Math 5301 – Numerical Analysis– Spring 2025
w/Professor Du

Paul Carmody
Final Project – May 8, 2025

Interpretation of Polling Data from Micro-controller input sensors.

Micro-controllers are often the first line of detection and active response to the environment. These are triggered through sensors that deliver a stream of data that must be interpreted before the properly coded response can be triggered. The data stream can fluctuate on a moment by moment basis but display overall aggregated behavior that is useful for an application. That is, the input data must be interpreted in order to give reliability of input data strenth that can then be used to determine responses. The data is interpreted through the integral of the polling data over time using the Simpson's Rule.

Assumptions and parameters:

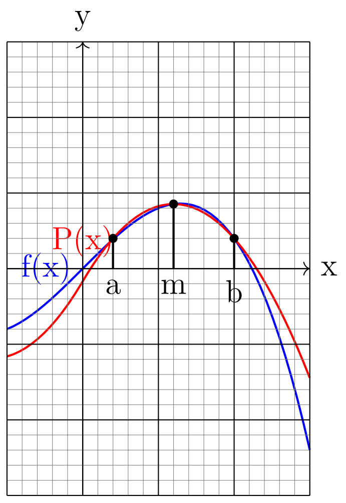
1. The data may be quite different between each measurement but changes smoothly.
2. The change in data (i.e., first derivative) also changes smoothly.
3. Make available *parameters of reliability.* These would give some indication as to the effect of accuracy and processor efficiency.
4. Attempts should be made to optimize where the greatest changes occur.
5. Reduce as small as possible, the 'software footprint', i.e., the amount of space used for processing and data.[1]

Simpson's 1/3 Rule

Because of the first two assumptions mentioned above, we can approximate each node of data with quadratic interpolation. That is, given two-endpoints we can take the midpoint and interpolate a quadratic curve, $P(x)$, with the following approximation:[2]

$$\int_a^b f(x)dx \approx P_m$$

$$P_m = \frac{b-a}{6}\left[\, f(a) + 4f(m) + f(b)\,\right]$$



where $m = \frac{a+b}{2}$. The graphs shows the quadratic interpolation $P(x)$ in red.

[1]Micro-controllers have very limited space for code and data for interpretation which must be shared with response subroutines.
[2]https://en.wikipedia.org/wiki/Simpsons_rule#Composite_Simpson's_1/3_rule

Error Analysis of Simpson's Rule.

The error for approximating via Simpon's Rule for $n = 2$ is

$$-\frac{1}{90}h^5 f^{(4)}(\xi) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi), \text{ for some } \xi \in [a,b], h = \frac{b-a}{2}$$

"Since the error term is proportional to the fourth derivative of f at ξ this shows that Simpon's Rule provides exact solutions for polynomial f of degree three or less, since the fourth derivative of such a polyomial is zero at all points."[3] Sensors used for micro-controllers usually are concerned with motion detection, infrared intensity, temperature, humidity, etc. this is a reasonable assumption that acceleration (in the case of motion as an example) would be a constant or change linearly and for a short interval (< 0.001 seconds).

Verification of Simpson's Rule

Because Simpson's rule gives accurate approximations to the third derivative, we need only verify quadratic interpolations. We can see that each of the terms of a quadratic are true with each of these terms

- The Constant Term:

$$\begin{aligned} f(x) &= 1 \\ \int_a^b dx &= (b-a) \\ P_m &= \frac{b-a}{6}\,(\,1+4+1\,) = b-a \end{aligned}$$

- The Linear Term:

$$\begin{aligned} f(x) &= x \\ \int_a^b axdx &= x^2|_a^b \\ &= \frac{b^2-a^2}{2} \\ P_m &= \frac{b-a}{6}\left(b + \frac{a+b}{2} + a\right) = \frac{b-a}{6}(3b+3a) \\ &= \frac{b^2-a^2}{2} \end{aligned}$$

