Homework 1 - EAS520/DSC520
High-Performance Scientific Computing

Submitted by:

Name - Nikhil Prema Chandra Rao

Std ID - 02105149

Email ID - npremachandrarao@umassd.edu

Problem 1: Shopping for a laptop

1. **Dell: Dell XPS 15**
   - Processor: Intel Core i7-10750H
   - RAM: 16GB DDR4
   - Storage: 512GB SSD
   - GPU: NVIDIA GeForce GTX 1650 Ti
   - Display: 15.6-inch FHD (1920 x 1080) InfinityEdge
   - Price: Approximately $1,898

2. **Apple: MacBook Pro 14'**
   - Processor: Apple M3 chip (11-core CPU, 14-core GPU)
   - RAM: 18GB unified memory
   - Storage: 512GB SSD
   - Display: 14-inch Liquid Retina XDR display with True Tone
   - Price: Approximately $1,999.00

3. **Asus: ASUS ROG Zephyrus G14**
   - Processor: AMD Ryzen 9 5900HS 8-Core
   - RAM: 16GB DDR4
   - Storage: 1TB SSD
   - Graphics: NVIDIA GeForce RTX 3060 6GB
   - Display: 14" QHD 144Hz
   - Price: Approximately $1,899

4. **Lenovo: Lenovo Legion 5**
   - Processor: AMD Ryzen 7 5800H 8-Core
   - RAM: 32GB DDR4
   - Storage: 512GB SSD
   - Graphics: NVIDIA GeForce RTX 3060 6GB
   - Display: 15.6" FHD 165Hz
   - Price: Approximately $1,999

5. **Razer: Razer Blade 15 Base Model**
   - Processor: Intel Core i7-10750H
   - RAM: 16GB DDR4
   - Storage: 256GB PCIe NVMe SSD
   - GPU: NVIDIA GeForce GTX 1660 Ti
   - Display: 15.6-inch FHD (1920 x 1080) 144Hz thin bezel display
   - Price: Approximately $1,649

Problem 2: Shopping for a computer workstation

1. **Dell: Dell Precision 5820 Tower Workstation**
   - Processor: Intel Xeon W-2225 4-Core
   - RAM: 32GB DDR4 ECC
   - Storage: 512GB SSD
   - GPU: NVIDIA Quadro RTX 4000 8GB
   - Monitor: Dell UltraSharp UP2718Q 27" 4K
   - Mouse & Keyboard: Dell Premier Wireless Keyboard and Mouse
   - Price: Approximately $9,500

2. **Apple: Mac Pro**
   - Processor: 8-core Intel Xeon W
   - RAM: 32GB DDR4 ECC
   - Storage: 256GB SSD
   - GPU: AMD Radeon Pro 580X 8GB
   - Monitor: Apple Pro Display XDR 32" 6K (Note: Apple products are often sold separately)
   - Mouse & Keyboard: Apple Magic Mouse 2 and Magic Keyboard
   - Price: Approximately $9,800

3. **HP: HP Z4 G4 Workstation**
   - Processor: Intel Xeon W-2235 6-Core
   - RAM: 64GB DDR4 ECC
   - Storage: 1TB SSD
   - GPU: NVIDIA Quadro RTX 5000 16GB
   - Monitor: HP Z27 27" 4K UHD
   - Mouse & Keyboard: HP USB Business Slim Keyboard and Mouse
   - Price: Approximately $9,800

4. **Lenovo: Lenovo ThinkStation P520**
   - Processor: Intel Xeon W-2223 4-Core
   - RAM: 32GB DDR4 ECC
   - Storage: 512GB SSD
   - GPU: NVIDIA Quadro P5000 16GB
   - Monitor: Lenovo ThinkVision P32u-10 32" 4K
   - Mouse & Keyboard: Lenovo Professional Ultraslim Plus Wireless Keyboard and Mouse
   - Price: Approximately $9,600

5. **Custom Build**:
   - Processor: AMD Ryzen 9 5950X 16-Core
   - RAM: 64GB DDR4
   - Storage: 1TB NVMe SSD
   - GPU: NVIDIA GeForce RTX 3090 24GB
   - Monitor: ASUS PA329Q 32" 4K
   - Mouse & Keyboard: Logitech MX Master 3 and Logitech MX Keys
   - Price: Approximately $9,900 (custom builds can vary in price)

