Assignment 5

1. Approximate the integral $\int_{13}^{15} \sin(x)\,\mathrm{d}x$ using each of the approximations we described using $h = 1$ and then again using $h = 0.5$. These include Riemann sums, the trapezoidal rule, Simpson's rule, our $O(h^5)$ centered rule, and the backwards 3-point and 4-point rules. Are the approximations using centered interpolating polynomials as accurate as those using only points to the left?

From Calculus, you know the correct answer is cos(13) – cos(15) = 1.667134694309018 to sixteen significant digits. Which is the most accurate formula?

Here is MATLAB code that calculates most of these:

```
a = 13.0;
b = 15.0;
soln = -cos(b) - (-cos(a));
printf( "Actual integral:  %.10f\n", soln );

for n = [2 4]
  h = (b - a)/n;
  rs = 0.0;
  tr = 0.0;
  c5 = 0.0;
  b3 = 0.0;
  b4 = 0.0;

  for k = 1:n
    x = a + k*h;

    rs = rs + sin(x);
    tr = tr + sin(x - h) +         sin(x);
    c5 = c5 - sin(x - 2*h) + 13.0*sin(x - h) +   13.0*sin(x) - sin(x + h);
    b3 = b3 - sin(x - 2*h) +  8.0*sin(x - h) +    5.0*sin(x);
    b4 = b4 + sin(x - 3*h) –  5.0*sin(x - 2*h) + 19.0*sin(x - h) + 9.0*sin(x);
  end

  rs = rs*h;
  tr = tr*h/2.0;
  c5 = c5*h/24.0;
  b3 = b3*h/12.0;
  b4 = b4*h/24.0;

  printf( "Riemann sum:      %.10f\n", rs );
  printf( "           Error: %.10f\n", abs( rs - soln ) );
  printf( "Trapezoidal rule: %.10f\n", tr );
  printf( "           Error: %.10f\n", abs( tr - soln ) );
  printf( "Centered order 5: %.10f\n", c5 );
  printf( "           Error: %.10f\n", abs( c5 - soln ) );
  printf( "Backward 3-point: %.10f\n", b3 );
  printf( "           Error: %.10f\n", abs( b3 - soln ) );
  printf( "Backward 4-point: %.10f\n", b4 );
  printf( "           Error: %.10f\n", abs( b4 - soln ) );
end

sr = (sin(13) + 4*sin(14) + sin(15))/6.0*2;
printf( "Simpson's rule:   %.10f\n", sr );
printf( "           Error: %.10f\n", abs( sr - soln ) );
sr = (sin(13) + 4*sin(13.5) + 2*sin(14) + 4*sin(14.5) + sin(15))/6.0*1;
printf( "Simpson's rule:   %.10f\n", sr );
>> printf( "           Error: %.10f\n", abs( sr - soln ) );
```

Here is C++ code that calculates most of these:

```
double a{ 13.0 };
double b{ 15.0 };
double soln{ -std::cos(b) - (-std::cos(a)) };
std::cout << "Actual integral:  " << soln << std::endl;

for ( unsigned int n{ 2 }; n <= 4; n += 2 ) {
  double h{ (b - a)/n );
  double rs{ 0.0 };
  double tr{ 0.0 };
  double c5{ 0.0 };
  double b3{ 0.0 };
  double b4{ 0.0 };

  for ( unsigned int k{ 1 }; k <= n; ++k ) {
    double x{ a + k*h };

    rs +=  std::sin(x);
    tr +=  std::sin(x - h)     +      std::sin(x);
    c5 += -std::sin(x – 2.0*h) + 13.0*std::sin(x - h) +     13.0*std::sin(x) -         std::sin(x + h);
    b3 += -std::sin(x – 2.0*h) +  8.0*std::sin(x - h) +      5.0*std::sin(x);
    b4 +=  std::sin(x – 3.0*h) –  5.0*std::sin(x – 2.0*h) + 19.0*std::sin(x - h) + 9.0*std::sin(x);
  }

  rs *= h;
  tr *= h/2.0;
  c5 *= h/24.0;
  b3 *= h/12.0;
  b4 *= h/24.0;

  std::cout << "Riemann sum:      " << rs
  std::cout << "           Error: " << std::abs( rs - soln ) << std::endl;
  std::cout << "Trapezoidal rule: " << tr
  std::cout << "           Error: " << std::abs( tr - soln ) << std::endl;
  std::cout << "Centered order 5: " << c5
  std::cout << "           Error: " << std::abs( c5 - soln ) << std::endl;
  std::cout << "Backward 3-point: " << b3
  std::cout << "           Error: " << std::abs( b3 - soln ) << std::endl;
  std::cout << "Backward 4-point: " << b4
  std::cout << "           Error: " << std::abs( b4 - soln ) << std::endl;
}
```