[3]https://en.wikipedia.org/wiki/Simpsons_rule#Composite_Simpson's_1/3_rule

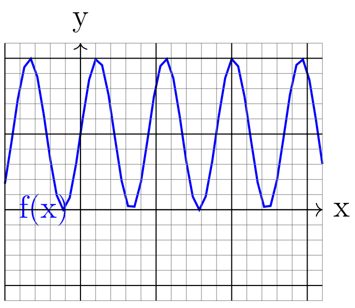
- The Quadratic Term:

$$\begin{aligned}
f(x) &= x^2 \\
\int_a^b x^2 dx &= \frac{1}{3}x^3|_a^b \\
&= \frac{b^3-a^3}{3} \\
P_m &= \frac{b-a}{6}\left(a^2 + 4\left(\frac{a+b}{2}\right)^2 + b^2\right) \\
&= \frac{b-a}{6}\left(a^2 + \left(a^2+2ab+b^2\right) + b^2\right) \\
&= \frac{b-a}{3}\left(a^2+2ab+b^2\right) \\
&= \frac{b^3-a^3}{3}
\end{aligned}$$

This evidence of Simpson's Rule on a single interval. These can be further refined by greater subdivisions and a resulting collection of quadratic interpolated elements (see the section on "Testing and Perfomance", below). This would lead to matrix of qudratic solutions that could increase complexity. However, we will see that a recursive approach will provide for greater efficiencies and more information and fulfill the requirements for the "Assumptions and Parameters" mentioned above.

A Recursive Solution.

Input data may appear more eratic than smooth as in this graph (and perhaps not as regular):



By requirement #4 from above, "Attempts should be made to optimize where the greatest changes occur", we can improve our efficiency by focusing our algorithm on sections that appear to have more activity. For this reason, instead of coding an interative approach with a finer poling data, we'll use a recursive approach dependent on the tolerance level of the overall estimated element. That is, if the estimate $m(b-a)$ is within tolerance range of $P(m)$ then we may stop. Otherwise, the interval is split into two intervals $[a, m]$ and $[m, b]$ and recursively called again. In this way, intervals with more eratic behavior will be further subdivided while those with smooth behavior will be quickly estimated.

Efficiency and Accuracy of Recursion.

Depth and Iterations. In a typical interative algorithm, cost and efficiency depend entirely on how often complicated calculations are made. This is true for a recursive solution as well, but an additional value, namely depth, offers us some measure of complexity or volatility that is not as easily available in a standard iterative solution.

Definition: Define the **depth, d** as the number of recursive calls before the tolerance level has been met.

The greater the depth the more recursions were required to make for an acceptable estimate. Clearly, if the data is smooth there will be fewer estimations needed. As the data is more eratic more recursions are needed and the depth increases. By returning the depth of the calculation we get a measure of the ”noise” or ”level of volatility” of the input data.

Definition: Define the **iterations, I** as the number of tests throughout the calculation.

Each recursive step constitutes one iteration, but might not require another recursive call (i.e., adding one to the depth). It is easy to see that a greater depth with fewer iterations indicates an isolation of volatile data (e.g., a data spike) and smaller depth in relations to the iterations indicates a smoother and more consistent data set. So, in addition to the estimate of the integral of f over $[a, b]$ we gain information on smoothness, consistency and volaltiliy. This is additional information at no cost to efficeincy that might be helpful contributing towards response to the environment.

Efficiency. Micro-controllers have very small capacity for memory (e.g., Arduino has only 32KB capacity[4]). Recursive routines can be very small leaving room for data. Further, their natural usage of stack memory recycles memory without having to manage garbage collection. Isolating the recursive call by the tolerance reduces the number of interations based entirely on the complexity of the data returning smooth intervals more quickly and executing more cycles on more volatile data where it is needed.

Lag Time. Micro-processors poll their sensors at 10K per second while the processor speed is 16 MHz, thus the lag time between measurement and evaluation depends greatly on the sample size. With the measurements of depth and iterations, the application can dynamically respond to sample size in order to gain a more timely response.

Caveat. It is possible that in a given sub-interval the midpoint could provide a value within tolerance of the expected value despite the fact that the interval might contain volatile data. Thus, passing the tolerance test, no deeper recursive calls are made. The depth, d, can serve as a confidence in calculation if other indicators do not support a lower volatility of data. Granularity of recursive calls, that is, instead of subdividing into two interval subdivide into 3 or 4 can be done on a per application basis, further reducing the potential for this mishap. Or, simply, compare the current results with the previous results for consistency.

The Algorithm.