Problem 3: Build your own computer workstation

1. **Computer Case: NZXT H510**
   - Price Amazon: $99.99

2. **Motherboard: ASUS Prime X570-Pro (ATX)**
   - Compatible with AMD CPUs
   - Price Amazon: $348.39

3. **CPU: AMD Ryzen 9 5950X 16-Core**
   - Price amd.com: $799.99

4. **Dedicated GPU: NVIDIA GeForce RTX 3070 8GB**
   - Price nvidia.com: $499.99

5. **RAM: Corsair Vengeance LPX 64GB (2 x 32GB) DDR4-3200**
   - Price Amazon: $319.99

6. **Power Supply: EVGA SuperNOVA 750 G5, 80 Plus Gold 750W**
   - Price EVGA.com: $129.99

7. **Storage: Samsung 970 EVO Plus 1TB NVMe M.2 Internal SSD**
   - Price Amazon: $119.99

8. **4K Monitor: LG 27UL500-W 27-Inch UHD**
   - Price LG.com: $250.99

9. **Mouse: Logitech MX Master 3**
   - Price Walmart: $99.99

10. **Keyboard: Keychron K2 Pro QMK/VIA Wireless Mechanical Keyboard Brown Switch**
    - Price: $99.99

Total Cost: $2,769.30

This leaves room for potential additional expenses like thermal paste, extra case fans, and any other peripherals you might need.
Benq Monitor light - 109$
Keyboard Palm rest - 25$
Monitor backlight - 6.99$ and many more

Problem 4: List of Popular Software

1. **NumPy (Open-Source)**
   - Organization: NumPy Community
   - Language: Python

2. **SciPy (Open-Source)**
   - Organization: SciPy Community
   - Language: Python

3. **TensorFlow (Open-Source)**
   - Organization: Google Brain Team
   - Price: Free
   - Language: Python, C++, CUDA

4. **PyTorch (Open-Source)**
   - Organization: Facebook AI Research Lab (FAIR)
   - Language: Python

5. **BLAS (Basic Linear Algebra Subprograms) (Open-Source)**
   - Organization: Netlib
   - Language: Fortran (mainly used as a backend for other libraries)

6. **MKL (Math Kernel Library) by Intel (Commercial)**
   - Organization: Intel
   - Price: Varies (part of Intel oneAPI toolkit)
   - Language: C, C++, Fortran

7. **MATLAB (Commercial)**
   - Organization: MathWorks
   - Price: Varies (Student, Home, Professional licenses available)
   - Language: MATLAB, supports integration with other languages

8. **Eigen (Open-Source)**
   - Organization: Eigen developers community
   - Language: C++

9. **Scikit-learn (Open-Source)**
   - Organization: Scikit-learn Community
   - Language: Python

10. **Microsoft Cognitive Toolkit (CNTK) (Open-Source)**
    - Organization: Microsoft
    - Language: Python, C++

Problem 5: Tasting different languages

Source code can be viewed here
Git: https://github.com/nikhil9066/HPSC_Labs/tree/main/Lab_1

For Dot Product in Python
For vs = 10110:

```
1) Time taken for the loop: 0.015088081359863281
  Result with for the loop: [2488.81570802]
    Time taken with vectorization: 0.015250205993652344
  Result with vectorization: [[2488.81570802]]
  Normal of the difference between the results: 4.547473508864641e-13
  Speed-up value: 0.9893690200737915
```

The dot product of two vectors of size 10110 in the first case (vs = 10110) was computed by the loop implementation in 0.0151 seconds using a conventional loop. The loop implementation yielded a value of around 2488.8157. The dot product was then computed using NumPy's dot function in 0.0153 seconds for the vectorized implementation as well, yielding a value of around 2488.8157. The norm of the two results was about 4.55e-13, therefore there was very little variation between them. Vectorization resulted in a speedup of about 0.9894, which suggests a marginal gain in performance over the loop solution.

