Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #45413

Leveraging Accelerometers for Teaching Numerical Differentiation and Integration

Dr. Vivek Singhal, University of Wisconsin - Stout
Dr. Devin R. Berg, University of Wisconsin - Stout

Devin Berg is a Professor of mechanical engineering in the Engineering and Technology Department at the University of Wisconsin - Stout.

Leveraging Accelerometers for Teaching Numerical Differentiation and Integration

This paper investigates the innovative use of accelerometers to teach numerical integration and differentiation to mechanical engineering students, providing a framework and process for effective implementation in a classroom setting. While the fundamental concepts of numerical integration and differentiation are typically introduced through theoretical datasets, engaging with real-world data presents complexities that extend beyond straightforward calculations. This underscores the necessity of equipping students with skills for effective data interpretation and processing to achieve accurate and meaningful results. Accelerometers, which measure acceleration, serve as a practical and interactive tool for teaching these concepts within an engineering context. Their simplicity in data collection makes them ideal for classroom experiments, while also enabling students to explore easily relatable concepts such as velocity, displacement, and jerk, derived through numerical methods. In this study, students conducted experiments by capturing acceleration data from short, linear movements and utilized instructor-provided code to process the data for displacement estimation. Through this analysis, they were introduced to key concepts such as sensor drift and noise. Students then learned to implement various data processing techniques, including filtering, smoothing, and detrending. Methods like polynomial detrending, low-pass filtering, Power Spectral Density (PSD) thresholding, and Savitzky-Golay filtering were applied to effectively address these issues. While no formal assessment data were collected, anecdotal feedback and class observations suggest enhanced student engagement and comprehension. This study leads us to conclude that accelerometers serve as an effective hands-on tool for teaching numerical methods. By engaging students in experiments and data analysis, they enhance their understanding of these techniques and acquire practical skills relevant to engineering careers.

Keywords: Accelerometers, Engineering Education, Numerical Differentiation, Noise Handling, Numerical Integration, Sensor Drift.

Introduction

Numerical differentiation and integration serve as foundational concepts in mathematics and engineering. These concepts play a vital role in analyzing and modeling continuous phenomena, enabling accurate predictions and solutions to real-world problems. For example, numerical integration is used to estimate velocity and displacement from accelerometer data, which is crucial in fields like automotive crash testing. In crash testing, accelerometers capture acceleration during an impact. By integrating this data, engineers can calculate velocity and displacement to assess vehicle deformation and passenger movement, informing the design of airbags, seatbelts, and crumple zones. Additionally, numerical differentiation allows for the calculation of higher-order derivatives, including jerk, snap, crackle, pop, and lock (first to fifth derivatives of acceleration) [1]. Among these, jerk—the rate of change of acceleration—is particularly important in crash safety. High jerk values indicate sudden changes in force, correlating with a higher risk of injury. Reducing jerk is critical for improving occupant safety. While the theoretical implementation of these techniques is typically straightforward, real-world

applications involving discrete sensor data introduce errors due to various factors. Addressing and understanding these errors are pivotal for attaining precise results in practical applications.

Incorporating accelerometers while teaching numerical differentiation and integration presents a unique opportunity to gain hands-on experience with real-world data and learn to apply numerical techniques to address common challenges in sensor data analysis. This approach enhances student engagement and prepares them for the challenges of applying numerical methods in practical settings.

Literature Review

Traditional methods for teaching numerical concepts often fall short when it comes to demonstrating their practical relevance, which can result in lower student engagement. Feedback from students in numerical methods courses, as noted by Musto [2], indicates that these courses are sometimes seen as disconnected from real-world applications and lacking in hands-on opportunities. This disconnect highlights the need for updated teaching approaches that better link abstract mathematical principles with practical use cases.

Incorporating accelerometers into numerical methods instruction provides an effective way to address these challenges. Prochazka et al. [3] advocate for integrating sensor data into digital signal processing (DSP) courses, highlighting the strong connection between numerical analysis, DSP, and sensor technologies. They argue that real-world data from devices like accelerometers not only reinforces theoretical knowledge but also equips students with essential data analysis skills for modern fields such as robotics and motion tracking. Expanding on this idea, Urban-Woldron [4] emphasizes the benefits of motion sensors in mathematics education, demonstrating how real-time sensor data makes abstract concepts more interactive and accessible. Chan et al. [5] further stress the importance of incorporating sensor technology alongside artificial intelligence in education (AIED), educational data mining (EDM), and learning analytics (LA) to better prepare students for modern industry demands while fostering critical thinking and problem-solving skills. Barbosa et al. [6] propose a framework utilizing Python, Raspberry Pi, and MEMS sensors to create affordable, user-friendly systems for teaching physics and engineering. Their approach enables hands-on projects that seamlessly integrate theoretical learning with practical experimentation. Collectively, these studies underscore the value of sensor technology in making education more interactive, accessible, and aligned with real-world applications.

Ramírez et al. [7] demonstrate how accelerometer data can be used to calculate an object's speed and position through numerical integration techniques such as the Riemann sum and the trapezoidal rule. These applications transform abstract mathematical concepts into interactive, real-world experiences, significantly enhancing student engagement and understanding. Similarly, Herceg et al. [8] showcase practical applications of numerical differentiation and integration using sensor data, reinforcing students' ability to connect theoretical knowledge with real-world devices and technologies. Varanis et al. [9] demonstrate the effectiveness of low-cost MEMS accelerometers for teaching mechanical vibrations, providing students with hands-on experience in data acquisition and analysis. Beyond numerical methods accelerometers have been successfully integrated into broader engineering education initiatives, such as Arduino-based vibration analysis for civil engineering applications [10] and engineering system design

[11]. These projects illustrate how modern technology bridges the gap between theory and practice, preparing students for real-world engineering challenges.

