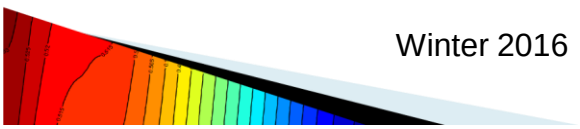
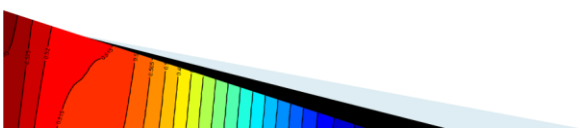
GNG 1106
Fundamentals of Engineering Computation
Numerical Methods



Winter 2016

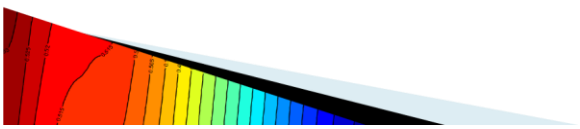
Topic 1: Root Finding



Root Finding Numerical Methods

Statement of the problem

- Engineering problems often require the roots of a function as a solution.
  - Designing a control system for a robotic arm
  - Designing spring and shock absorbers for an automobile
  - Analyzing the response of a motor
- Create a function that can find the real roots of polynomials in a given interval.
  - The function will receive the coefficients of a polynomial, fill in an array with the root values, and return the number of roots found.



Background: Roots of Polynomials of Degree 3

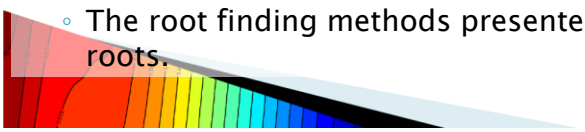
- A polynomial is a function of one variable having the following general form:

$$f(x) = c_0 x^N + c_1 x^{N-1} + c_2 x^{N-2} + \cdots + c_{N-1} x + c_N$$

- You will see in lab 7 that we can represent the polynomial of any degree using an array and a function to compute its value
- The general form of a polynomial of degree 3 is:

$$f(x) = c_0 x^3 + c_1 x^2 + c_2 x + c_3$$

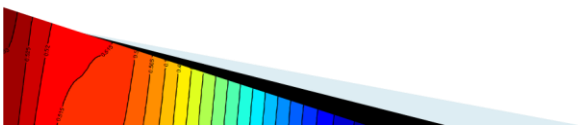
- Recall that the roots of a polynomial f(x) are the values of x for which f(x) = 0.
  - A polynomial of degree N has exactly N roots which can be real or complex valued.
  - Complex roots always appear in complex conjugate pairs.
    - E.g.: $x_1 = a + jb$ and $x_2 = a - jb$ where $j = \sqrt{-1}$
  - The root finding methods presented here only finds real roots.



- In the case of polynomials of degree 2 or 1, the roots are easily observed by factoring the polynomial.
  - E.g.: $f(x) = x^2 + x - 6$ (degree 2)
    $\Rightarrow f(x) = (x-2)(x+3)$

    $\therefore$ the roots are 2 and –3.
  - E.g.: $f(x) = 2x - 4$ (degree 1)
    $\Rightarrow f(x) = 2(x-2)$

    $\therefore$ the root is 2.



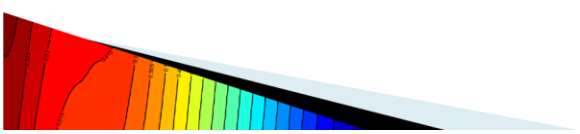
- The quadratic equation can also be used to determine the roots of polynomials of degree 2:

$$f(x) = ax^2 + bx + c$$

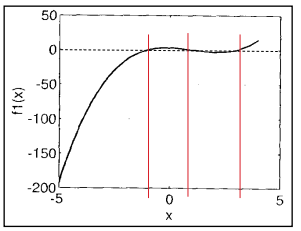
has the roots: $x_1 = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$ and $x_2 = \dfrac{-b - \sqrt{b^2 - 4ac}}{2a}$

  - E.g.: $f(x) = x^2 + 3x + 3$ has the pair of complex conjugate roots:

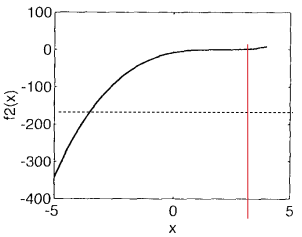
$$x_1 = -1.5 + j0.87 \quad \text{and} \quad x_2 = -1.5 - j0.87$$



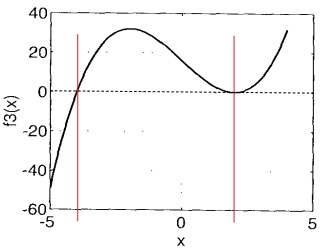
- A polynomial of degree 3 has exactly 3 roots. If the coefficients of the polynomial are real, then we only have 4 possible cases for the roots:
  - **Case 1**: Three distinct real roots.
    - E.g.: $f_1(x) = x^3 - 3x^2 - x + 3$
      $\Rightarrow f_1(x) = (x-3)(x+1)(x-1)$
      - $\therefore$ the roots are 3, −1, +1.



