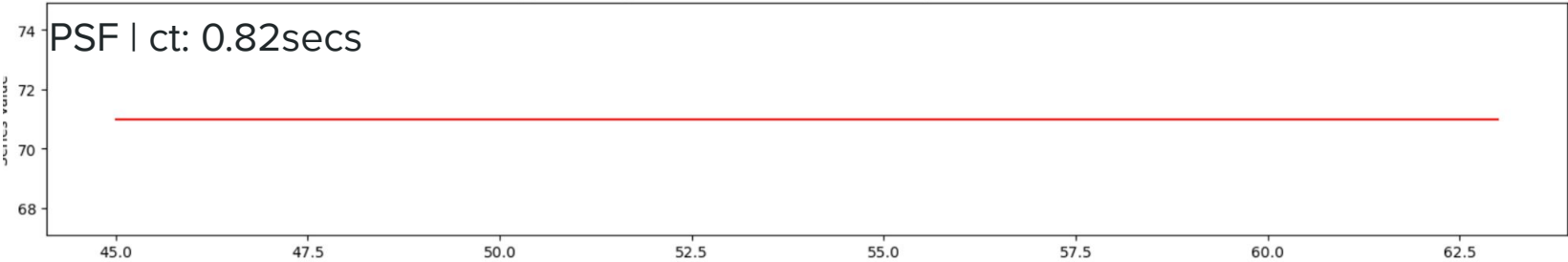
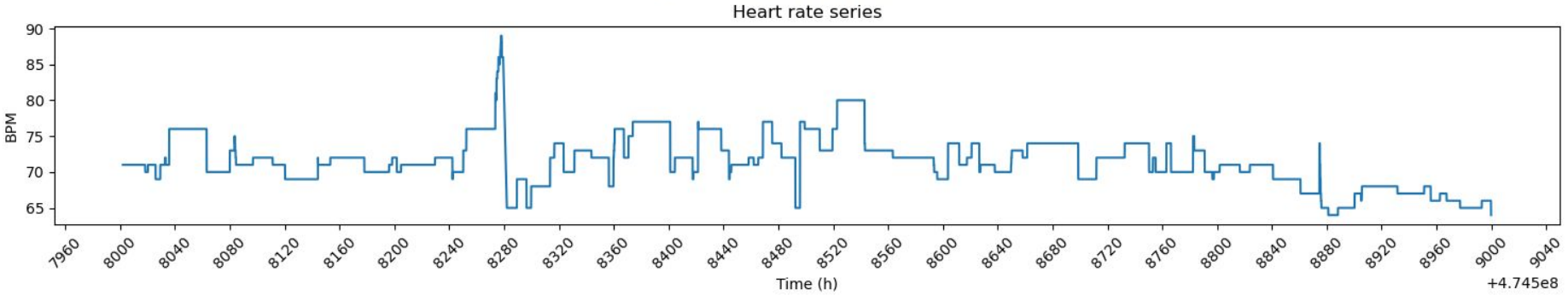