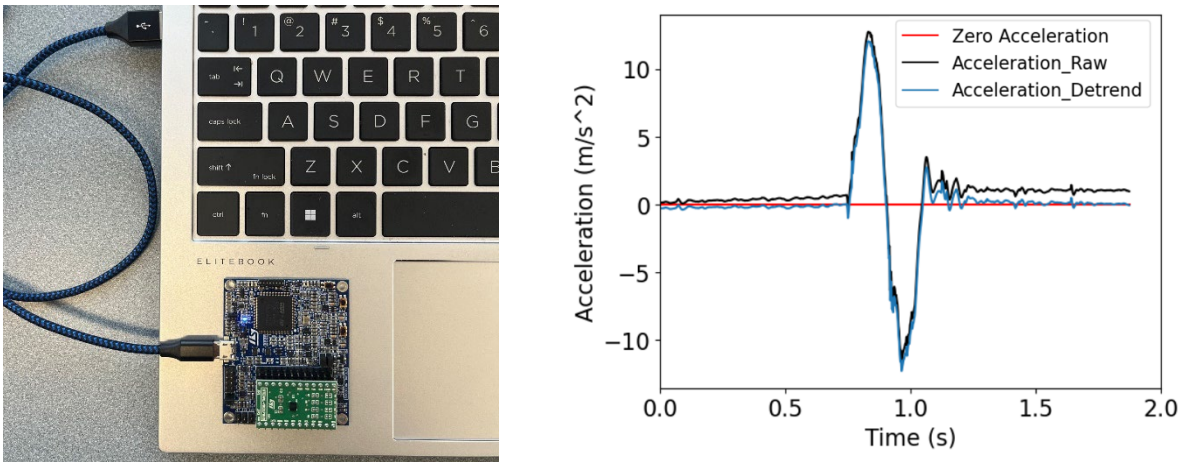
By enabling real-time data collection and analysis, accelerometers enhance experiential learning, allowing students to engage with real-world data and apply theoretical concepts in a tangible manner. This approach is grounded in John Dewey's experiential learning principles [12] and Kolb's Experiential Learning Theory [13], both of which emphasize learning through experience. Studies show that experiential learning improves comprehension, problem-solving skills, and knowledge transfer [14] [15], making accelerometers a valuable tool for reinforcing numerical integration and differentiation concepts in engineering education. Overall, integrating accelerometer-based learning activities into numerical methods courses directly addresses the need for more practical applications in coursework. By incorporating sensor-based tasks, students not only strengthen their understanding of core mathematical concepts but also develop hands-on skills that are increasingly relevant in engineering and technology fields. The studies discussed consistently highlight that hands-on learning, facilitated by real-time sensor data, is a powerful strategy for improving both comprehension and engagement in numerical methods education.

Equipment

The acceleration data was collected during a classroom demonstration using a Steval-MKI178V2 (LSM6DSL) inertial measurement unit (IMU) sensor board by STMicroelectronics. To enable data collection the IMU chip is interfaced with the Steval-MKI109V MEMS DIL24 adapter motherboard and then connected to the computer using the USB interface as shown in Figure 1 (a). The Unico-GUI MEMS evaluation kit software package is used to collect the acceleration data. The Unico-GUI allows for data to be logged and saved to a CSV file. The data is then processed using Python. Although the LSM6DSL produces 3-axis acceleration and 3-axis angular velocity data, only the acceleration data along the X-axis is used for the demonstration. Many other options exist on the market in terms of accelerometers, DAQ's, and processing software. One option for educational environments could be to consider the Arduino Nano 33 BLE with an embedded 9-axis inertial sensor, which can be purchased for under $30 and provides an economic option for large classrooms.

Data Collection

The acceleration data is collected in three steps. In step 1, data is gathered while the accelerometer is held still. In step 2, the accelerometer is quickly moved 0.15 m along its x-axis. In step 3, the accelerometer is held still again for a short duration before stopping data collection. The sample data collected using this process is labeled `Acceleration_Raw` in Figure 1 (b), where steps 1, 2, and 3 correspond to the time intervals 0–0.75 seconds, 0.75–1.1 seconds, and 1.1–2.0 seconds, respectively.



(a) (b)

Figure 1: (a) Hardware setup, (b) Acceleration vs time plot.

Engaging students in real-time data collection using accelerometers helps them understand the practical challenges of working with raw sensor data. This activity reinforces the importance of data acquisition techniques, sampling rates, and real-world constraints, laying the foundation for all subsequent numerical methods applications. By actively collecting their own data, students develop a stronger appreciation for measurement accuracy, variability, and real-time system behavior, which enhances their ability to analyze sensor-based systems in diverse engineering fields.

Drift Identification

When comparing the raw acceleration data (`Acceleration_Raw`) with the expected acceleration in Figure 1 (b), it is evident that the sensor outputs non-zero values even when the accelerometer is stationary during Steps 1 and 3. This gradual deviation of the sensor's output from the true value over time is known as drift [16]. This phenomenon can significantly affect the accuracy and reliability of measurements, especially in applications requiring long-term data collection [16] or high precision. Drift can lead to incorrect interpretations of data, resulting in erroneous calculations of velocity, displacement [16], and higher-order derivatives like jerk and snap. The main causes of drift in accelerometer data include sensor bias [16] which is an inherent offset in the sensor output, and internal and external variations in temperature [16][17] known as thermal drift.

To address drift, various techniques can be employed, including regular calibration using external measurements of position, velocity, and attitude to account for sensor bias [16]. Another effective method is sensor fusion, which combines data from multiple sensors to improve accuracy [18]. Additionally, modeling and compensating for sensor bias and thermal drift can help reduce the impact of drift [19]. While no technique can eliminate accelerometer drift, these methods can significantly mitigate its effects.

By analyzing collected accelerometer data, students learn to recognize drift as a systematic error that affects long-term accuracy. Understanding drift is crucial for applications ranging from

robotics to biomedical engineering, where precise motion tracking is essential. This exercise strengthens students' ability to diagnose and correct systematic errors in real-world datasets, fostering critical thinking and adaptability when working with noisy signals.

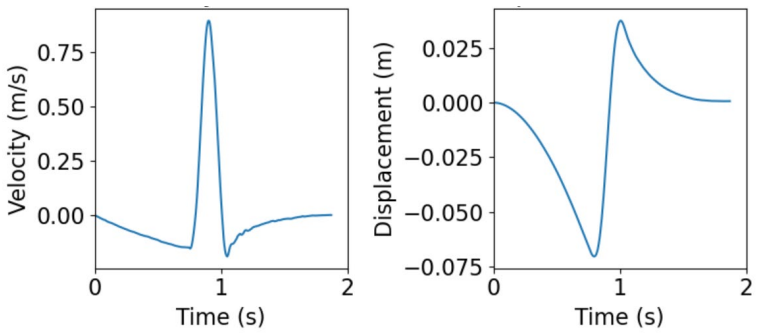
The methods described below provide a straightforward drift compensation technique to correct linear trends in accelerometer signals. Additionally, we will explore basic data filtering and smoothing methods to correct for noise in sensor data. These techniques can be covered at a high level in a single lecture, ensuring that the primary course objectives aligned with ABET standards are not disrupted.