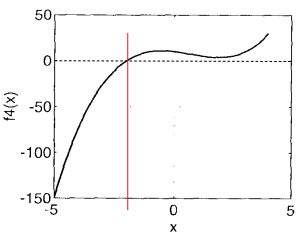
  - **Case 2**: Three identical real roots.
    - E.g.: $f_2(x) = x^3 - 6x^2 + 12x - 8$
      $\Rightarrow f_2(x) = (x-2)^3$
      - $\therefore$ the roots are +2, +2, +2
      - (the root +2 is a triple root).



  - **Case 3**: One distinct real root and 2 identical real roots.
    - E.g.: $f_3(x) = x^3 - 12x + 16$
      $\Rightarrow f_3(x) = (x+4)(x-2)^2$
      $\therefore$ the roots are −4, +2, +2
      (the root +2 is a double root).



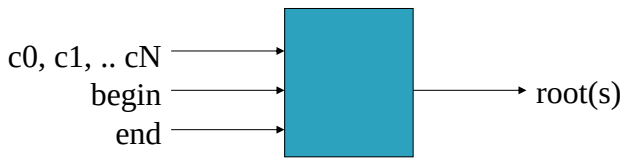
  - **Case 4**: One real root and a pair of complex conjugate roots.
    - E.g.: $f_4(x) = x^3 - 2x^2 - 3x + 10$
      $\Rightarrow f_4(x) = (x+2)[x-(2+j)][x+(2-j)]$
      $\therefore$ the roots are −2, −2+j, −2−j.



These concepts on roots apply to higher degree polynomials.

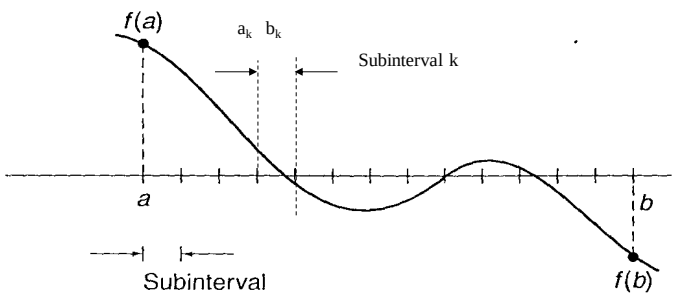
I/O Description

- Inputs:
  - Coefficients: c_0, c_1, c_2, … c_N to specify the polynomial:
    $$f(x) = c_0x^N + c_1x^{N-1} + c_2x^{N-2} + \cdots + c_{N-1}x + c_N$$
  - The beginning and end of the interval to search: begin, end
- Output:
  - The real roots found in the search interval



Test cases and Design
Incremental Search Technique

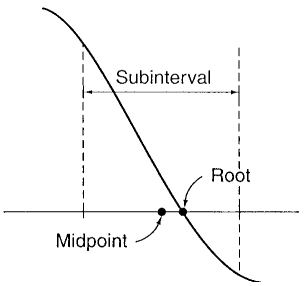
- A computer program that could determine the roots of a polynomial is of obvious interest. The incremental search technique is a simple numerical technique that can be used to determine the real roots of a polynomial. This technique operates as follows:
  - We delimit a search interval a ≤ x ≤ b where we suspect the existence of a real root (use graphing to find such intervals).
  - The interval a≤x≤b is divided into a number of equal subintervals.



Incremental Search Technique (continued)

- Each subinterval is verified for a change in sign in f(x).
  - For example, over subinterval k, we have $f(a_k)$ which is positive and $f(b_k)$ which is negative; a root is thus present for x in the range $a_k \le x \le b_k$.
- We can now simply approximate this root by the value of x located at the midpoint of the subinterval.

➔This approximation becomes more accurate as the size of the subinterval decreases.



- The change in sign in f(x) over a subinterval is easily detected by testing the product $f(a_k) \cdot f(b_k)$; this product will be negative if a change in sign has occurred.
- If the point a_k is on or very close to a root, then $f(a_k)$ will be approximately zero. The method can still detect this case since the product $f(a_k) \cdot f(b_k)$ will also be approximately zero, and $f(a_k)$ and $f(b_k)$ can then be verified individually for a root.
  - N.B.: Real numbers in a computer (`double`) are rarely exactly equal to zero after a function evaluation.

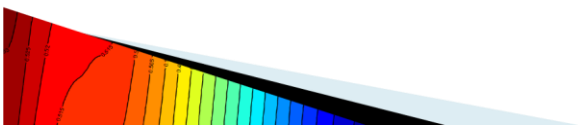
    ➔We usually apply a condition something like:

    ```
    if (real_var < SMALL_NUMBER)
    ```

    instead of testing:

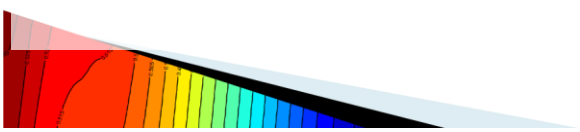
    ```
    if (real_var == 0.0)
    ```

- The incremental search technique can fail if:
  - the initial search interval $a \le x \le b$ is ill-chosen,
  - the subintervals are too large,
  - there exists a double real root, since a change in sign in f(x) does not occur in this case.