For the purposes of this paper, the code listed here is in MatLab. The implementation code will be for that of the native language for the micro-controller (typically a version of C).

```

global pdata inc;
pdata = createArray(1,100);

function [new_est, depth, iterat] = simpson_third(est, a, b, level, itr,
    tol)

        % est: estimate of current iteration
        % a: left boundary
        % b: right boundary
        % level: how depend has recursion gone
        % itr: how many iterations have been executed
```

[4]https://docs.arduino.cc/hardware/uno-rev3/

```
        % tol:  tolerance level

        % set default responses
        global pdata inc;

        depth = level;
        iterat = itr+1;
        new_est = est;
        if a==b          % we have no interval to consider
                return;
        end

        m = floor((b-a)/2);      % midpoint

        if m <= 1        % cannot recurse any deeper
                return;
        end
        m=m+a;

        % find the new estimate

        new_est = ((b-a)*inc)/6*( pdata(a) + 4 * pdata(m) + pdata(b) );

        if abs(est - new_est) <= tol % is this estimate close enough?
                return;
        end

        new_est = new_est/2.0;
        [left, depth_left, left_itr] = simpson_third(new_est, a, m, level
           +1, itr, tol);
        [right, depth_right, right_itr] = simpson_third(new_est, m, b,
           level+1, itr, tol);
        new_est = left+right;
        depth = max(depth_left, depth_right);
        iterat = left_itr + right_itr;

end

n=1000;

% Test 1: y=x, [0, 4]
inc = 4/n;
for i = 1:n
        pdata(i) = 1;%(inc*i)^2;
end;

[t1_est, t1_dept, t1_rtr] = simpson_third(0, 1, n, 0, 0, inc);
fprintf('Test 1: estimate = %2.3f, Depth=%d, Iterations=%d\n', t1_est,
   t1_dept, t1_rtr);

% Test 2: y=x^2, [0, 4]
inc = 4/n;
for i = 1:n
```

```
        pdata(i) = (inc*i)^2;
end;

[t2_est, t2_dept, t2_rtr] = simpson_third(22, 1, n, 0, 0, inc);
fprintf('Test 2: estimate = %2.3f, Depth=%d, Iterations=%d\n', t2_est,
   t2_dept, t2_rtr);

% Test 3: y=|sin(x)|, [0, pi/2]
inc = pi/2/n;
for i = 1:n
        pdata(i) = abs(sin(i*inc));
end;

[t3_est, t3_dept, t3_rtr] = simpson_third(22, 1, n, 0, 0, inc);
fprintf('Test 3: estimate = %2.3f, Depth=%d, Iterations=%d\n', t3_est,
   t3_dept, t3_rtr);

% Test 4: y=sin(5x)+2, [0, pi/2]
inc = pi/2/n;
for i = 1:n
        pdata(i) = sin(11*i*inc)+1;
end;

[t4_est, t4_dept, t4_rtr] = simpson_third(22, 1, n, 0, 0, inc);
fprintf('Test 4: estimate = %2.3f, Depth=%d, Iterations=%d\n', t4_est,
   t4_dept, t4_rtr);

% test 5: complex partitions
inc = pi/2/n;
for i = 1:n/2;
        pdata(i) = inc*i;
end
for i=n/2:n;
        pdata(i) = sin(11*i*inc) + 1;
end

[t5_est, t5_dept, t5_rtr] = simpson_third(22, 1, n, 0, 0, inc);
fprintf('Test 5: estimate = %2.3f, Depth=%d, Iterations=%d\n', t5_est,
   t5_dept, t5_rtr);
%[est, dept, itr] = simpson_third(pi/2, 1, n, 1, 1, 1e-3)
```

Testing and Performance.

Performance testing must include the following criteria:

- Test 1: **Verify that integration is accurate.**

  Equation: $y = 1$
  Interval: $[0, 4]$.
  Expectations: Expected results: 4 and short depth and iterations

- Test 2: Simple curves that show quick response.

  Equation: $y = x^2$
  Interval: $[0, 4]$.

Expectation: Expected results: 21.333 and short depth and iterations

- Test 3: Slightly more complex curves indicate more intricate responses (i.e., greater depth).

  Equation: $y = |\sin(x)|$
  Interval: $[0, \pi/2]$.
  Expectation: result: 1, Increased depth and iterations.

- Test 4: Very volatile data. Great variations in value indicating greater depth and interations.

  Equation: $y = \sin(11x) + 1$
  Interval: $[0, \pi/2]$.
  Expectation: Result: 1.57, increased depth and iteration values.