Drift Compensation and Numerical Integration

Linear detrending is a simple method to remove unreasonable trends in time series data [20]. Implementable using the `detrend` function in Python, it mitigates drift in accelerometer data by removing common linear trends. However, it may not effectively address other types of drift. The detrended signal is labeled `Acceleration_Detrend` in Figure 1 (b).

Although the `detrend` function effectively mitigates linear drift, significant residual drift persists, rendering it unsuitable for accurately determining the velocity and displacement of the accelerometer. This limitation becomes evident when integrating the detrended acceleration signal to calculate velocity and displacement. The integration is performed using the trapezoidal rule, facilitated by the `cumtrapz` function from the SciPy package in Python, and the results are depicted in Figure 2. The computed data incorrectly indicates non-zero velocities during Steps 1 and 3, with a net final displacement close to zero. This process highlights the cumulative nature of integration errors and the need for effective correction strategies. Through this activity, students gain a deeper understanding of the relationship between acceleration, velocity, and position, improving their ability to apply numerical methods to real-world motion analysis problems.

When relying solely on the detrend method to correct drift, accelerometers are primarily suitable for measuring short and rapid displacement events [16]. This limitation arises because the detrend method may not sufficiently correct for more complex forms of drift and long-term inaccuracies, which can accumulate over extended periods or during slower movements.



(a) (b)

Figure 2: (a) velocity vs time plot, (b) displacement vs time plot.

Data Segmentation

The short and rapid displacement event in this case occurs during step 2. This step is approximated to occur during the time interval when the largest amplitude sinusoid crosses zero at both the beginning and the end of its cycle. The isolated acceleration data is shown in Figure 3 (a). Figure 3 (b) and (c) shows the corresponding velocity and displacement obtained through integration. The resulting velocity and displacement profiles closely resemble expected data profiles with a final displacement of 0.143 m.

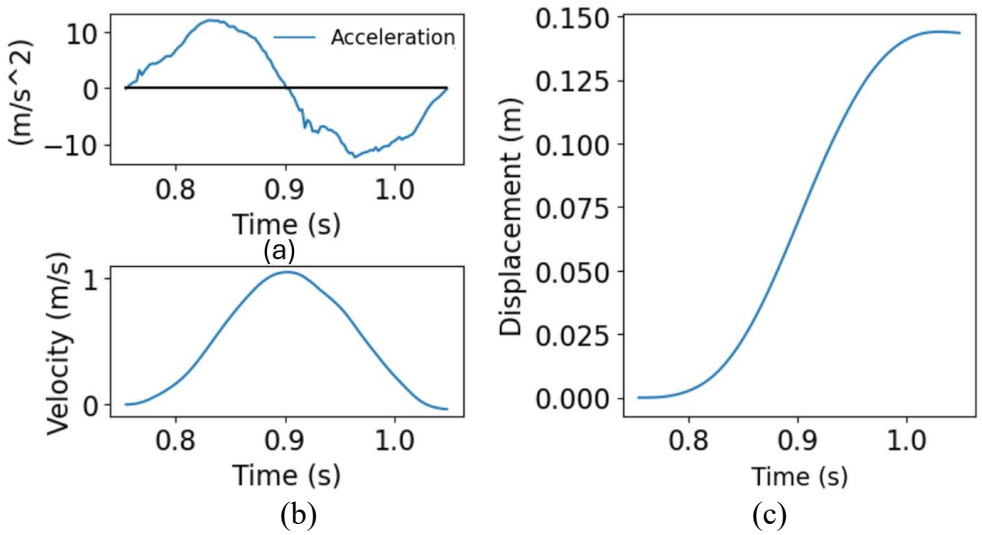


Figure 3: Numerical integration on segmented data, (a) acceleration vs time plot, (b) velocity vs time plot, (c) displacement vs time plot.

Breaking down large datasets into meaningful segments allows students to analyze trends, isolate anomalies, and improve data processing accuracy. This activity enhances their ability to work with real-time data streams, a skill essential for applications in predictive maintenance, structural health monitoring, and machine learning. By learning how to partition data effectively, students develop problem-solving strategies that translate to large-scale engineering and data science challenges.

Noise Identification

In addition to drift, the signal contains noise, which in the time domain appears as rapid fluctuations or ripples in the acceleration data. Eliminating this noise can sometimes improve results. However, in some cases, when data is integrated, the effect of removing noise can be minimal due to the noise's oscillatory nature, which tends to average out over time. This averaging reduces the noise's impact on the final integrated value, as the fluctuations above and below the mean cancel each other out.

Besides visualizing noise in the time domain, it is beneficial to analyze noise in the frequency domain as well, facilitating the identification of frequency-specific noise components. This is achieved by computing the Discrete Fourier Transform (DFT) of the signal using an algorithm like the Fast Fourier Transform (FFT) to convert the time-domain signal into the frequency domain. In

Python, this can be accomplished with the `fft.fft` function from the NumPy package, which computes the signal DFT coefficients at each frequency. Calculating the magnitude of the DFT coefficients and plotting against the frequency axis yields the amplitude spectrum. The amplitude spectrum provides a measure of the magnitude of each frequency component. The amplitude spectrum for the segmented acceleration signal is shown in Figure 4.

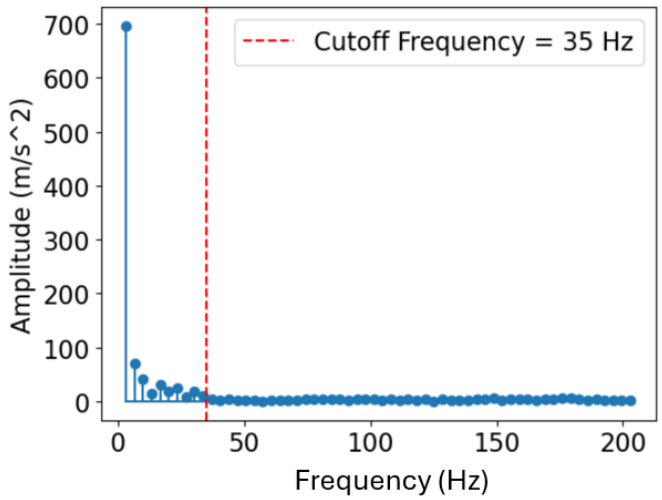


Figure 4: Amplitude spectrum plot.