Case 1: f(x) = 2x – 4

- Lets apply the method to the polynomial f(x) = 2x – 4
  - We choose as search interval $1 \le x \le 3$ and divide it into subintervals of 0.5
- Our inputs are:
  c2 = 2, c3 = –4,
  begin = 1. end = 3
  Subinterval 1: $a_1 = 1.0$ and $b_1 = 1.5$
  $f(a_1) \cdot f(b_1) = (-2)(-1) = +2$
  Positive, so there is no root here.
  Subinterval 2: $a_2 = 1.5$ and $b_2 = 2.0$
  $f(a_2) \cdot f(b_2) = (-1)(0) = 0$
  Zero, so we have a root on an extremity.
  $f(a_2) = -1$ so not a root;
  $f(b_2) = 0$ so we have a root at $b_2 = 2.0$.



Case 1: f(x) = 2x – 4 (continued)

Subinterval 3: $a_3 = 2.0$ and $b_3 = 2.5$

$f(a_3) \cdot f(b_3) = (0)(1) = 0$

Zero, so we have a root on an extremity.

$f(a_3) = 0$ so we have a root at $a_3 = 2.0$; already detected!

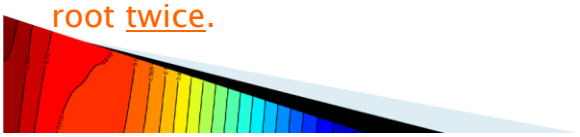
$f(b_3) = 1$ so not a root.

Subinterval 4: $a_4 = 2.5$ and $b_4 = 3.0$

$f(a_4) \cdot f(b_4) = (1)(2) = +2$

Positive, so there is no root here.

- We have correctly detected the root at $x = 2.0$, which has fallen directly on an extremity, given our choice of subintervals. We must be careful not to detect the same root twice.



Case 2: f(x) = 2x – 4

- Let's again apply this method to the function f(x) = 2x – 4 but using subintervals having a spacing of 0.3
- Our inputs are: c0 = 2, c1 = –4, begin = 1, end = 3

Subinterval 1: $a_1 = 1.0$ and $b_1 = 1.3$

$f(a_1) \cdot f(b_1) = (-2)(-1.4) = +2.8$

Positive, so there is no root here.

Subinterval 2: $a_2 = 1.3$ and $b_2 = 1.6$

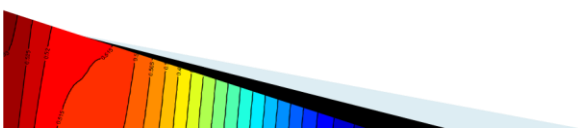
$f(a_2) \cdot f(b_2) = (-1.4)(-0.8) = +1.12$

Positive, so there is no root here.

Subinterval 3: $a_3 = 1.6$ and $b_3 = 1.9$

$f(a_3) \cdot f(b_3) = (-0.8)(-0.2) = +1.6$

Positive, so there is no root here.



Case 2: f(x) = 2x – 4

Subinterval 4: $a_4 = 1.9$ and $b_4 = 2.2$

$f(a_4) \cdot f(b_4) = (-0.2)(0.4) = -0.08$

Negative, so there is a root in this interval.

Approximate the root at the midpoint: $(1.9+2.2)/2 = 2.05$.

Subinterval 5: $a_5 = 2.2$ and $b_5 = 2.5$

$f(a_5) \cdot f(b_5) = (0.4)(1) = +0.4$

Positive, so there is no root here.

Subinterval 6: $a_6 = 2.5$ and $b_6 = 2.8$

$f(a_6) \cdot f(b_6) = (1)(1.6) = +1.6$

Positive, so there is no root here.

Subinterval 7: $a_7 = 2.8$ and $b_7 = 3.1 \rightarrow b_7 = 3.0$

$f(a_7) \cdot f(b_7) = (1.6)(2.0) = +3.2$

Positive, so there is no root here.

Note how the use of hand solved cases can help define the method (algorithm).