# Estudo do valor de K (Motion Magnitude)



# Comparação PSF (k=4) X SF Default

data 2024-02-18 01-00-00.csv

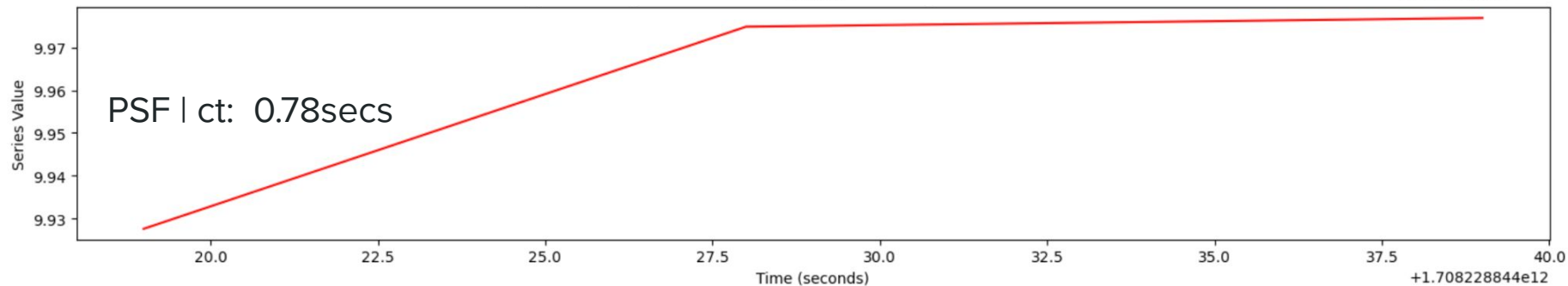
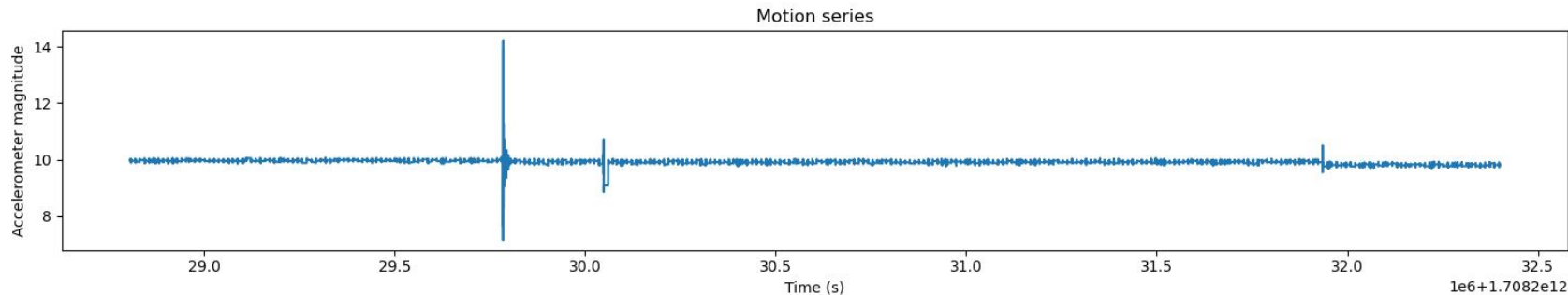