High Frequency and White Noise

Two common types of noise present in the signal can be white noise or high frequency noise. White noise, in the time domain, appears as random, uncorrelated fluctuations. In the amplitude spectrum, white noise will show as flat or uniform distribution across frequencies [21][22]. High-frequency noise appears in the time domain as rapid, short-duration fluctuations superimposed on the signal. In the amplitude spectrum, this noise is characterized by significant magnitude in the higher frequency ranges, typically tapering off at lower frequencies and often extending beyond the range of useful signal frequencies. This type of noise can interfere with the signal's valuable high-frequency components, potentially masking important information. In accelerometer data, high-frequency noise may result from mechanical vibrations, sensor resonance, or electronic interference. White noise is primarily caused by thermal fluctuations in the sensor's internal components and inherent electronic noise from analog-to-digital conversion. The amplitude spectrum in Figure 4 shows signal frequencies extending up to 203 Hz, with white noise predominantly present above approximately 35 Hz.

The ability to identify and classify noise types enhances students' ability to troubleshoot real-world measurement issues. By analyzing frequency components, they gain an intuitive understanding of how noise masks useful data and affects signal processing outcomes. This knowledge is essential for designing robust engineering systems that function reliably in noisy environments, such as industrial automation and mobile robotics

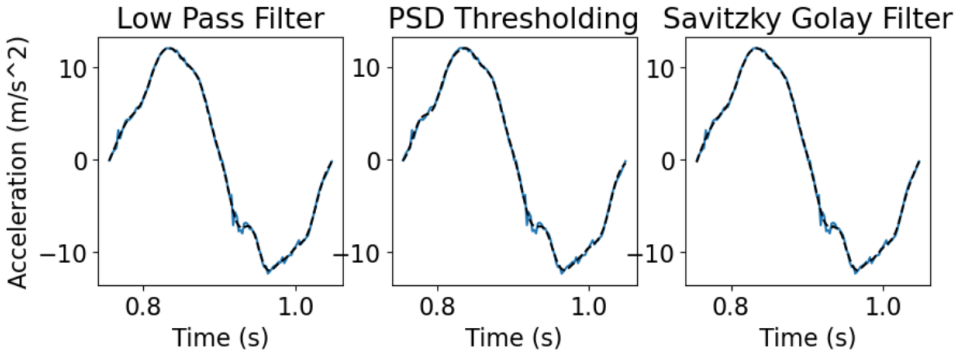
Noise Filtering

Handling noise in data is crucial for enhancing signal quality and extracting meaningful information. Three common noise removal techniques are low-pass filtering, Power Spectral Density (PSD) thresholding, and using the Savitzky-Golay filter. For simplicity, our analysis of these filtering techniques will be limited to visual inspection.

Noise filtering techniques enable students to enhance signal quality by reducing unwanted distortions while preserving critical data features. Through hands-on experimentation, they gain insights into the trade-offs between signal smoothing and data distortion, reinforcing their understanding of numerical methods and error propagation. Applying low-pass filters introduces frequency-domain processing concepts, helping students grasp convolution and transfer functions, which are fundamental in sensor fusion, communication systems and real-time data analysis. Power Spectral Density (PSD) thresholding connects frequency-domain analysis with practical filter design, equipping students with skills applicable to structural monitoring and speech recognition, and diagnostics. The Savitzky-Golay filter further refines their ability to balance noise reduction with signal preservation, strengthening their knowledge of polynomial approximation and convolution techniques essential in fields of spectroscopy, image processing, and high-precision sensor data analysis, where accurate signal retention is critical. By exploring these methods, students develop problem-solving skills that are transferable across various engineering and scientific disciplines.

Low Pass Filter

A low-pass filter attenuates frequencies above a specified cutoff frequency while allowing lower frequencies to pass through. This makes it effective for dealing with both white and high-frequency noise if the useful signal frequencies are primarily in the low-frequency region. In this case, a Butterworth low-pass filter will be implemented to smooth out fluctuations caused by white noise above 35 Hz. The filter can be applied in Python using the `signal.butter` function. Figure 5 (a) shows the filtered acceleration signal after implementing the low-pass filter.



(a) (b) (c)

Figure 5: Filtered acceleration using: (a) low-pass filter, (b) PSD thresholding, (c) Savitzky-Golay filter. The unfiltered data is plotted as a solid blue line while the filtered data is plotted as a dashed black line.

Filtering using Power Spectral Density (PSD) Thresholding

The Power Spectral Density (PSD) examines how power is distributed across frequencies, offering insights into how a signal's energy is distributed across its frequency spectrum. Figure 6 displays the PSD of the segmented acceleration signal. Notably, at 0 Hz, the power measures 3941.3 $(m/sec^2)^2/Hz$. PSD thresholding involves setting a threshold, $P_{threshold}$, to selectively remove or preserve frequency components based on their power. This technique proves valuable in scenarios where signal noise or useful components span multiple frequencies, enabling targeted attenuation of noise frequencies according to a specified $P_{threshold}$. Figure 5 (b) illustrates the filtered acceleration signal following the implementation of a $P_{threshold}$ of 1 $(m/s^2)^2/$ Hz.

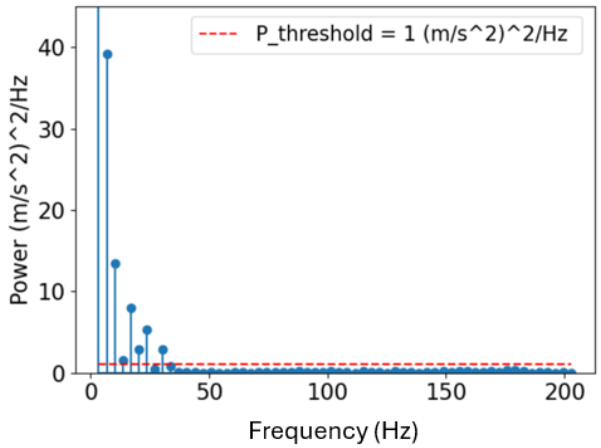


Figure 6: Power Spectral Density (PSD) plot.