- We have identified the root at $x = 2.05$.
- Careful with the last interval: $b_7 = 3.1$ falls outside of our search interval $1 \le x \le 3$ so we set $b_7 = 3.0$ before evaluating $f(b_7)$

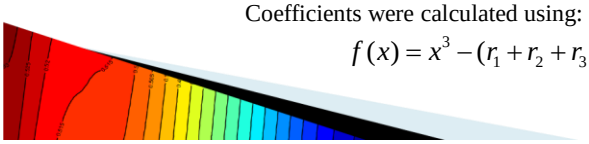
Test Cases

- The presented test cases illustrated how to find the process of finding the roots.
- The sub-intervals are rather large, which leads to less precision (e.g. finds root at 2.05 instead of 2.0).
- Can get the computer to use much smaller sub-intervals, for example, divide the search interval by 1000 (define SI_RESOLUTION symbolic constant to 1000).
- Example test cases:

Polynomial	Roots
$f(x) = 2x - 4$	2.0
$f(x) = x^2 + 4.3x + 2.1$	-0.5618, -3.7382
$f(x) = x^3 - 4.65x^2 - 24.73x + 1.248$	-3.2, 0.05, 7.8
$f(x) = 0.5x^3 - 2.325x^2 - 12.365x + 0.624$	-3.2, 0.05, 7.8

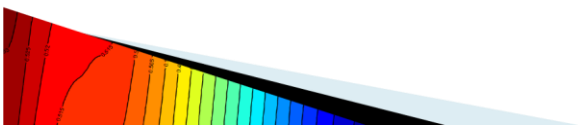
Coefficients were calculated using:

$$f(x) = x^3 - (r_1 + r_2 + r_3)x^2 + (r_1 r_2 + r_2 r_3 + r_1 r_3)x - r_1 r_2 r_3$$



Design

- Use the following functions:
  - `findAllRoots` – scans all sub-intervals in the interval and calls `findRoot` to see if a root exists in each sub-interval
  - `findRoot` – Checks if a root exists in the subinterval.
  - `poly` – function used to compute the value of the polynomial given an array of coefficients.



19

```
int findAllRoots(double start, double end,
                 int n, double coefficients[],
                 double roots[])
```

Parameters:

`start, end`: start and end of interval (x values) for searching roots

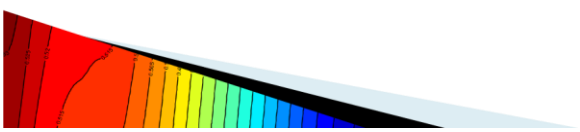
`n` – number of coefficients

`coefficients` – reference to array of coefficients

`roots` – reference to array for storing roots

Return Value:

The number of roots found.



20

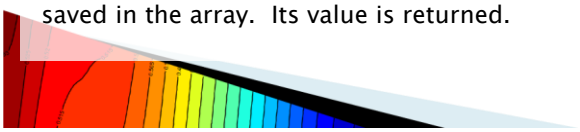
```
int findAllRoots(double start, double end,
                 int n, double coefficients[],
                 double roots[])
```

Logic/Algorithm

This function finds all the roots on the interval defined between `start` and `end`. The function first defines a subinterval size as `subinter = (end-start)/SI_RESOLUTION`. A determinant loop, using `cntr`, is then used to evaluate each subinterval as follows:

- The limits of the subinterval is defined with
- `ak = start + (cntr*subinter)` and
- `bk = start + (cntr+1)*subinter` (if `bk` is greater than `end`, it is set to `end`)
- The function `findRoot` is called to see if the subinterval contains and if it does,
  - the root is saved in the roots array using the index rootIx and
  - rootIx is incremented.

Upon exiting the loop, the index `rootIx` contains the number of roots saved in the array. Its value is returned.




```
int findRoot(double left, double right, int n,
             double coeffs[], double *root)
```

Parameters:

`left, right`: values of x at the edges of the subinterval

`n` – number of coefficients

`coeffs` – reference to array of coefficients

`root` – pointer to variable for saving the root value.

Return Value: TRUE if a root exists in the sub-interval and FALSE otherwise.

Logic/Algorithm

Compute f(x) with call to `poly` at left edge and save in `fx_left`

Compute f(x) with call to `poly` at right edge and save in `fx_right`

Save `fx_left*fx_right` to `factor`

Set `retVal` to `FALSE`,

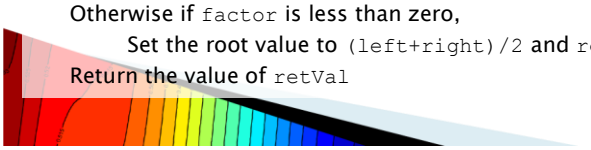
If absolute value of `factor` is less than `ALMOST_0` (10^{-4}), then check if root is at left edge (absolute value of `fx_left` less than `ALMOST_0`).

Set the root value to `left` and `retVal` to `TRUE`.

Otherwise if `factor` is less than zero,

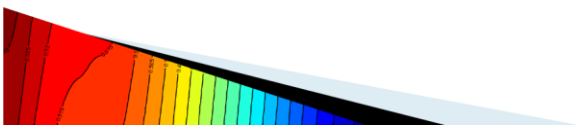
Set the root value to `(left+right)/2` and `retVal` to `TRUE`

Return the value of `retVal`



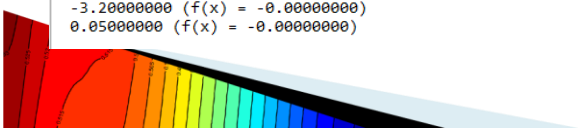
Implementation

- See the CodeBlocks Project IncFindRoots
- The function `main` is used to run a number of test cases
  - It calls the function `testCase` to find the roots and display results for a given input.