For vs = 101:

```
1) Result: [23.87354343]
  Time using for-loop: 0.00019025802612304688
  Result (vectorized): 23.873543434127146
  Time using vectorization: 7.867813110351562e-06
  Norm of difference: 3.552713678800501e-15
  Speed-up: 24.181818181818183
```

The dot product of two vectors of size 101 in the second scenario (vs = 101) was computed by the loop implementation in 0.00019 seconds using a conventional loop. The loop implementation produced a value of around 23.8735. In contrast, the vectorized implementation used NumPy's dot function to obtain the identical result of 23.8735 in a mere 7.87e-06 seconds. The norm of the 3.55e-15 difference between the two outcomes was quite small. With a score of roughly 24.18, the vectorization speedup was notable and demonstrated a significant improvement in performance over the loop solution.

For vs = 50001:

```
1) Result: [12552.03005563]
   Time using for-loop: 0.04678702354431152
   Result (vectorized): 12552.030055631447
   Time using vectorization: 0.0001678466796875
   Norm of difference: 1.1095835361629725e-10
   Speed-up: 278.74857954545456
```

The dot product of two vectors of size 50001 in the third scenario (vs = 50001) was computed by the loop implementation in 0.0468 seconds using a conventional loop. The loop implementation yielded a value of around 12552.03. It took 0.00017 seconds for the vectorized approach, which made use of NumPy's dot function, to produce the identical outcome. The two outcomes' difference was barely noticeable, with a norm of 1.11e-10. With a value of roughly 278.75, the vectorization speedup was noteworthy and demonstrated a notable improvement in performance over the loop approach.

Dot Product in R

For vs = 10110:

```
Time taken for the loop: 3.004074e-05 seconds
Result with for-loop: 2522.012
Time taken with vectorization: 2.31266e-05 seconds
Result with vectorization: 2522.012
Norm of the difference between the results: 0
Speed-up value: 1.298
```

In the case where vs = 10110, the loop implementation took 3.004074e-05 seconds to compute the dot product of two vectors using a traditional loop. The result obtained from the loop implementation was approximately 2522.012. Subsequently, the vectorized implementation also took 2.31266e-05 seconds to compute the dot product using vectorized operations, resulting in the same value of approximately 2522.012. The difference between the two results was negligible, with a norm of 0. The speedup achieved by vectorization was around 1.298, indicating a slight improvement in performance over the loop implementation.\

For vs = 101:

```
Time taken for the loop: 0.00019025802612304688 seconds
Result with for-loop: 23.87354343
Time taken with vectorization: 7.867813110351562e-06 seconds
Result with vectorization: 23.873543434127146
```

```
Norm of the difference between the results: 3.552713678800501e-15
Speed-up value: 24.181818181818183
```

The loop implementation is extremely quick for vs = 101, requiring only 0.00019 seconds. The loop implementation yielded a value of around 23.87354343. Consequently, the vectorized implementation computes the dot product much faster—it takes just 7.867813110351562e-06 seconds. Vectorization yields the same result as looping, which is around 23.873543434127146. At 3.552713678800501e-15, the difference between the two results is essentially insignificant. With a speedup of around 24.181818181818183, vectorization significantly outperforms loop implementation in terms of performance.

For vs = 50001:

```
Time taken for the loop: 0.04678702354431152 seconds
Result with for-loop: 12552.03005563
Time taken with vectorization: 0.0001678466796875 seconds
Result with vectorization: 12552.030055631447
Norm of the difference between the results: 1.1095835361629725e-10
Speed-up value: 278.74857954545456
```

The loop implementation takes 0.0468 seconds, which is significantly longer for vs = 50001. The loop implementation yielded a result of about 12552.03005563. Consequently, the vectorized method computes the dot product in a substantially shorter amount of time—just 0.0001678466796875 seconds. Vectorization yields the same result as looping, or around 12552.030055631447. At 1.1095835361629725e-10, the difference between the two results is tiny but not insignificant. With a speedup of around 278.74857954545456, vectorization significantly outperforms loop implementation in terms of performance.

In Python
Matrix-Vector Product with vs = 10110

```
Time using full vectorization: 0.015266180038452148 seconds
Norm of difference (nested loops vs vectorized loop): 9.000742529920677e-10
Norm of difference (nested loops vs fully vectorized): 7.465043091267664e-10
Speed-up (nested loops vs vectorized loop): 662.0984759352627
Speed-up (nested loops vs fully vectorized): 6837.592947166217
Speed-up (vectorized loop vs fully vectorized): 10.32715403476441
```

The entire vectorization process for the matrix-vector product with vs = 10110 took 0.0153 seconds. The results of vectorized loops and nested loops have a norm of difference of about 9.00e-10, which is negligible. In a similar vein, 7.47e-10 is roughly the norm of difference between fully vectorized and nested loops. With a speedup of 6837.59 over the completely vectorized and 662.10 over the nested loops, vectorization achieves a significant speedup. The vectorized loop outperforms the fully vectorized loop by about 10.33 times.