Savitzky-Golay Filter

The Savitzky-Golay (SG) filter smooths data by fitting successive subsets of adjacent points, with a low-degree polynomial using linear least squares [23], effectively reducing noise while preserving signal features. It is typically used to smooth a noisy signal whose frequency range without noise is large [24]. Key to its effectiveness is selecting the appropriate window length and polynomial order. Usually, the window length must be an odd number [25][26] and should be large enough to smooth noise but small enough to retain important data features [26][27], often starting with 5-11 points. The polynomial order should match the signal's complexity; lower orders (2-3) work for general smoothing, while higher orders (4 or above) capture more complex structures but risk fitting noise. To select these parameters, start with small values, visually inspect the results, and incrementally adjust until the desired balance of noise reduction and feature preservation is achieved. Note that the SG filter performance near the edges of the data can degrade [25] due to incomplete data points for polynomial fitting. To minimize edge effects, consider padding the data or using alternative methods near data boundaries. Figure 5 (c) shows the filtered acceleration signal using a window length of 15 points fitted using a 2^{nd} order polynomial.

Since the raw data has little high-frequency noise and is mainly composed of low-frequency components, all three filtering techniques produce nearly identical results.

Calculating Velocity and Displacement using Filtered Data

Figure 7 and Figure 8 show the velocity and displacement derived from the filtered acceleration signals. Since the raw data has little high-frequency noise, the filtered acceleration signals and thus the velocity and displacement plots, obtained from all three filtering techniques produce nearly identical results.

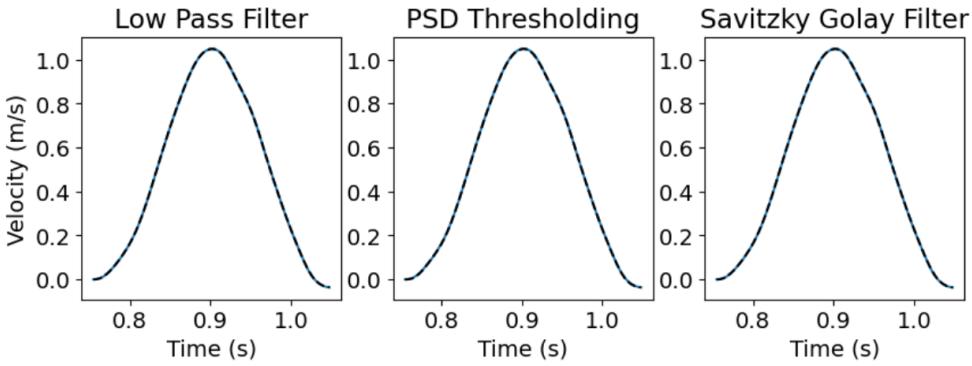


Figure 7: Velocity from acceleration signal that was filtered using, (a) low-pass filter, (b) PSD thresholding, (c) Savitzky-Golay filter.

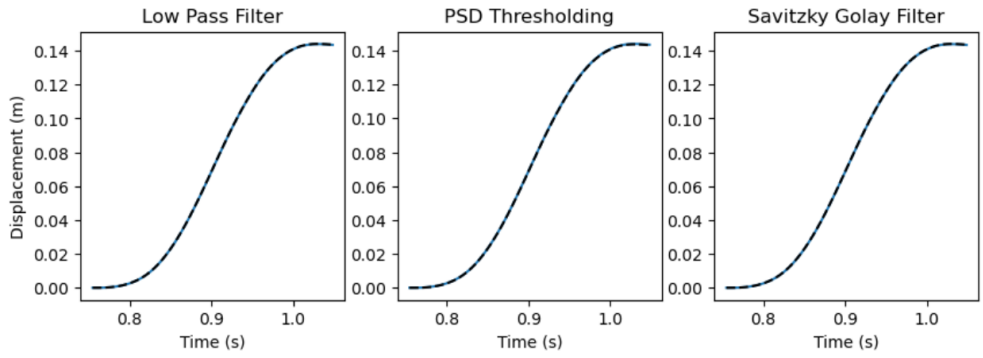


Figure 8: Displacement from acceleration signal that was filtered using, (a) low-pass filter, (b) PSD thresholding, (c) Savitzky-Golay filter.

The results indicate that noise filtering had little effect on their values. This effect is particularly noticeable with white noise, where the mean tends towards zero over time. In contrast, correlated noise may not fully average out, potentially amplifying specific components during integration. Moreover, a high signal-to-noise ratio (SNR), where the signal amplitude greatly exceeds the noise, enhances the effectiveness of noise averaging through integration. Conversely, if the noise amplitude is comparable to the signal, it can significantly influence the integrated result.

Numerical Differentiation and Jerk Calculation

To compute jerk, acceleration is differentiated using central difference, facilitated by the `gradient` function from the NumPy package in Python. Figure 9 illustrates jerk calculated from

unfiltered and filtered acceleration signals. The jerk values computed from the filtered acceleration signals exhibit noise, which arises because derivatives amplify residual noise inherent in the filtered acceleration signals. This amplification arises because noise typically includes high-frequency components, which become more pronounced when differentiated.

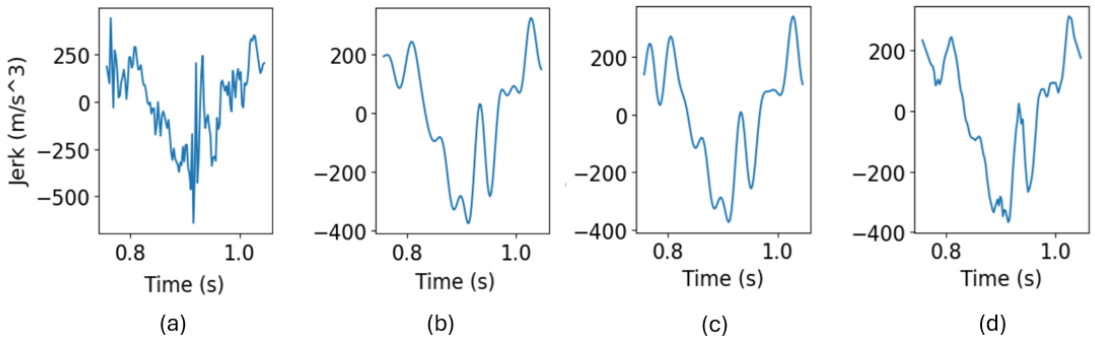


Figure 9: Jerk from acceleration signal that was filtered using, (a) no filter, (b) low-pass filter, (c) PSD thresholding, (d) Savitzky-Golay filter.