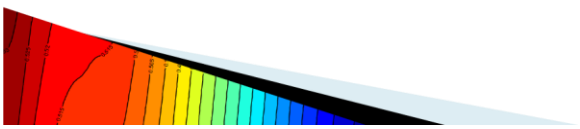


Testing and Verification

```
F:\UofO\Courses\CurrentCourses\GNG1106\Fall2016\Notes\GNG1106Code\Module8\IntSearchRootFinding\bin\Debug\IntSearchRootFind

Finding roots for f(x) = 2x - 4 in interval between 0.0 and 5.0
Found 1 roots:
  2.00000000 (f(x) = 0.00000000)

Finding roots for f(x) = x^2 + 4.3x + 2.1 in interval between 0.0 and 5.0
Found 2 roots:
  -3.73500000 (f(x) = -0.01027500)
  -0.56500000 (f(x) = -0.01027500)

Finding roots for f(x) = x^3 - 4.65x^2 - 24.73x + 1.248 in interval between -26 and 7.9
Found 3 roots:
  -3.20225000 (f(x) = -0.08050965)
  0.05215000 (f(x) = -0.05417392)
  7.81525000 (f(x) = 1.30442659)

Finding roots for f(x) = 0.5x^3 - 2.325x^2 - 12.365x + 0.624 in interval between -26 and 7.9
Found 3 roots:
  -3.20225000 (f(x) = -0.04025483)
  0.05215000 (f(x) = -0.02708696)
  7.81525000 (f(x) = 0.65221330)

Finding roots for f(x) = 0.5x^3 - 2.325x^2 - 12.365x + 0.624 in interval between -4 and 1
Found 2 roots:
  -3.20000000 (f(x) = -0.00000000)
  0.05000000 (f(x) = -0.00000000)
```

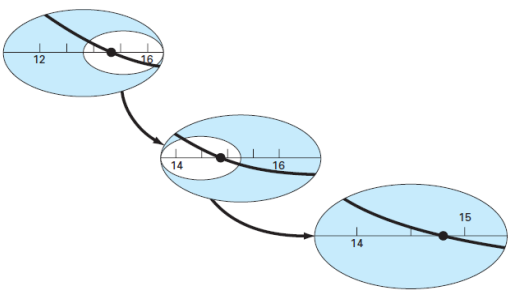


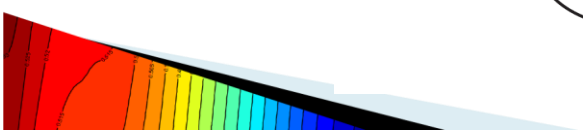
Another Root Finding Numerical Method: Bisection Method



Bisection Root Finding Method

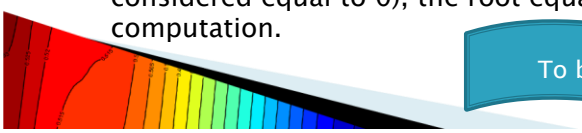
- Interval is repeatedly divided in half.
  - Faster than incremental search, but can only find one root
  - Function must change sign over original interval
  - Evaluate the function at the midpoint of the interval – assume this to be the root estimate.
  - Select the half that contains the root (i.e. half where the function changes sign) and repeat until desired accuracy is reached.





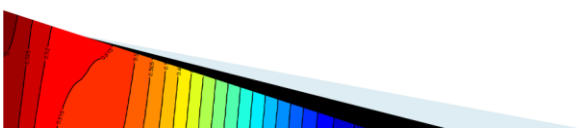
Bisection Root Finding Method

- Algorithm:
  - Step 1: Choose the search interval: x_L lower edge (left) and x_U upper edge (right) such that the function changes sign over the interval. This can be checked by ensuring that $f(x_L)f(x_U) < 0$.
  - Step 2: An estimate of the root x_R is determined by $x_R = (x_L + x_U)/2$.
  - Step 3: Make the following evaluations to determine in which subinterval the root lies:
    - (*a*) If $f(x_L)f(x_R) < 0$, the root lies in the lower subinterval. Therefore, set $x_U = x_R$ and return to step 2.
    - (*b*) If $f(x_L)f(x_R) > 0$, the root lies in the upper subinterval. Therefore, set $x_L = x_R$ and return to step 2.
    - (*c*) If $|f(x_L)f(x_R)|$ < some small tolerance value (that is, can be considered equal to 0), the root equals x_R; terminate the computation.



