Dynamic Trapezoidal Rule

IRVAN JAHJA / 13509099
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509099@std.stei.itb.ac.id

May 6, 2012

Abstract

The Trapezoidal Rule to approximate integral is known for its simplicity, both in implementation and in description. The method usually takes as input the bounds of the integral, as well as the number of pieces. In the most commonly found trapesium method implementation, all the pieces are of the same length. This paper discuss the possibility of having the piece length dynamic: that is, the lengths of the pieces may differ for different pieces. Furthermore, this paper presents comparison between our dynamic trapesium method results and the regular trapesium method results for several example (hopefully representative) functions.

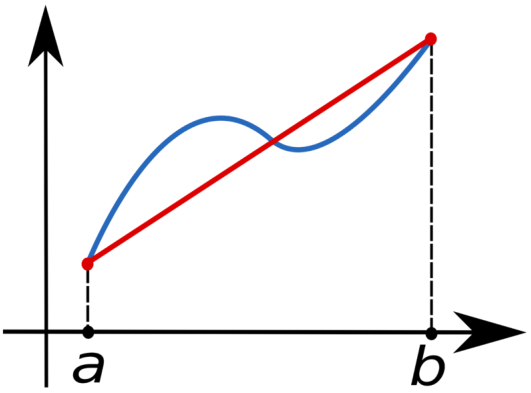

Figure 1: Illustration of the Trapezoidal Rule

1 Introduction

1.1 Background

Integral approximation is an important field of numerical computation, taking part in fields such as eletrical engineering, chemistry, and nuclear science. Due to the importance of this field, various papers and books have been published [3] [1]. In addition, various techniques have been developed: Trapezium method, Rectangle method, Simpson's method [2], to name just few.

One particular method, the trapezoidal rule, distinguish itself from the others by mean of simplicity, both in description and in implementation. Indeed, it is by this virtue that it usually get the honor to be the first algorithm taught in many courses to approximate integral. However, it suffers from lack of precision compared to other algorithms, such as Gaussian quadrature.

1.2 Roadmap

2 Trapezoidal Rule

The basic idea of trapezoidal rule is a method to approximate the definite integral:

$$\int_a^b f(x)\,\mathrm{d}x$$

such that

$$\int_a^b f(x)\,\mathrm{d}x \approx (b-a)\frac{f(a)+f(b)}{2}$$

This method is illustrated in Figure 1.

Note that this approximation is exact if the function $f(x)$ is linear.

2.1 Iterative Trapezoidal Rule

To increase the accurracy of the Trapezoidal Rule, the interval $a...b$ is cut into several regions (n). Then, the trapezoidal rule is iteratively applied to each of them, and the sum of their results is returned as the result of the execution. That is, the approximation becomes

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n}(b_i - a_i)\frac{f(a_i) + f(b_i)}{2}$$

Where a_i and b_i are the two end points of the i-th region.

The choice of the length of each of the regions can be arbitrary, but a commonly used heuristic is to have identical length for all the regions. In this paper however, we will also use choices of a_i and b_i that are not integers.

3 Idea

The idea of our improvement for the Trapezoidal Rule is based on the observation that some sections of a function can be better approximated by trapezoids than some others. For instance, for the function

$$f(x) = \frac{1}{x}$$

The approximation by trapezoid is better for the regions covering large positive values of x, as opposed to the approximation used for very small positive values of x. The trapezoidal approximation for $\int_1^2 f(x)$ is 0.75, and its actual value is around 0.6931471805599 (thus accounts for around 8.2% error). On the other hand, the trapezoidal approximation for $\int_{100}^{101} f(x)$ is 0.00995049, while its actual value is 0.00995033 (thus accounts for only around 0.00196% error).

Thus, it does not make sense to approximate both areas with the same number and length of trapezoids. To achieve better accuracy in this case, it's better to approximate regions with larger error with more trapezoids, and approximate regions with less error with less trapezoids.

3.1 Algorithm

Our algorithm works by splitting regions with large approximated error into several smaller regions, thus in effect approximating that region with more trapezoids. We will first describe how we approximate the error of a region, and then how we subdivide the regions with large errors.

3.2 Approximating Error

The immediate question is on how to measure the error of a region. To evaluate this error exactly defeats the purpose of approximating integral, so what we want is a way to approximate the error of the region. Our algorithm uses the following function to calculate the error of a region covering from a to b:

$$(b-a)(|(f(a) - f(c)) - (f(c) - f(b))|)$$

Where c is equal to $\frac{a+b}{2}$.