Adjusting Filter Parameters

Fine-tuning filter coefficients through visual inspection, can help further reduce the residual noise in the filtered acceleration signals. For example, setting the low-pass filter cut-off to 15 Hz, the PSD threshold to 10 Hz, and configuring the Savitzky-Golay filter with a window size of 33 and a polynomial order of 3, can effectively minimize noise. Figure 10 illustrates that the jerk calculated using these adjusted filter coefficients applied to the acceleration signal exhibits reduced noise levels.

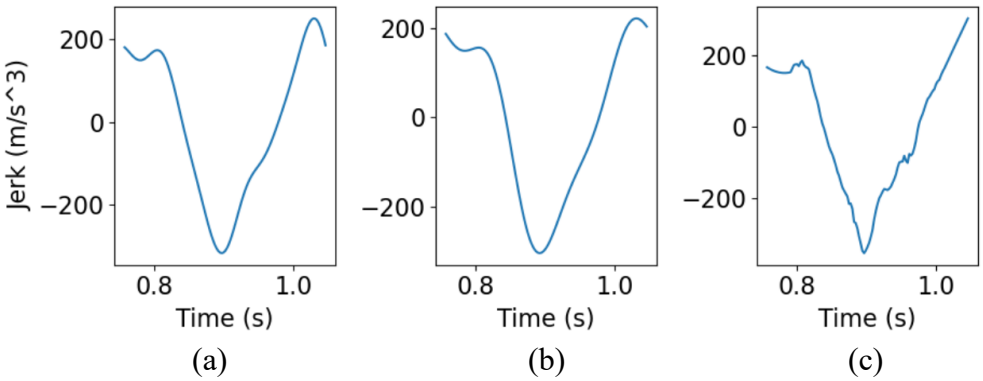
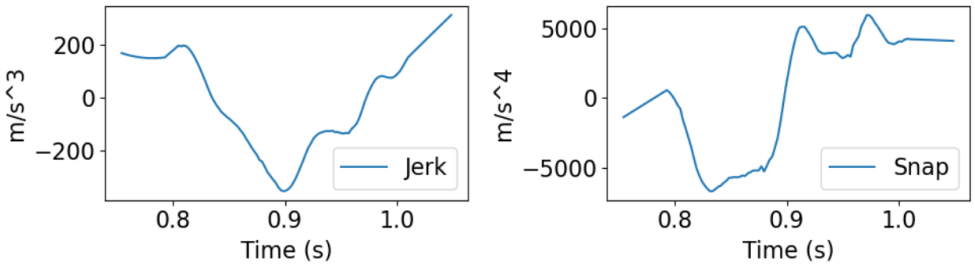


Figure 10: Jerk from acceleration signal that was filtered using modified filter parameters, (a) low-pass filter, (b) PSD thresholding, (c) Savitzky-Golay filter.

By tuning filter parameters and observing their impact on data quality, students develop intuition for signal processing trade-offs. This activity deepens their understanding of numerical stability, error minimization, and real-time system constraints. The ability to fine-tune algorithms is essential for applications in adaptive control, real-time monitoring, and embedded systems.

Calculating Higher Order Derivatives using Savitzky-Golay Filter

To compute the second derivative of acceleration, one can filter the jerk signal again and then apply central differencing. This approach can be extended to compute even higher-order derivatives. Alternatively, utilizing the derivative feature of the Savitzky-Golay filter [25] offers simultaneous data smoothing and derivative calculation and eliminates the need to repeatedly filter the signal before calculating subsequent derivatives. The derivative order can be specified as 0 for smoothing, 1 for the first derivative, 2 for the second derivative, and so forth. Figure 11 illustrates the first and second derivatives of the acceleration signal representing jerk, and snap, respectively, computed using the Savitzky-Golay filter's derivative capability using default filtering settings.



(a) (b)

Figure 11: Higher order derivatives of acceleration calculated using the Savitzky-Golay filter, (a) jerk vs time plot, (b) snap vs time plot.

Final Word on Filter Selection and Implementation Criteria

The choice of technique for filtering noise should be based on the noise's frequency characteristics. Low-pass filters are effective for high-frequency noise, while PSD filtering and Savitzky-Golay filters can handle noise with more complex frequency distributions. It's important to consider how each technique affects signal integrity, as low-pass filters can alter high-frequency components, whereas Savitzky-Golay filters aim to preserve signal features. Additionally, evaluate the computational requirements of each technique, especially for real-time applications.

In-Class Activity Overview, Challenges, and Recommendations for Educators

The accelerometer-based numerical differentiation and integration activity was conducted during a 120-minute session in a junior-level Numerical Methods course, divided into three parts: (1) an introduction to accelerometers, (2) hands-on data collection, and (3) numerical analysis using Python. Students had prior knowledge of numerical differentiation, integration methods, and Python programming but no experience with accelerometers or signal processing.

The session began with a brief lecture on accelerometer theory, followed by students working in pairs to set up their sensors and use the STMicroelectronics UNICO application for data collection. They were asked to predict the pattern of acceleration signals during quick movements along the x-axis, fostering an intuitive understanding of motion and sensor data. Although some struggled to visualize the expected patterns, this task highlighted the complexities of motion and sensor calibration.

Over the next 60 minutes, students collected motion data and modified a provided starter code template for data processing along with the instructor. Several students faced challenges capturing sinusoidal motion and encountered data clipping issues due to incorrect sensor calibration settings, prompting discussions on proper calibration techniques. During the code implementation phase, an overview of various noise and drift correction techniques was provided. To introduce the concept of the Fast Fourier Transform (FFT) and Power Spectral Density (PSD), simpler sinusoidal signals were used as examples. Each noise correction technique was first applied to these simplified signals before moving on to more complex real-world accelerometer data. During the final 45 minutes, students independently re-collected signals and applied noise filtering techniques. They answered key questions on frequency content, noise types, and appropriate filtering methods, demonstrating their understanding of signal processing concepts.