Responsible for 99.43% of the series

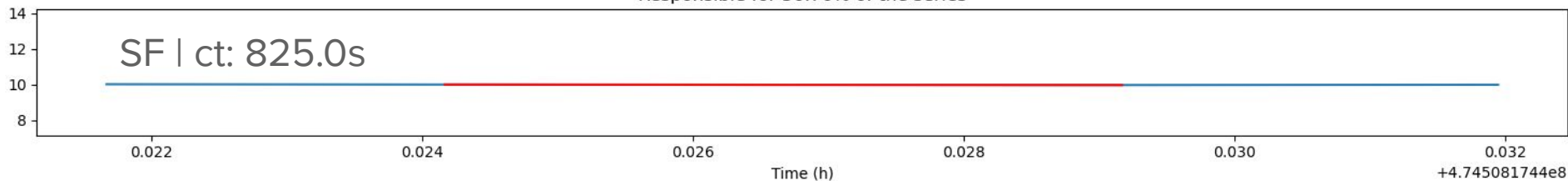




# Comparação PSF (k=4) X SF Default

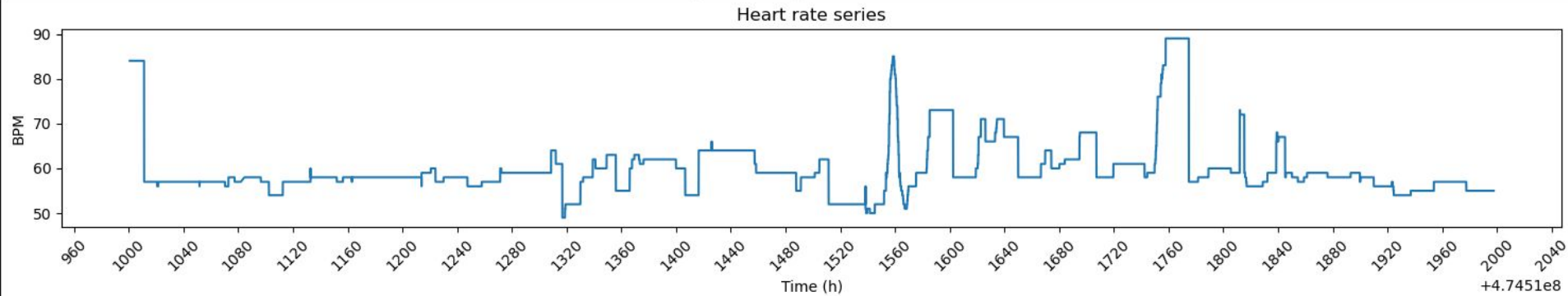


Responsible for 36.76% of the series

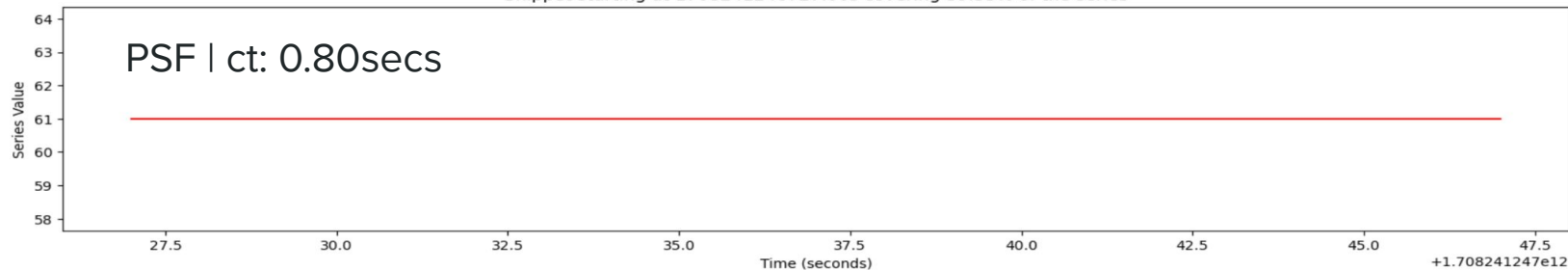


# Comparação PSF (k=4) X SF Default

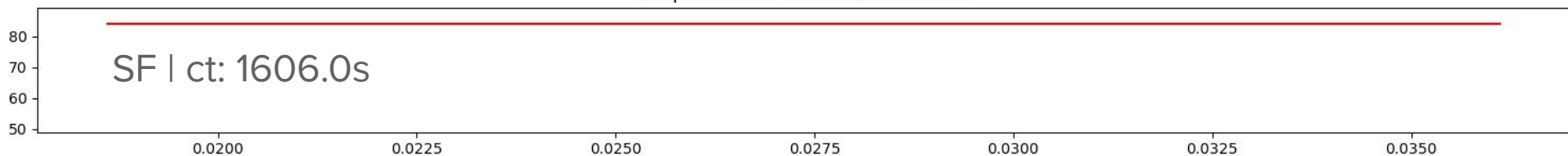
data 2024-02-18 04-00-00.csv



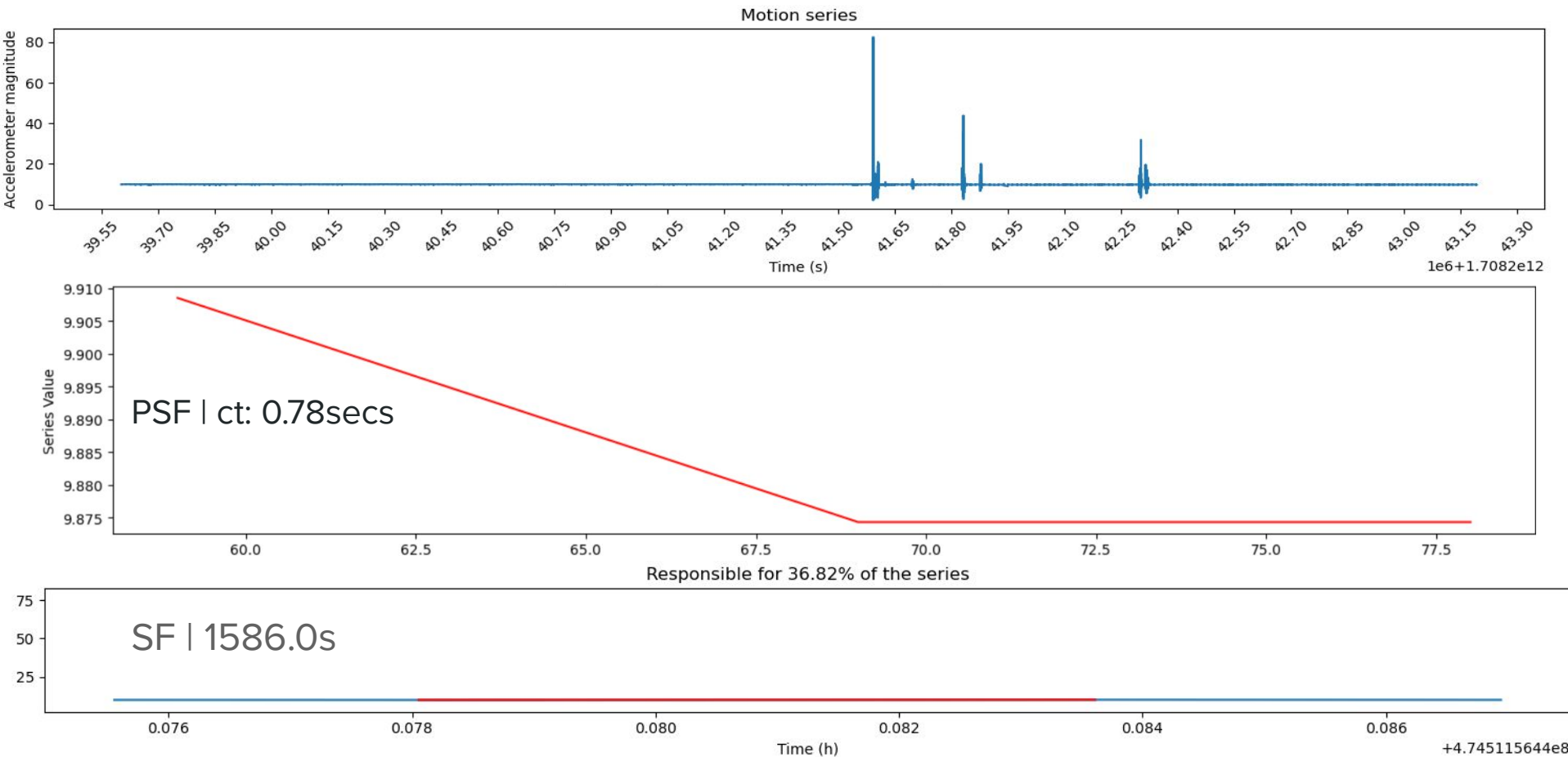
Snippet starting at 1708241246727.00s covering 39.53% of the series