The output is

```
Actual integral:  1.6671346943

Riemann sum:      1.6408951959
           Error: 0.0262394985
Trapezoidal rule: 1.5258347942
           Error: 0.1412999001
Centered order 5: 1.6427385836
           Error: 0.0243961107
Backward 3-point: 1.6339230834
           Error: 0.0332116109
Backward 4-point: 1.6924265209
           Error: 0.0252918266
Simpson's rule:   1.6776280999
           Error: 0.0104934056

Riemann sum:      1.6897873390
           Error: 0.0226526447
Trapezoidal rule: 1.6322571381
           Error: 0.0348775562
Centered order 5: 1.6655599277
           Error: 0.0015747666
Backward 3-point: 1.6643861444
           Error: 0.0027485499
Backward 4-point: 1.6694930783
           Error: 0.0023583840
Simpson's rule:   1.6677312528
           Error: 0.0005965585
```

The 4-point backward divided-difference rule is less accurate than the 4-point centered divided-difference formula, as expected. Simpson’s rule, which awkwardly spans two intervals, is the most accurate.

2. Suppose we have a function that is piecewise constant, but discontinuous, so that $f(a) = 1$ and $f(b) = 0$, and somewhere between a and b, the value of the function drops from 1 to 0. We don't know exactly when between a and b the function f drops from 1 to 0, so what is the minimum and maximum possible values of the integral $\int_a^b f(x)\,dx$? Use each of the formulas that estimate the integral of a function over one interval, including the trapezoidal rule, the O(h^5) centered rule, the O(h^4) three-point backward rule and the O(h^5) four-point backward rule. Recall some values may be outside the range $[a, b]$, so assume $f(x) = 1$ for $x < a$ and $f(x) = 0$ for $x > b$.

Which formula would you say is the best approximation?

If the discontinuity is close to a, then the integral is close to zero, while if the discontinuity is close to be, then the integral is close to $b - a$. Approximating this integral with each of these techniques results in

1. $½(1 + 0)(b - a) = 0.5(b - a)$
2. $\frac{-1+13+0+0}{24}(b - a) = 0.5(b - a)$
3. $\frac{-1+8+0}{12}(b - a) = 0.58333\cdots(b - a)$
4. $\frac{1-5+19+0}{24}(b - a) = 0.625(b - a)$

Of these, the first two provide the minimum error because the approximation is exactly between the minimum and maximum values of the integral; however, the second formula does require future data (that is, data to the right of the interval).

3. Given the readings from a sensor that are being taken periodically,

6.2615, 6.8847, 7.4471, 8.0392, 8.6836

where the last is the most recent reading, do the following with least-squares best-fitting linear polynomials:

a. approximate the value of the underlying signal at the time of the last reading,
b. estimate the value of the underlying signal one time step into the future,
c. estimate the rate of change of the underlying signal assuming the readings are being taken once every 10 seconds, and
d. estimate the integral over the most recent time step of the underlying signal, again, assuming the readings are being taken once every 10 seconds.

Solving $A^TA\mathbf{c} = A^T\mathbf{y}$ where $A = \begin{pmatrix} 0 & 1 \\ -1 & 1 \\ -2 & 1 \\ -3 & 1 \\ -4 & 1 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} 8.6836 \\ 8.0392 \\ 7.4471 \\ 6.8847 \\ 6.2615 \end{pmatrix}$,

you get the interpolating polynomial $0.59987t + 8.66296$, and thus we have:

a. $b = 8.66296$
b. $a + b = 9.26283$
c. $a/10 = 0.059987$ units per second
d. $10(b - a/2) = 83.63025$ unit seconds

4. Given the readings from a sensor that are being taken periodically,

3.786, 3.2866, 2.5966, 1.6497, 0.5556

where the last is the most recent reading, do the following with least-squares best-fitting quadratic polynomials:

a. approximate the value of the underlying signal at the time of the last reading,
b. estimate the value of the underlying signal one time step into the future,
c. estimate the rate of change of the underlying signal assuming the readings are being taken once every 10 seconds, and
d. estimate the integral over the most recent time step of the underlying signal, again, assuming the readings are being taken once every 10 seconds.

Solving $A^{\mathrm{T}}A\mathbf{c} = A^{\mathrm{T}}\mathbf{y}$ where $A = \begin{pmatrix} 0 & 0 & 1 \\ 1 & -1 & 1 \\ 4 & -2 & 1 \\ 9 & -3 & 1 \\ 16 & -4 & 1 \end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix} 0.5556 \\ 1.6497 \\ 2.5966 \\ 3.2866 \\ 3.7860 \end{pmatrix}$, you get the interpolating polynomial $-0.10330714285714 26t^2 - 1.222998571428570t + 0.5487457142857153$, and thus we have:

a. $c = 0.5487457142857153$
b. $a + b + c = -0.77756$
c. $b/10 = -0.1222998571428570$ units per second
d. $10(a/3 - b/2 + c) = 1.125809285714286$ unit seconds

5. In Question 4, what would be your best estimate as to when the underlying signal will be zero?

Because b is negative, the smaller root can be calculated with the formula $\frac{-2c}{b-\sqrt{b^2-4ac}}$, which yields the value 0.4328616173640030 but this is in scaled time, so in real time, the zero should occur approximately 4.33 seconds into the future.