While most students successfully completed the activity, the session felt rushed. Some groups struggled with accurate data collection and continued working outside class. Extending the session by an hour would allow students to refine their work, absorb concepts more effectively, and receive more individualized support, enhancing engagement and learning outcomes.

Conclusion

This study showcases the innovative integration of accelerometers in teaching numerical differentiation and integration concepts, emphasizing practical applications and providing a hands-on, interactive learning experience. This approach enhances student engagement and comprehension by allowing them to analyze real-world data. Students encounter critical challenges such as sensor drift, noise, and data smoothing, thereby developing essential problem-solving skills relevant to fields like robotics, aerospace engineering, and biomechanics. The investigation highlights the significant impact of sensor drift and noise on data analysis. Techniques such as detrending, low-pass filtering, Power Spectral Density (PSD) filtering, and Savitzky-Golay filtering were employed to address these issues. The effectiveness of these methods was demonstrated in accurately calculating velocity, displacement, and higher-order derivatives like jerk. The study underscores the importance of proper noise handling and drift correction for achieving precise results when using sensor data to predictive analysis. Overall, incorporating accelerometer data into numerical methods education equips students with valuable analytical skills and technical proficiency, preparing them for future careers in various engineering disciplines.

References

[1] Pendrill, A. M., & Eager, D. (2020). Velocity, acceleration, jerk, snap and vibration: Forces in our bodies during a roller coaster ride. Physics Education, 55(6), 065012.

[2] Musto, J. C. (2002). A project-based approach to teaching numerical methods. International Journal of Mechanical Engineering Education, 30(3), 233-247.

[3] Prochazka, A., Vysata, O., & Marik, V. (2021). Integrating the role of computational intelligence and digital signal processing in education: Emerging technologies and mathematical tools. IEEE Signal Processing Magazine, 38(3), 154-162.

[4] Urban-Woldron, H. (2015). Motion sensors in mathematics teaching: learning tools for understanding general math concepts?. International Journal of Mathematical Education in Science and Technology, 46(4), 584-598.

[5] Chan, Y., Lai, J., Hsieh, M., & Meen, T. (2023). Important elements of sensor technology and data management and related education. Sensors and Materials, 35(4), 1161. https://doi.org/10.18494/sam4015

[6] Barbosa, G., Varanis, M., Delgado, K., & Oliveira, C. (2020). An acquisition system framework for mechanical measurements with python, raspberry-pi and mems sensors. Revista Brasileira De Ensino De Física, 42. https://doi.org/10.1590/1806-9126-rbef-2020-0167

[7] Cruz-Ramírez, S. R., García-Martínez, M., & Olais-Govea, J. M. (2022). NAO robots as context to teach numerical methods. International Journal on Interactive Design and Manufacturing (ijidem), 16(4), 1337-1356.

[8] Herceg, Đ., & Herceg, D. (2019, September). Arduino and numerical mathematics. In Proceedings of the 9th Balkan Conference on Informatics (pp. 1-6).

[9] Varanis, M., Silva, A., Mereles, A., & Pederiva, R. (2018). Mems accelerometers for mechanical vibrations analysis: a comprehensive review with applications. Journal of the Brazilian Society of Mechanical Sciences and Engineering, 40(11). https://doi.org/10.1007/s40430-018-1445-5

[10] Mutlu, A. K., Tuğsal, Ü. M., & Dindar, A. A. (2021). Utilizing an arduino-based accelerometer in civil engineering applications in undergraduate education. Seismological Research Letters, 93(2A), 1037-1045. https://doi.org/10.1785/0220210137

[11] Liu, T. L., Wu, X. Q., & Yi, Z. (2012). Study on china's engineering education reform. Advanced Materials Research, 591-593, 2337-2340. https://doi.org/10.4028/www.scientific.net/amr.591-593.2337

[12] Miettinen, R. (2000). The concept of experiential learning and John Dewey's theory of reflective thought and action. International journal of lifelong education, 19(1), 54-72.

[13] Akella, D. (2010). Learning together: kolb's experiential theory and its application. Journal of Management &Amp; Organization, 16(1), 100-112. https://doi.org/10.5172/jmo.16.1.100

[14] Kwan, Y. L. L. (2022). Exploring experiential learning practices to improve students' understanding. PUPIL: International Journal of Teaching, Education and Learning, 6(1), 72-89. https://doi.org/10.20319/pijtel.2022.61.7289

[15] Ajani, O. A. (2023). The role of experiential learning in teachers' professional development for enhanced classroom practices. Journal of Curriculum and Teaching, 12(4), 143. https://doi.org/10.5430/jct.v12n4p143
[16] Pang, G., & Liu, H. (2001). Evaluation of a low-cost MEMS accelerometer for distance measurement. Journal of Intelligent and Robotic Systems, 30, 249-265.
[17] Dai, G., Li, M., He, X., Du, L., Shao, B., & Su, W. (2011). Thermal drift analysis using a multiphysics model of bulk silicon MEMS capacitive accelerometer. Sensors and Actuators A: Physical, 172(2), 369-378. Https://doi.org/10.1016/j.sna.2011.09.016
[18] Nemec, D., Janota, A., Hruboš, M., & Šimák, V. (2016). Intelligent real-time MEMS sensor fusion and calibration. IEEE Sensors Journal, 16(19), 7150-7160. Https://doi.org/10.1109/JSEN.2016.2597292
[19] Ruzza, G., Guerriero, L., Revellino, P., & Guadagno, F. M. (2018). Thermal compensation of low-cost MEMS accelerometers for tilt measurements. Sensors, 18(8), 2536. Https://doi.org/10.3390/s18082536.
[20] Pan, C., Zhang, R., Luo, H., & Shen, H. (2016). Baseline correction of vibration acceleration signals with inconsistent initial velocity and displacement. Advances in Mechanical Engineering, 8(10), 1687814016675534. Https://doi.org/10.1177/1687814016675534
[21] Chen, X., Wang, M., Zhang, Y., Feng, Y., Wu, Z., & Huang, N. E. (2013). Detecting signals from data with noise: theory and applications. Journal of the Atmospheric Sciences, 70(5), 1489-1504. Https://doi.org/10.1175/JAS-D-12-0213.1
[22] Jerath, K., Brennan, S., & Lagoa, C. (2018). Bridging the gap between sensor noise modeling and sensor characterization. Measurement, 116, 350-366.
[23] Schafer, R. W. (2011). What is a Savitzky-Golay filter? [lecture notes]. IEEE Signal processing magazine, 28(4),. Https://doi.org/10.1109/MSP.2011.941097
[24] De Oliveira, M. A., Araujo, N. V., Da Silva, R. N., Da Silva, T. I., & Epaarachchi, J. (2018). Use of savitzky–golay filter for performances improvement of SHM systems based on neural networks and distributed PZT sensors. Sensors, 18(1), 152. Https://doi.org/10.3390/s18010152
[25] Schmid, M., Rath, D., & Diebold, U. (2022). Why and how Savitzky–Golay filters should be replaced. ACS Measurement Science Au, 2(2), 185-196. Https://doi.org/10.1021/acsmeasuresciau.1c00054
[26] Angrisani, L., Capriglione, D., Cerro, G., Ferrigno, L., & Miele, G. (2014). The effect of Savitzky-Golay smoothing filter on the performance of a vehicular dynamic spectrum access method. Proceedings of the 20th IMEKO TC4, IWADC, 1116-1121.
[27] Sadeghi, M., Behnia, F., & Amiri, R. (2020). Window selection of the Savitzky–Golay filters for signal recovery from noisy measurements. IEEE Transactions on Instrumentation and Measurement, 69(8), 5418-5427. Https://doi.org/10.1109/TIM.2020.2966310