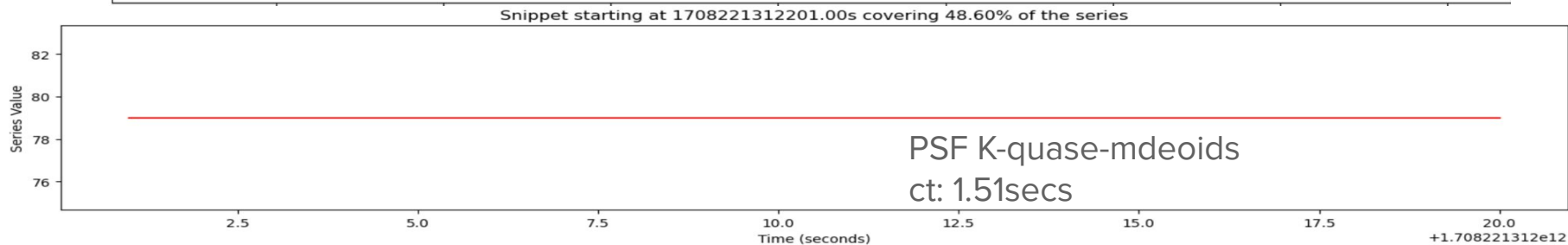
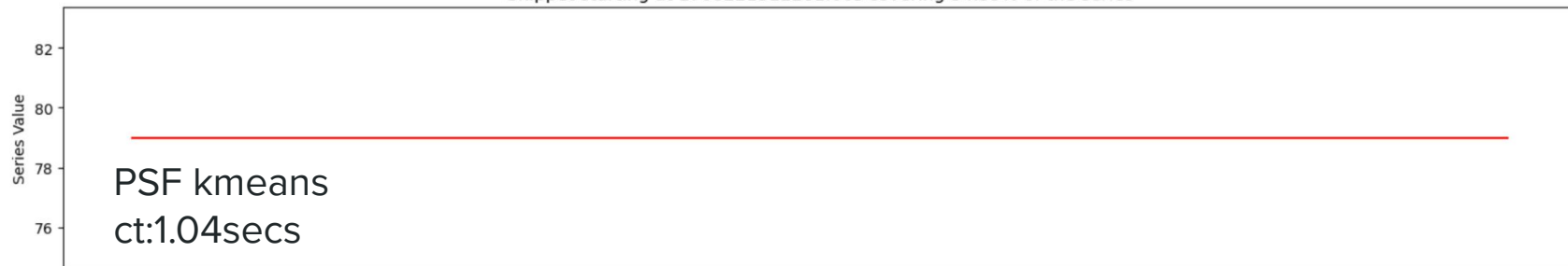
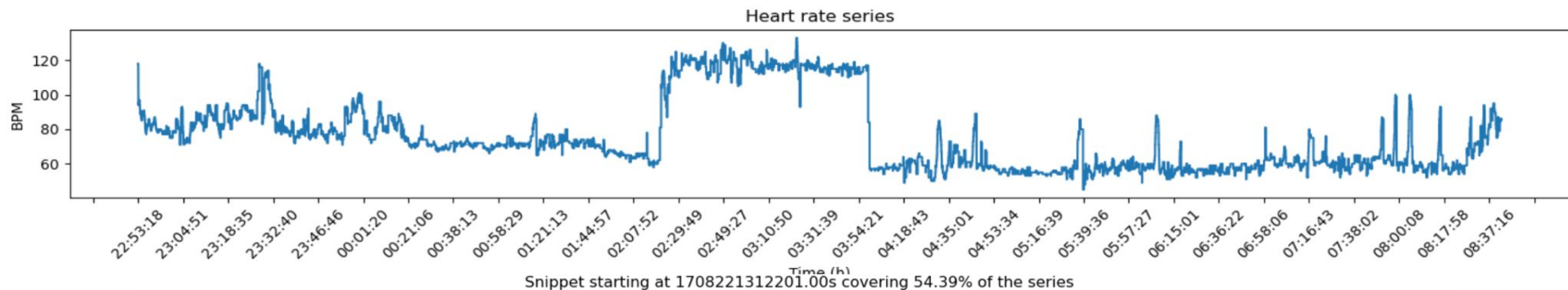
Responsible for 99.38% of the series