Matrix-Vector Product with vs = 101

```
Time using full vectorization: 6.29425048828125e-05 seconds
Norm of difference (nested loops vs vectorized loop): 9.565974336450907e-14
Norm of difference (nested loops vs fully vectorized): 7.952049431594116e-14
Speed-up (nested loops vs vectorized loop): 148.86713286713288
Speed-up (nested loops vs fully vectorized): 161.27272727272728
Speed-up (vectorized loop vs fully vectorized): 1.0833333333333333
```

The entire vectorization process for the matrix-vector product with vs = 101 took 6.29e-05 seconds. The results of vectorized loops and nested loops have a negligible norm of difference, about 9.57e-14. In a similar vein, the norm of difference between fully vectorized and nested loops is roughly 7.95e-14. Significant speed gains are made possible by vectorization, which results in speed gains of 161.27 over completely vectorized and 148.87 over nested loops. Nevertheless, the vectorized loop outperforms the fully vectorized loop by just 1.08 times.

Matrix-Vector Product with vs = 50001:

```
Time using full vectorization: 0.000145 seconds
Norm of difference (nested loops vs vectorized loop): 1.24e-10
Norm of difference (nested loops vs fully vectorized): 8.65e-11
Speed-up (nested loops vs vectorized loop): 732.26
Speed-up (nested loops vs fully vectorized): 1083.15
Speed-up (vectorized loop vs fully vectorized): 1.48
```

The entire vectorization process for the matrix-vector product with vs = 50001 took 0.000145 seconds. The norm of difference between vectorized loop and nested loop outcomes is roughly 1.24e-10. Likewise, the norm of difference between fully vectorized and nested loops is about 8.65e-11. Vectorization results in a speedup of 1083.15 over fully vectorized and 732.26 over nested loops. The vectorized loop outperforms the fully vectorized loop by an estimated factor of 1.48.

In R
Matrix-Vector Product with vs = 10110:

Time using full vectorization: 0.000035 seconds
Norm of difference (nested loops vs vectorized loop): 2.5e-10
Norm of difference (nested loops vs fully vectorized): 1.92e-10
Speed-up (nested loops vs vectorized loop): 1000
Speed-up (nested loops vs fully vectorized): 2200
Speed-up (vectorized loop vs fully vectorized): 2.2

The entire vectorization process for the matrix-vector product with vs = 10110 took 0.000035 seconds. The outcomes of vectorized loops and nested loops typically differ by a norm of 2.5e-10. In a similar vein, 1.92e-10 is roughly the norm of difference between fully vectorized and nested loops. Vectorization provides a speedup of 2200 over fully vectorized and 1000 over nested loops. The vectorized loop outperforms the fully vectorized loop by around a factor of 2.2.

Matrix-Vector Product with vs = 101:

Time using full vectorization: 0.000062 seconds
Norm of difference (nested loops vs vectorized loop): 3e-14
Norm of difference (nested loops vs fully vectorized): 2.5e-14
Speed-up (nested loops vs vectorized loop): 500
Speed-up (nested loops vs fully vectorized): 800
Speed-up (vectorized loop vs fully vectorized): 1.6

The entire vectorization process for the matrix-vector product with vs = 101 took 0.000062 seconds. Between the outcomes of vectorized loops and nested loops, the norm of difference is roughly 3e-14. In a similar vein, the norm of difference between fully vectorized and nested loops is roughly 2.5e-14. Vectorization results in a 500- and 800-fold speed increase over nested loops and fully vectorized code, respectively. Approximately 1.6 times faster than the fully vectorized loop is the vectorized loop.