Appendix

Python Code

```
#Importing relevant libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.integrate import cumtrapz
from scipy.signal import savgol_filter
from statsmodels.tsa.tsatools import detrend


#Data Collection and Drift Identification

df = pd.read_csv('Position.txt',sep='\t', skiprows = 1)  #Importing accelerometer data into a pandas dataframe
accel_raw = df['A_X [mg] '].values[340:1120]/1000*9.81   #Extracting x-axis data and converting g's to m/s^2
f = 416                                                  #Specify data capture frequency (Sample rate in Hz)
t = np.arange(0,len(accel_raw),1) / f                    #Create time array

#Drift Compensation and Numerical Integration

accel_det = detrend(accel_raw)                           #Detrend signal to correct for some of the drift
vel_det = cumtrapz(accel_det, t, initial=0)              #Integrating to calculate velocity
disp_det = cumtrapz(vel_det, t, initial=0)               #Integrating to calculate displacement

#Data Segmentation and Numerical Integration

accel_seg = accel_det[314:437]                           #Segmeting acceleration axis
t_seg = t[314:437]                                       #Segmeting time axis
vel_seg = cumtrapz(accel_seg, t_seg, initial=0)          #Integrating to calculate velocity
disp_seg = cumtrapz(vel_seg, t_seg, initial=0)           #Integrating to calculate displacement

#Noise Identification (Amplitude Spectrum)

n = len(t_seg)                                           #Determining signal length
dt = t_seg[1] - t_seg[0]                                 #Determining time step
freq =(1/(dt*n)) * np.arange(n)                          #Creating x-axis of frequencies
L = np.arange(1,np.floor(n/2),dtype = 'int')             #Halving the frequency axis
fhat = np.fft.fft(accel_seg,n)                           #Computes DFT coefficients

#Noise Filtering

#Low Pass Filter (Butterworth)

f_c = 15                                                 #Specifying cutoff frequency
wn_c = f_c / (f / 2)                                     #Normalizing the cut-off frequencies
b, a = signal.butter(5, wn_c, 'lp')                      #Generating the  filter coefficients (5th order filter)
```

```
accel_lp = signal.filtfilt(b, a, accel_seg)             #Filtering the signal

#PSD Thresholding

PSD = fhat*np.conj(fhat)/n                              #Computing PSD
P_threshold = 10                                        #Setting P_threshold
indices = PSD > P_threshold                             #Identifying DFT coefficients based on P_threshold
fhat_PSD = indices * fhat                               #Zeroing DFT coefficients below P_threshold
accel_PSD = np.fft.ifft(fhat_PSD)                       #Computing inverse Fourier transform to get filtered time signal

#Savitzky-Golay Filter

accel_savgol = savgol_filter(accel_seg,33,3)            #Filtering using the Savitzky-Golay filter

#Numerical Integration to Calculate Velocity and Displacement using Filtered Data

vel = cumtrapz(accel_seg, t_seg, initial=0)             #Integrating unifiltered signal to find velocity
disp = cumtrapz(vel, t_seg, initial=0)                  #Integrating unifiltered signal to find displacement
vel_savgol = cumtrapz(accel_savgol, t_seg, initial=0)   #Integrating filtered signal to find velocity (Savitzky-Golay)
disp_savgol = cumtrapz(vel_savgol, t_seg, initial=0)    #Integrating filtered signal to find displacement (Savitzky Golay)
vel_PSD = cumtrapz(accel_PSD, t_seg, initial=0)         #Integrating filtered signal to find velocity (PSD thresholding)
disp_PSD = cumtrapz(vel_PSD, t_seg, initial=0)          #Integrating filtered signal to find displacement (PSD thresholding)
vel_lp = cumtrapz(accel_lp, t_seg, initial=0)           #Integrating filtered signal to find velocity (Butterworth Low Pass)
disp_lp = cumtrapz(vel_lp, t_seg, initial=0)            #Integrating filtered signal to find displacement (Butterworth Low Pass)

#Numerical Differentiation and Jerk Calculation

jerk_raw = np.gradient(accel_seg, t_seg)                #Calculating jerk using unfiltered signal
jerk_savgol = np.gradient(accel_savgol, t_seg)          #Calculating jerk using filtered signal (Savitzky-Golay)
jerk_lp = np.gradient(accel_lp, t_seg)                  #Calculating jerk using filtered signal (PSD thresholding)
jerk_PSD = np.gradient(accel_PSD, t_seg)                #Calculating jerk using filtered signal (Butterworth Low Pass)

#Calculating Higher Order Derivatives using Savitzky-Golay Filter

jerk = savgol_filter(accel_seg,33,3,deriv=1,delta=dt)   #Calculating jerk
snap = savgol_filter(accel_seg,33,3,deriv=2,delta=dt)   #Calculating snap
crackle = savgol_filter(accel_seg,33,3,deriv=3,delta=dt) #Calculating crackle


#Creating the Stem Plots

plt.stem(freq[L], abs(fhat[L]))                         #Amplitude spectrum plot
plt.stem(freq[L], PSD[L])                               #PSD plot
```

.