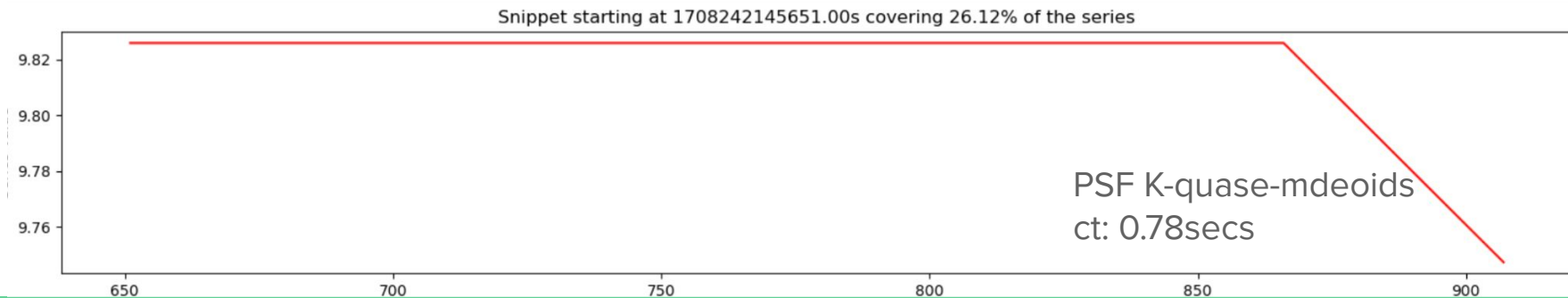
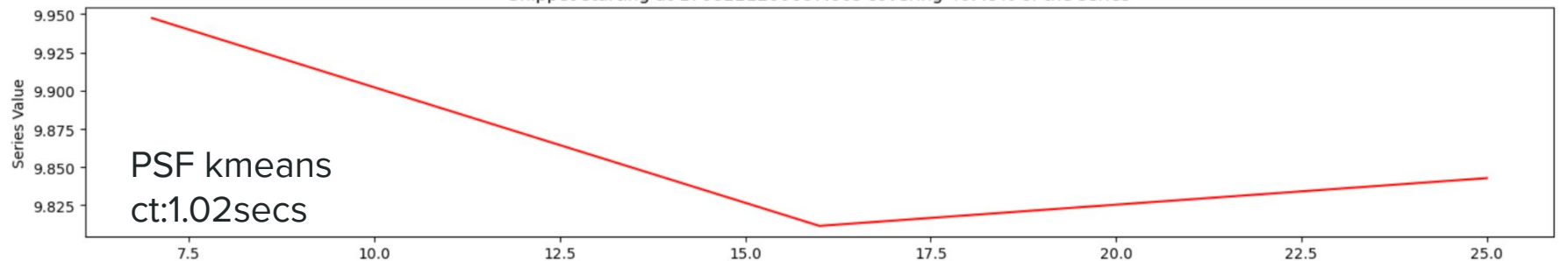
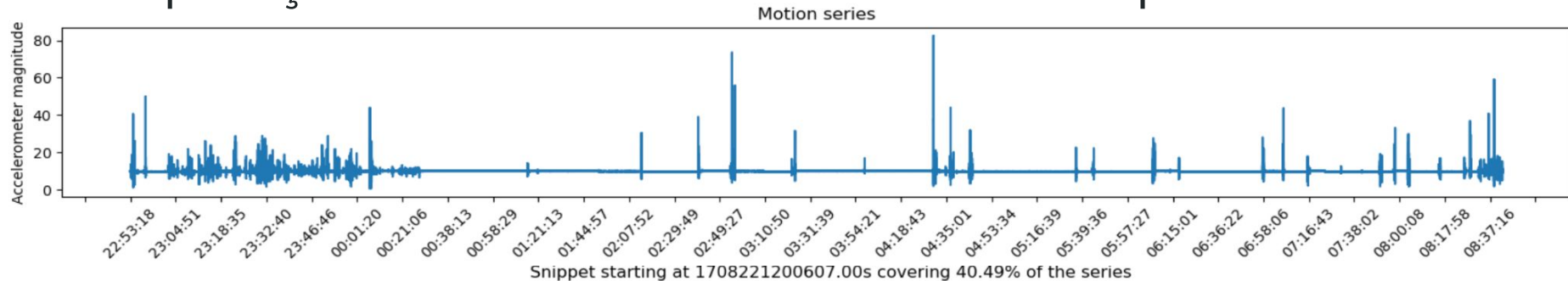
# Comparação PSF (k=4) X SF Default