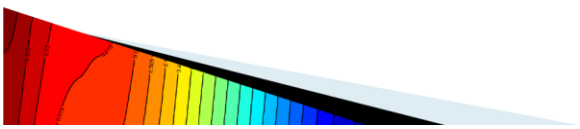
To be applied in Lab 7

Topic 2: Euler's Method – Solving Differential Equations



Solving Differential Equation

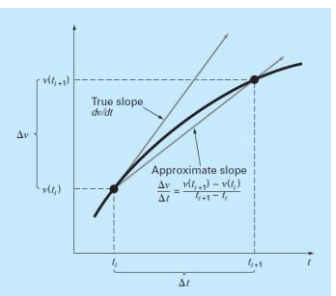
- Engineering describes many systems using a set of differential equations.
  - For example, for the parachutists example, the following differential equation can be used to represent how velocity changes over time: $$\frac{dv}{dt} = \frac{mg - cv}{m}$$ Equation 1
  - Recall that the derivative of the velocity equal acceleration and that acceleration is F/m, where F is force and m the mass
  - In the above expression:
    - mg is the force due to gravity
    - cv is the force due to the drag of the parachute
- The velocity function, v(t), can be solved analytically using calculus (which has been done in your labs).
  - Not always possible, so we would like to solve the DE directly.

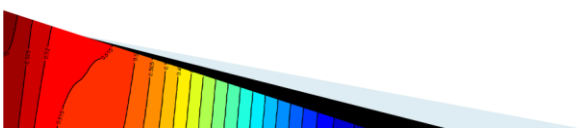


Euler's Numerical Method for Solving Differential Equations

- Recall the definition of the derivative from calculus
$$\frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t}$$ Equation 2
- The derivative represents the slope of the function at time t
- To create a numerical method consider:
  - Vary time in small steps: $t_0\, t_1\, t_2 \ldots t_i \ldots t_f$ where $\Delta t = t_{i+1} - t_i$ for $1 < i < f$ (t_0 must be given)
  - Change in v can be defined as: $\Delta v = v_{i+1} - v_i$
- The derivative can be approximated by:

$$\frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{v_{i+1} - v_i}{\Delta t}$$ Equation 3





Euler's Numerical Method for Solving Differential Equations

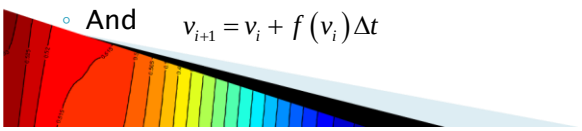
- If the differential equation has the form $\frac{dv}{dt} = f(v)$ Equation 4
- Then Equation 3 and Equation 4 can be combined to give the following result

$$\frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{v_{i+1} - v_i}{\Delta t} = f(v_i)$$

$$v_{i+1} - v_i = f(v_i)\Delta t$$

$$v_{i+1} = v_i + f(v_i)\Delta t \qquad \text{Equation 5}$$

- Equation 5 is knows as the difference equation and can be used to approximate the solution to the differential equation with the form of Equation 4
- Thus if the value of velocity, v_0, is given at time t_0, then it is possible to calculate all v_i for all t_i $(1 < i < f)$ for a given Δt
  - Recall that: $t_{i+1} = t_i + \Delta t$
  - And $v_{i+1} = v_i + f(v_i)\Delta t$



Euler's Numerical Method for The Parachutist Problem

- Recall the differential equation for velocity

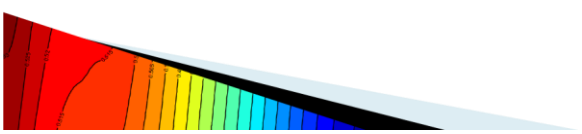
$$\frac{dv}{dt} = f(v) = \frac{mg - cv}{m} \qquad \text{Equation 6}$$

- The difference equation for Equation 6 is:

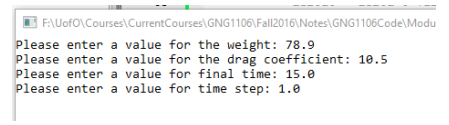
$$v_{i+1} = v_i + f(v_i)\Delta t = v_i + \frac{mg - cv_i}{m}\Delta t \qquad \text{Equation 7}$$

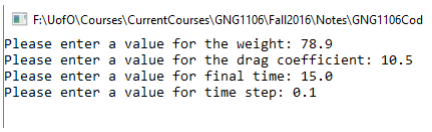
- How can you fill in two arrays, one for storing values of time (t_i) and one for storing values of velocity (v_i)?

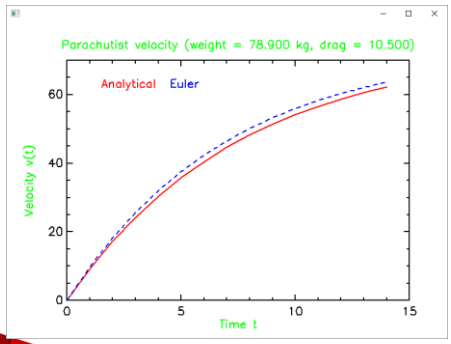
- See the CodeBlocks project ParachutistEuler.