That is, our algorithm approximates the error linear in the following two terms:

- $(b-a)$, or, the length of the region. Indeed the expected error should be proportional with the length of the region
- $(b-a)(|(f(a) - f(\frac{a+b}{2})) - (f(\frac{a+b}{2}) - f(b))|)$, or, the difference of the difference of heights between its two end points and the midpoint. Notice that this is an approximation of $f(x)\frac{\mathrm{d}}{\mathrm{d}^2x}$. Indeed the expected error should also be proportional with derivative at approximately that region.

3.3 Trapezoids Distribution

We are now ready to present how our algorithm assigns the regions of each trapezoid. For the sake of our discussion, we will assume that n, the number of regions, is fixed.

The algorithm tries to calculate the region with the largest error. Then, it divides that region into two smaller regions with equal size. That is, it divides the region spanning from a to b into two regions, one spanning from a to $\frac{a+b}{2}$, the other one spanning from $\frac{a+b}{2}$ to b.

An execution of the iteration above is illustrated in Figure 2 and Figure 3.

The algorithm continues to do the above iteration as long as the number of pieces is less than n.

3.4 Starting Point

The algorithm above assumes that we have several starting regions. We tried and implemented two alternatives.

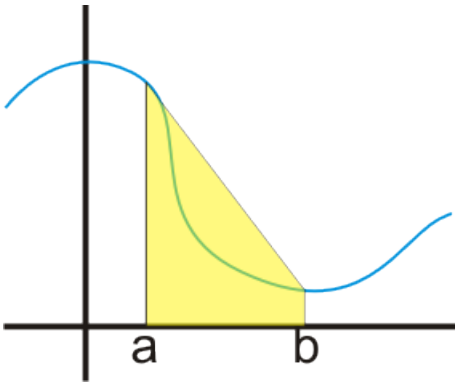

Figure 2: Trapezoid before splitted into two smaller trapezoids

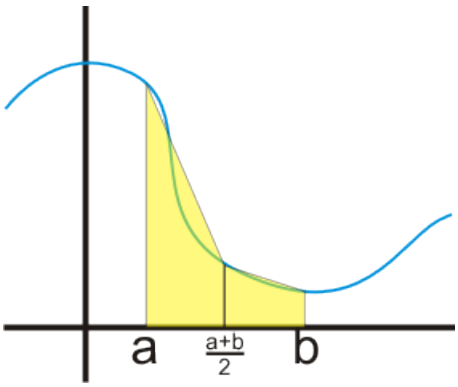

Figure 3: Trapezoid split into two smaller trapezoids of equal region length

3.4.1 Fully Dynamic

This variant starts with exactly 1 region, spanning from a to b. Hence, it performs the iteration as described above exactly $n - 1$ times. This, however, is rather unstable for certain classes of functions, since there may exist a large region represented by a large trapezoid.

3.4.2 Partially Dynamic

So, instead of going to the far extreme, the partial approach starts with $\frac{n}{2}$ initial regions of the same length, partitioning a through b into $\frac{n}{2}$ regions. Hence, we perform exactly $\frac{n}{2}$ iterations of the algorithm above.

Note that this hybrid approach does not necessarily need to start with $\frac{n}{2}$. We chose $\frac{n}{2}$ as to obtain the benefits of the dynamic (accuracy) and the non-dynamic (stability) approaches.

3.5 Complexity

The only non trivial part of the algorithm is during the select maximum error part. The naive implementation that represents the regions as a linked list will run in $O(n^2)$ time. This, however, can be improved to $O(n \log n)$ by storing the regions in a priority queue. Hence, we are able to get the region with maximum error in $O(\log n)$, to take it out of the priority queue in $O(\log n)$, and to insert two new regions in $O(\log n)$. Hence, the overall running time is $O(n \log n)$.

4 Experimental Results

4.1 Running Example

We will now approximate the value of our running example:

$$\int_1^{11} \frac{1}{x} \mathrm{d}x$$

The function is illustrated in Figure 4. The results of the approximations are shown in Figure 5. Furthermore, the case when the function is approximated using 10 Trapezoids is illustrated in Figure 6 through 8.