# Comparação PSF Kmeans Default X PSF K-quase-medoids

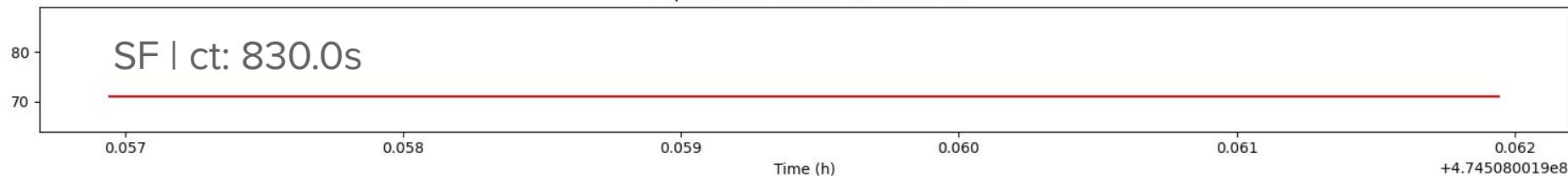
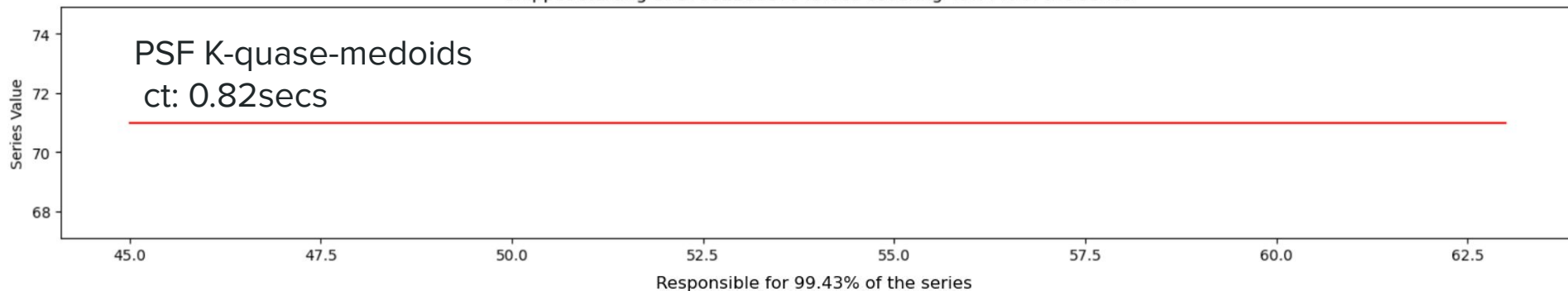
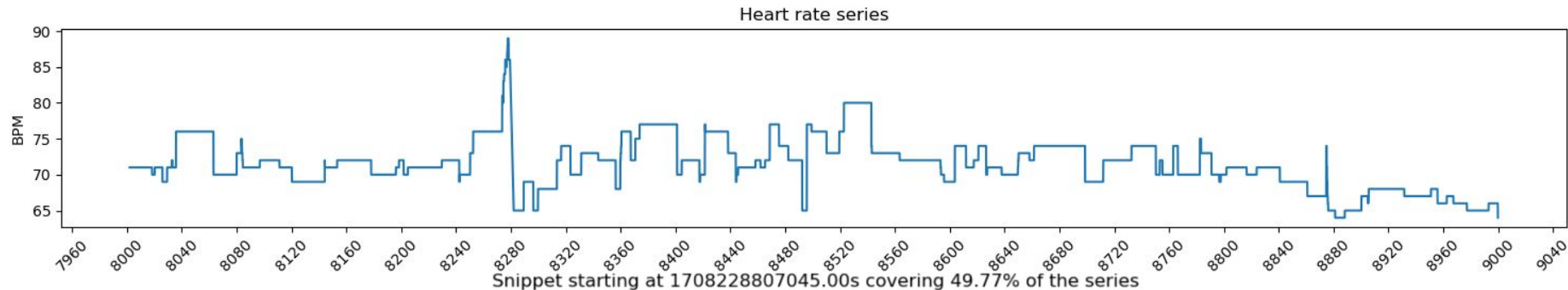