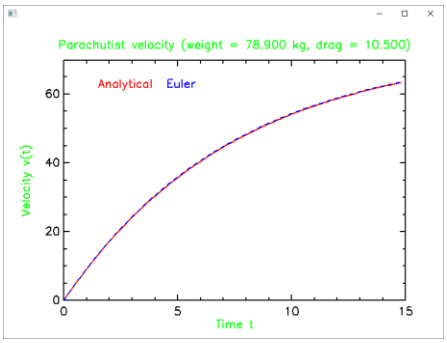


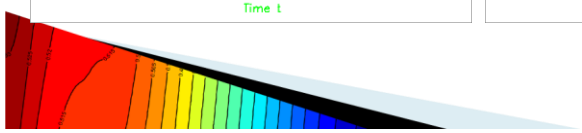
Project ParachutistEuler





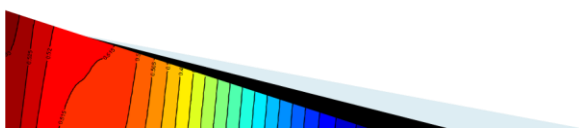






33

Topic 3: Trapezoidal Rule for Integration



34

Solving Definite Integrals

- Engineering also used integration for solving problems.
  - For example, for the parachutists example, we want to find the distance after time t. We have the velocity of the parachutist given as

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right) \quad \text{Equation 1}$$

  - To find the distance the velocity is integrated as follows.

$$d(t) = \int_0^t v(u)du = \frac{gm}{c}\int_0^t \left(1 - e^{-(c/m)u}\right)du \quad \text{Equation 2}$$

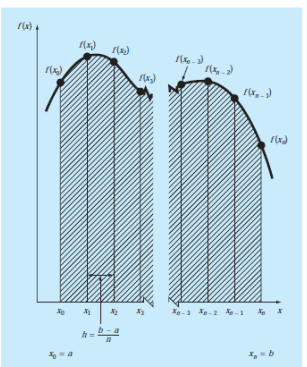
- Again we can solve the integral, but the computer can solve the integral directly using a numerical method.
  - For some problems it is not always possible to find the analytical solution.
  - The analytical solution for d is:

$$d(t) = \frac{gm}{c}\left[t - \frac{m}{c}\left(1 - e^{-(c/m)t}\right)\right] \quad \text{Equation 3}$$

- The Trapezoidal rule numerical method is presented.

Trapezoidal Rule

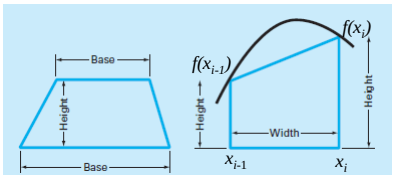
- Recall that the definite integral of $f(x)$ gives the area underneath the curve of a function
  - The gray area in the figure shows the area for $f(x)$ between the limits a and b.
  - As in the case of Euler's method, the range of x between a and b are subdivided into steps of width $h = (b - a)/n$,
  - That is, x is varied in small steps: $x_0\ x_1\ x_2\ \ldots\ x_i\ \ldots x_n$ where $\Delta x = t_{x+1} - t_x$ for $1 < i < f$ (x_0 must be given)
  - The area under the curve between any two values of x, a_i, can be using a trapezoid, presented in the next slide.
  - The integral between the limits of a and b are the sum of all areas a_i ($0 \le i \le n - 1$)



$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} a_i \quad \text{Equation 4}$$

Area of the Trapezoid

- The area of a trapezoid is equal to the average of the bases times the height
  - Trapezoids in the integration intervals are on there sides, and thus the area is:

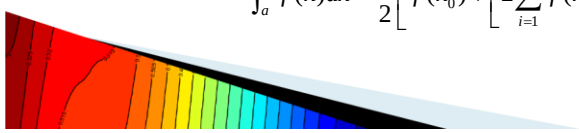


$$a_i = Width \times Avg(Height)$$

$$= (x_i - x_{i-1})\left(\frac{f(x_{i-1}) + f(x_i)}{2}\right) = \frac{h}{2}(f(x_{i-1}) + f(x_i)) \qquad \text{Equation 5}$$

- Thus we obtain: $\int_a^b f(x)dx = \sum_{i=1}^{n} a_i = \frac{h}{2}\sum_{i=1}^{n}[f(x_{i-1}) + f(x_i)]$ Equation 6
- By grouping terms in the sum, for ($h=(x_n-x_0)/2$), we end up with

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + \left[2\sum_{i=1}^{n-1} f(x_i)\right] + f(x_n)\right] \qquad \text{Equation 7}$$



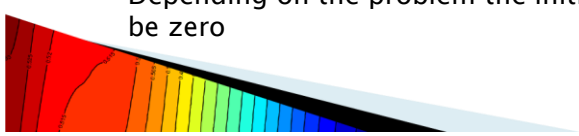
37

Integrating over x

- Equation 7 gives how to compute the definite integral from a to b.
- What if we want to compute the integral, call it $I(x)$, values at it progresses over each value of x_i
- Modify Equation 7 to compute the $I(x_i)$ from the value of $I(x_{i-1})$ (see the last slide for derivation)

$$I(x_i) = I(x_{i-1}) + \frac{h}{2}[f(x_{i-1}) + f(x_i)] \qquad \text{Equation 8}$$