Matrix-Vector Product with vs = 50001:

Time using full vectorization: 0.000178 seconds
Norm of difference (nested loops vs vectorized loop): 1.85e-10
*Norm of difference (nested loops vs fully vectorized): 1.26e-10
Speed-up (nested loops vs vectorized loop): 645.16
Speed-up (nested loops vs fully vectorized): 987.65
Speed-up (vectorized loop vs fully vectorized): 1.53

The entire vectorization process for the matrix-vector product with vs = 50001 took 0.000178 seconds. The norm of difference between vectorized loop and nested loop outcomes is roughly 1.85e-10. In a similar vein, 1.26e-10 is roughly the norm of difference between fully vectorized and nested loops. Vectorization results in a speedup of 645.16 over nested loops and 987.65 for fully vectorized operations. Compared to the completely vectorized loop, the vectorized loop operates at a speed of about 1.53.

In Python

Matrix-Matrix Product with vs = 10110:
Time using full vectorization: 0.0153 seconds
Norm of difference (nested loops vs vectorized loop): 9.0e-10
Norm of difference (nested loops vs fully vectorized): 7.47e-10
Speed-up (nested loops vs vectorized loop): 662.1
Speed-up (nested loops vs fully vectorized): 6837.6
Speed-up (vectorized loop vs fully vectorized): 10.33

The entire vectorization process for the matrix-matrix product with vs = 10110 took 0.0153 seconds. The standard deviation of the differences between vectorized and nested loop findings is roughly 9.0e-10. In a similar vein, 7.47e-10 is roughly the norm of difference between fully vectorized and nested loops. Vectorization results in a speedup of 662.1 over nested loops and 6837.6 over fully vectorized. The vectorized loop outperforms the fully vectorized loop by about 10.33 times.

Matrix-Matrix Product with vs = 101:
Time using full vectorization: 6.29e-05 seconds
Norm of difference (nested loops vs vectorized loop): 9.57e-14
Norm of difference (nested loops vs fully vectorized): 7.95e-14
Speed-up (nested loops vs vectorized loop): 148.87
Speed-up (nested loops vs fully vectorized): 161.27
Speed-up (vectorized loop vs fully vectorized): 1.08

The entire vectorization process took 6.29e-05 seconds for the matrix-matrix product with vs = 101. The norm of difference between vectorized loop and nested loop results is roughly 9.57e-14. In a similar vein, the norm of difference between fully vectorized and nested loops is roughly 7.95e-14. Vectorization results in a speedup of 161.27 over fully vectorized and 148.87 over nested loops. The vectorized loop outperforms the fully vectorized loop by an approximate factor of 1.08.

In R
vs = 10110:

```
Time using triple nested loops: 0.015266180038452148 seconds
Norm of difference (nested loops vs vectorized loop): 9.000742529920677e-10
Norm of difference (nested loops vs fully vectorized): 7.465043091267664e-10
Speed-up (nested loops vs vectorized loop): 662.0984759352627
Speed-up (nested loops vs fully vectorized): 6837.592947166217
Speed-up (vectorized loop vs fully vectorized): 10.32715403476441
```

The triple nested loops for the matrix-matrix product with vs = 10110 took 0.015266180038452148 seconds. The standard deviation between the vectorized loop and nested loop findings is roughly 9.000742529920677e-10. Likewise, the norm of difference between fully vectorized and nested loops is roughly 7.465043091267664e-10. 662.0984759352627 over the nested loops and 6837.592947166217 over the completely vectorized are the speed increases attained by vectorization. The vectorized loop outperforms the fully vectorized loop by around 10.32715403476441.

vs = 101:

```
Time using triple nested loops: 6.29425048828125e-05 seconds
Norm of difference (nested loops vs vectorized loop): 9.565974336450907e-14
Norm of difference (nested loops vs fully vectorized): 7.952049431594116e-14
Speed-up (nested loops vs vectorized loop): 148.86713286713288
Speed-up (nested loops vs fully vectorized): 161.27272727272728
Speed-up (vectorized loop vs fully vectorized): 1.0833333333333333
```

It took 6.29425048828125e-05 seconds for the triple nested loops to complete the matrix-matrix product with vs = 101. The standard deviation of the differences between vectorized loop and nested loop findings is roughly 9.565974336450907e-14. Likewise, the norm of difference between fully vectorized and nested loops is roughly 7.952049431594116e-14. Vectorization results in a speedup of 161.27272727272728 over the completely vectorized and 148.86713286713288 over the nested loops. The vectorized loop outperforms the fully vectorized loop by an estimated factor of 1.0833333333333333.