# Comparação PSF Kmeans Default X PSF K-quase-medoids

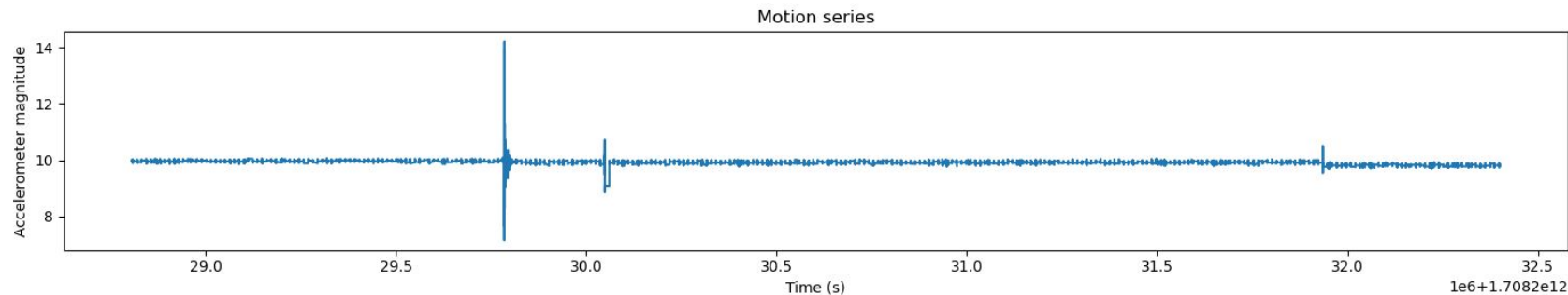


# Comparação PSF k-quase-medoids (k=4) X SF Default

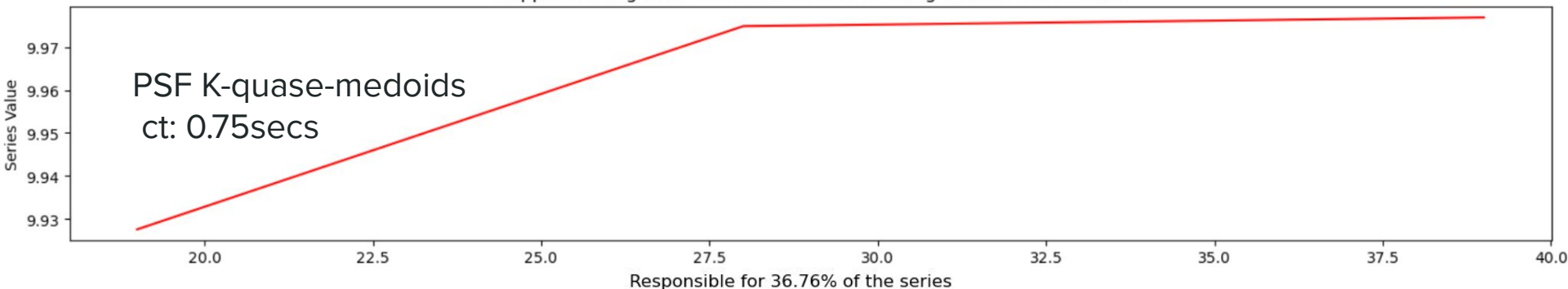
data 2024-02-18 01-00-00.csv



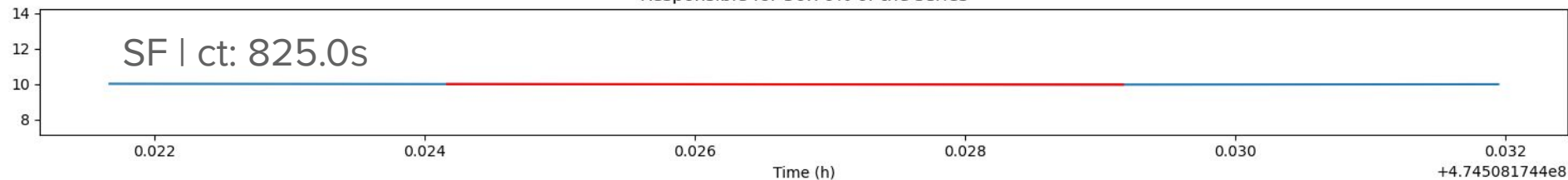
# Comparação PSF k-quase-medoids (k=4) X SF Default