- Where $I(x_0) = 0$ and $I(x_i)$ is compute for $1 < i < n$
- Note that $I(x)$ describes the change in the function from point x_0, that is, $I(x_0) = 0.0$.
  - Depending on the problem the initial desired value may not be zero



38

Trapezoidal Rule for The Parachutist Problem

- Recall the integral for the distance of the parachutist.

$$d(t) = \frac{gm}{c}\int_0^t \left(1 - e^{-(c/m)u}\right) du \qquad \text{Equation 9}$$

  - The distance $d(t_0) = 0$ where $t_0 = 0.0$
- Applying Equation 8 to Equation 9 yields

$$f(t_i) = \left(1 - e^{-(c/m)t_i}\right)$$

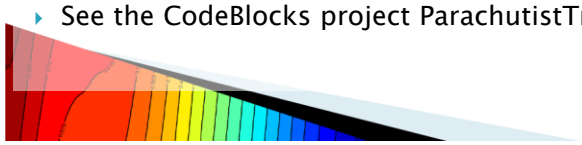
$$I(t_i) = I(t_{i-1}) + \frac{h}{2}\left[f(t_{i-1}) + f(t_i)\right] \qquad \text{Equation 10}$$

$$d(t_i) = \frac{gm}{c}\left[I(t_i)\right]$$

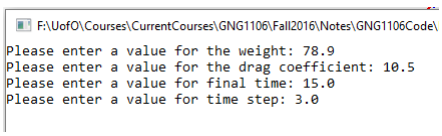
- The analytical solution for $d(t_i)$ is

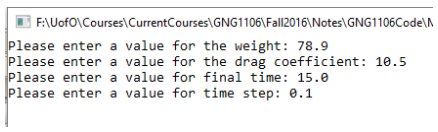
$$d(t_i) = \frac{gm}{c}\left[t_i - \frac{m}{c}\left(1 - e^{-(c/m)t_i}\right)\right] \qquad \text{Equation 11}$$

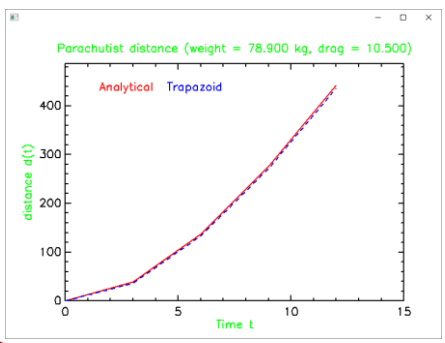
- See the CodeBlocks project ParachutistTrapezoid.

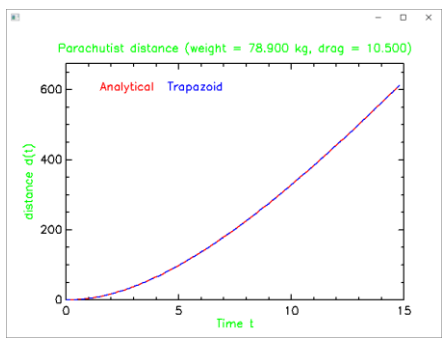


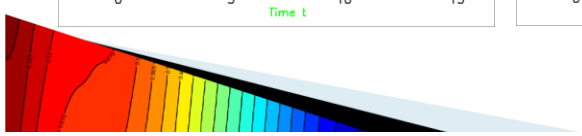
Project ParachutistTrapezoid





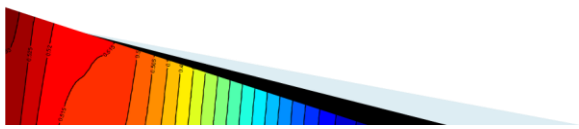






40

Next Module



Derivation for computing $I(x_i)$

$$I(x_{i-1}) = \frac{h}{2}\left[f(x_0) + \left[2\sum_{k=1}^{i-2} f(x_k) \right] + f(x_{i-1}) \right]$$

$$I(x_i) = \frac{h}{2}\left[f(x_0) + \left[2\sum_{k=1}^{i-1} f(x_k) \right] + f(x_i) \right]$$

$$= \frac{h}{2}\left[f(x_0) + \left[2\sum_{k=1}^{i-2} f(x_i) \right] + 2f(x_{i-1}) + f(x_i) \right]$$

$$= \frac{h}{2}\left[f(x_0) + \left[2\sum_{k=1}^{i-2} f(x_i) \right] + f(x_{i-1}) \right] + \frac{h}{2}\left[f(x_{i-1}) + f(x_i) \right]$$

$$= I(x_{i-1}) + \frac{h}{2}\left[f(x_{i-1}) + f(x_i) \right]$$