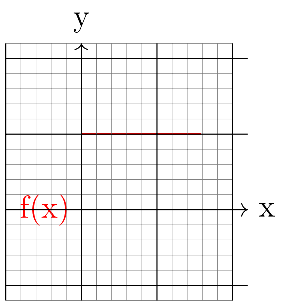
- Test 5: Partitioned data values with no relation to each other. This should indicate a mixture of depth and iterative values from previous tests.

  Equation:

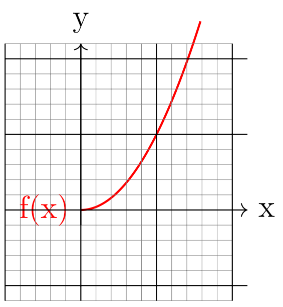
$$y = \begin{cases} x & 0 < x < \pi/4 \\ \sin(11x) + 1 & \pi/4 < i < \pi/2 \end{cases}$$

  Interval: $[0, \pi/2]$
  Expection: result: 1.142, a mixture of depth and iteration values.

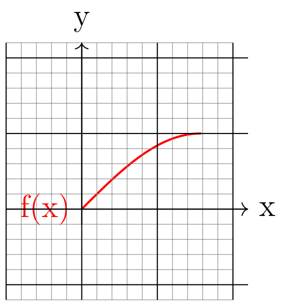
Graphs of the Test Data



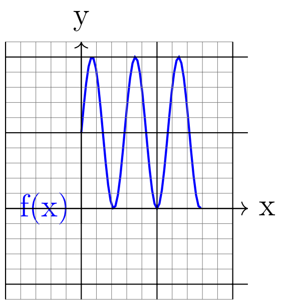
Test 1: $y = 1$



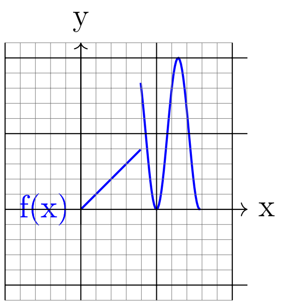
Test 2: $y = x^t$



Test 3: $y = |\sin(x)|$



Test 4: $\sin(11x) + 1$



Test 5: partitioned

The actual data can be more iratic and unpredictable than these samples. These indicate the variety of potential data input and how testing reveals the response to volatility.

	Analytic	Estimated	Depth	Iteration	Accuracy
Test 1	4	3.996	1	2	99.9%
Test 2	21.33	21.566	9	161	98.4%
Test 3	1	1	5	30	100%
Test 4	1.57	1.668	8	119	93.8%
Test 5	1.142	1.036	8	83	91%

The data supports the goal of the algorithm to spend more time on volatile areas of data and less time on smooth curves. This releases resources and processing power for the micro-controller to perform other tasks.

Recursion vs Iteration.

Iteration would require the complete computation of all n data points regardless of the quality of the data. This algorithm describes such a situation.

```
ext = 0;
for i=1:n-1
        m = (pdata(i)+pdata(i+1))/2;
        est = est + inc/6*(pdata(i) + 4*m + pdata(i+1));
end
```

These five lines of code have a smaller processor footprint than the 35 lines of code of the recursive algorithm (after removing comments and blank lines). And, the iterations for each calculation is always of order $o(n)$. We can see from the table above, though, that with 1000 data points the number of iterations of the worst case is only 16% of that typical iterative case. The recursion being binary in nature, this algorithm is of order $o(\log(n))$ having much greater response time without significant loss of accuracy while providing addtional helpful information.

Tolerance and Accuracy.

It is worth noting that the tolerance is closely associated with the value of h, the incremental value between each successive node. That is, if $h \approx 10^{-3}$ then the accuracy is in this range for each subinterval. Taking Test 4, as an example, we can start to see that the number of wave zeros in the interval, N, is limited as we cannot measure any more finely. In this case,

$$\sin(Nx) = 0 \to x = \frac{\pi}{N}, \frac{2\pi}{N}, \frac{3\pi}{N}, \ldots$$
$$h_{\min} = \frac{\pi}{N} < 0.001 \to N \approx 30$$

That is, if the sensor is fluctuating wildly over 300 times a second the algorithm isn't likely to provide an accurate measure. It is clear that volatile changes in the environment reduce the effectiveness of the algorithm.