For this function, both Dynamic Trapezoidal Rules achieve better accuracy than the Regular Trapezoidal Rule. However, there are no noticable differences between the result of the Fully Dynamic Trapezoidal Rule with the Partially Dynamic one. Thus, there exists some

Trapezoids	Regular	Full	Partial	Exact
10	2.4744227994227996	2.423962621643628	2.423845598845599	2.39789527279837
100	2.398720889730268	2.3981571368047554	2.3981535242004837	2.39789527279837
1000	2.3979035371778394	2.3978978623359493	2.397897853805243	2.39789527279837

Figure 5: $\int_1^{11} \frac{1}{x} \mathrm{d}x$ Approximation Results

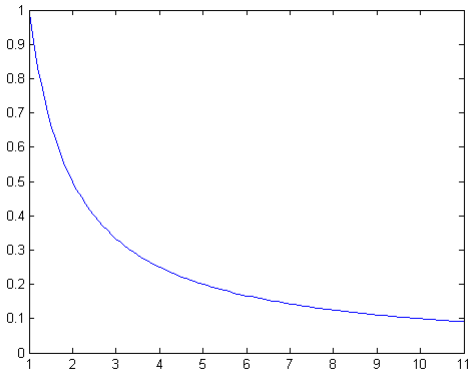

Figure 4: Plot of $f(x) = \frac{1}{x}$

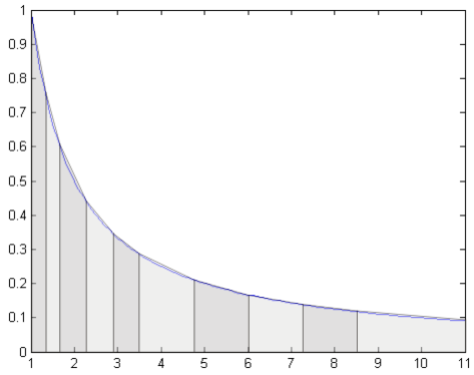

Figure 7: $f(x) = \frac{1}{x}$ approximated using Full Dynamic Trapezoid Rule with 10 trapezoids

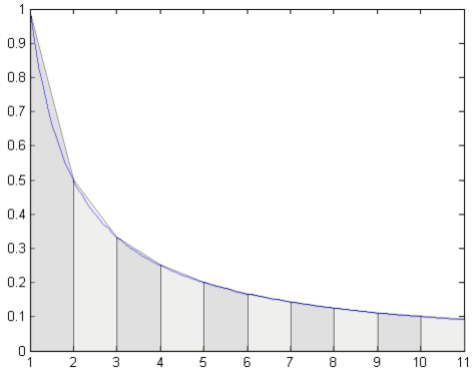

Figure 6: $f(x) = \frac{1}{x}$ approximated using Regular Trapezoid Rule with 10 trapezoids

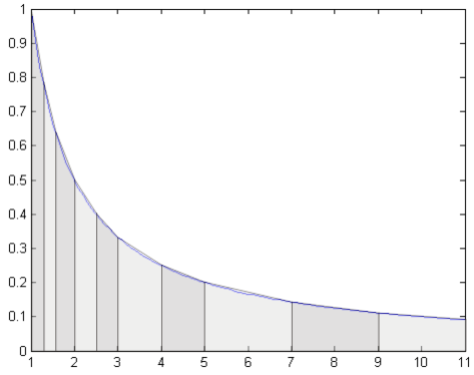

Figure 8: $f(x) = \frac{1}{x}$ approximated using Partially Dynamic Trapezoid Rule with 10 trapezoids

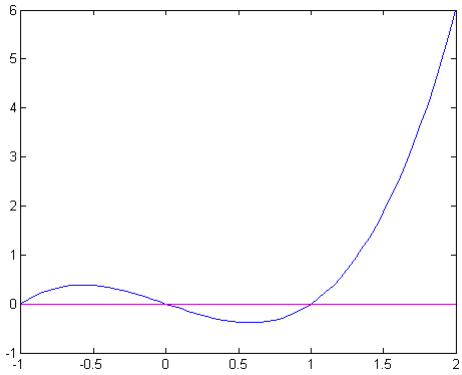

Figure 9: Plot of $f(x) = x^3 + x$

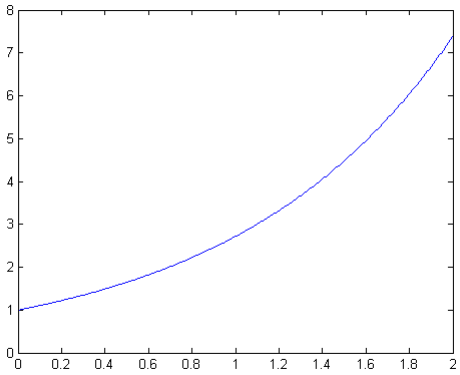

Figure 11: Plot of $f(x) = e^x$

set of functions for which the dynamic trapezoidal rules achieve better performance.