Snippet starting at 1708228844319.00s covering 40.48% of the series

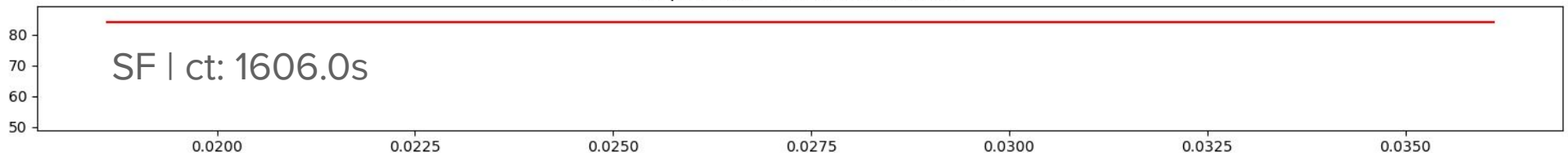
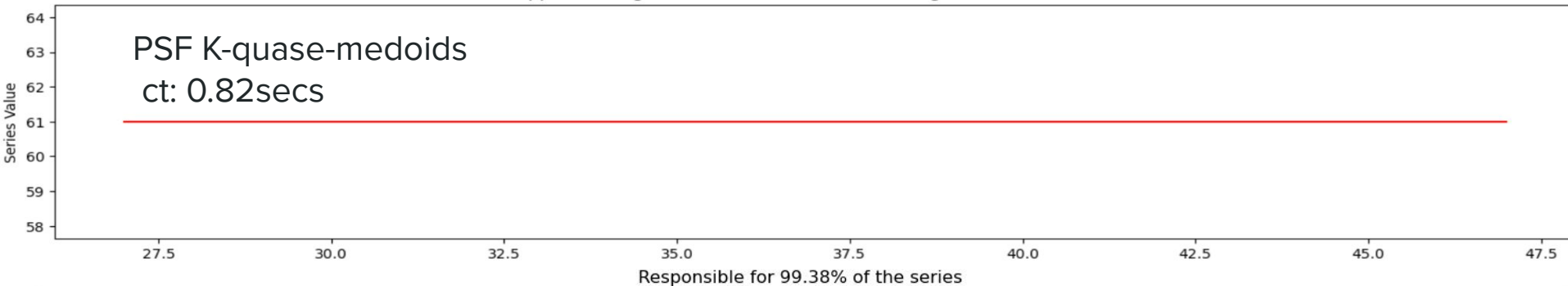
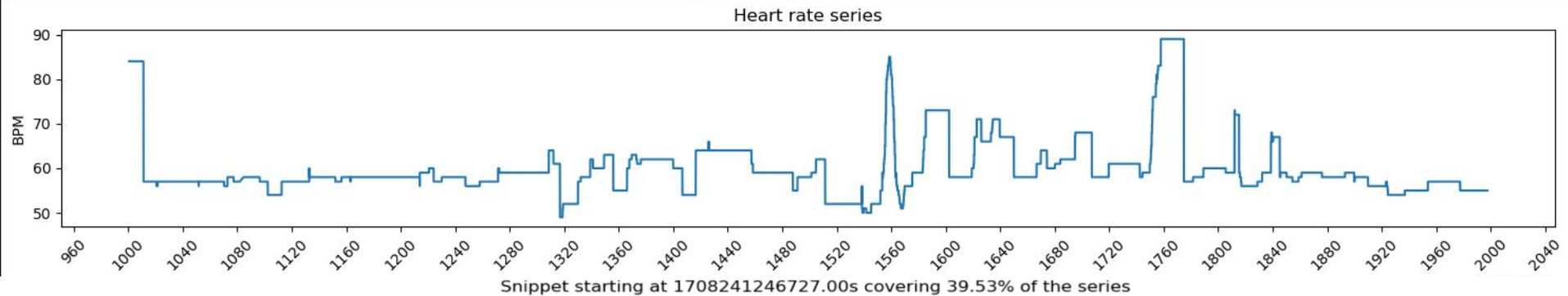


SF | ct: 825.0s



# Comparação PSF k-quase-medoids (k=4) X SF Default

data 2024-02-18 04-00-00.csv





# Comparação PSF k-quase-medoids (k=4) X SF Default