4.2 Polynomial Function

From the group of polynomial functions, we chose the following integral:

$$\int_{-1}^{2} x^3 + x \mathrm{d}x = \int_{-1}^{2} x(x-1)(x+1)\mathrm{d}x$$

The function is illustrated in Figure 9. The results of the approximations are shown in Figure 10.

For this function, again both Dynamic Trapezoidal Rules achieve better accurracy than the Regular Trapezoidal Rule. It is worth noting that for this function, the difference in accurracy is not very big. As with the previous function, there are no noticable differences between the result of the Fully Dynamic Trapezoidal Rule with the Partially Dynamic one. Again, this demonstrated that the dynamic trapezoidal rule achieves better performance on some functions.

4.3 Exponential Function

From the group of exponential functions, we chose the following integral:

$$\int_{-1}^{2} e^x \mathrm{d}x$$

The function is illustrated in Figure 11. The results of the approximations are shown in Figure 12.

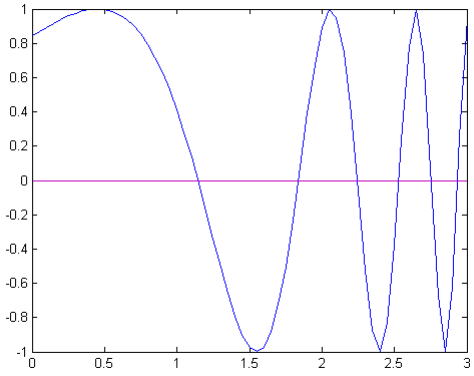

Figure 13: Plot of $f(x) = \sin(e^x)$

For this example, the approximation given by all three approaches are identical. However, the Full Dynamic Rule produces a slightly worse result. This demonstrates that the Full Dynamic approach is less stable on some functions.

4.4 Periodic Function

From the group of exponential functions, we chose the following integral:

$$\int_{0}^{3} \sin(e^x)\mathrm{d}x$$

The function is illustrated in Figure 13. The results of the approximations are shown in Figure 14.

Trapezoids	Regular	Full	Partial	Exact
10	5.317499999999999	5.29066875	5.2899627685546875	5.25
100	5.250675000000017	5.250535021875	5.250415306538343	5.25
1000	5.250006749999657	5.250005346654752	5.250005390072772	5.25

Figure 10: $\int_{-1}^{2} x^3 + x \mathrm{d}x$ Approximation Results

Trapezoids	Regular	Full	Partial	Exact
10	6.410338768199613	6.410955683541401	6.4103387681996145	6.38905609893
100	6.389269066047513	6.389271462915947	6.389269066047505	6.38905609893
1000	6.389058228615881	6.389058223688743	6.389058228615871	6.38905609893

Figure 12: $\int_{-1}^{2} e^x \mathrm{d}x$ Approximation Results

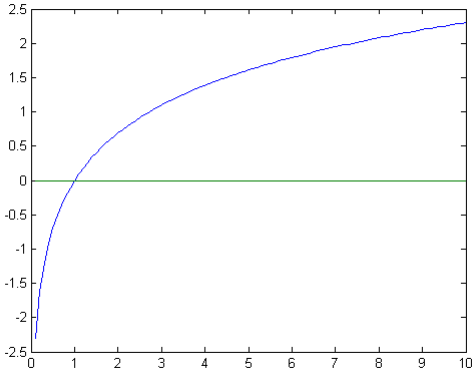

Figure 15: Plot of $f(x) = \log(x)$

For this function, again both Dynamic Trapezoidal Rules achieve slightly better accurracy than the Regular Trapezoidal Rule. It is worth noting that for this function, the difference in accurracy is not very big.

4.5 Logarithmic Function

From the group of logarithmic functions, we chose the following integral:

$$\int_{0.1}^{10} \log(x) \mathrm{d}x$$

Here, $\log(x)$ denotes the natural logarithm of x.

The function is illustrated in Figure 15. The results of the approximations are shown in Figure 16.

For this function, both Dynamic Trapezoidal Rules achieve better accurracy than the Regular Trapezoidal Rule.

5 Conclusion

For all our of experiments, the partially dynamic trapezoidal rule achieves same to better accurracy than the regular trapezoidal rule. The full dynamic trapezoidal rule is less stable in that respect. Hence, we have demonstrated a possible improvement to the Trapezoidal Rule that retains its simplicity.

We note however, that our algorithm requires the ability to evaluate the value of the function as some given points, unlike the original trapesium algorithm which can process a predefined set of values.

6 Appendix

6.1 Python Implementation of Algorithm

RegularTrapesiumIntegral implements the trapezoidal rule with equal lengths of regions.

PartialDynamicTrapesiumIntegral implements the trapezoidal rule with $\frac{n}{2}$ initial regions.

FullDynamicTrapesiumIntegral implements the trapezoidal rule with 1 initial region.

Note that this implementation runs in $O(n^2)$, for the sake of conciseness. Extending it to run in $O(n \log n)$ using priority queue should be immediate.

```
def TrapesiumCalculate(points, f):
```

Trapezoids	Regular	Full	Partial	Exact
10	0.8321589398367933	0.5242524831787573	0.5348750428263918	0.6061244734
100	0.6065832841134191	0.6062251519068935	0.6054346615572611	0.6061244734
1000	0.6061290186207896	0.6061171688714586	0.6061176534923454	0.6061244734

Figure 14: $\int_0^3 \sin(e^x)\mathrm{d}x$ Approximation Results

Trapezoids	Regular	Full	Partial	Exact
10	12.859761993925044	13.229542659488793	13.22898435745594	13.356109439
100	13.348242519243419	13.354870936068403	13.354899436643572	13.356109439
1000	13.35602860759894	13.356097327129387	13.356097336893054	13.356109439

Figure 16: $\int_{0.1}^{10} \log(x)\mathrm{d}x$ Approximation Results

```
  ret = 0.0
  for i in range(len(points)-1):
    ret += (f(points[i]) + \
            f(points[i+1])) \
           / 2 * (points[i+1] \
                  - points[i])
  return ret

def RegularTrapesiumIntegral(
    low, hi, n, f):
  assert n > 0
  assert low <= hi
  step = (hi - low) / n
  pos = low
  h = [low]
  for _ in range(n):
    pos += step
    h.append(pos)

  return TrapesiumCalculate(h, f)

def PartialDynamicTrapesiumIntegral(
    low, hi, n, f):
  assert n > 0
  assert n % 2 == 0
  assert low <= hi

  n //= 2

  h = [low]
  for i in range(n):
    h.append(low + (i+1) * \
             (hi - low)/n)
  for _ in range(n):
    # Find the largest error
    c = -1
    c_err = 0
    for j in range(len(h)-1):
      vt = f((h[j] + h[j+1])/2)
      err = (h[j+1] - h[j]) * \
            abs((vt - f(h[j])) - \
                (f(h[j+1]) - vt))
      if c == -1 or c_err < err:
        c_err = err
        c = j

    mid = (h[c] + h[c+1]) / 2
    c += 1
    h = h[:c] + [mid] + h[c:]

  return TrapesiumCalculate(h, f)

def FullDynamicTrapesiumIntegral(
    low, hi, n, f):
  assert n > 0
  assert low <= hi

  h = [low, hi]
  n -= 1

  for _ in range(n):
    # Find the largest error
    c = -1
    c_err = 0
    for j in range(len(h)-1):
```

```
            vt = f((h[j] + h[j+1])/2)
            err = (h[j+1] - h[j]) * \
                  abs((vt - f(h[j])) - \
                      (f(h[j+1]) - vt))
            if c == -1 or c_err < err:
              c_err = err
              c = j

        # Calculate midpoint
        mid = (h[c] + h[c+1]) / 2
        c += 1
        h = h[:c] + [mid] + h[c:]

    return TrapesiumCalculate(h, f)
```

6.2 Declaration of Non Plagiarism

I hereby confirm that this paper is the product of my own work and is not an excerpt and/or translation of the work of other entities.

Bandung, May 6th 2012

Irvan Jahja
13509099

References

[1] Begnaud Francis Hildebrand. *Introduction to numerical analysis: 2nd edition*. Dover Publications, Inc., New York, NY, USA, 1987.

[2] John H. Matthews. *Simpson's 3/8 Rule for Numerical Integration*. California State University, Fullerton, 2004.

[3] Gordon K. Smyth. Numerical integration.