CSE218-NUMERICAL METHODS

1905072-Mahir Labib Dihan

1) SOLVING NONLINEAR EQUATIONS

a) Bisection:

i) **Theorem:** An equation f(x)=0, where f(x) is a real continuous function, has at least one root between x_l and x_u if $f(x_l)f(x_u) < 0$

ii) **Steps:**

(1) Choose x_l and x_u as two guesses for the root such that $f(x_l)\ f(x_u)\ <\ 0$ [It's better to choose two x values on the same side of y axis where the root is] Note: We need to check $f(x_l)\ f(x_u)\ <\ 0$ only before the first iteration.

(2) Estimate the root, x_m of the equation $f(x) = 0$ as the mid-point between x_l and x_u as, $x_m = \frac{x_l+x_u}{2}$

(3) Update the value of x_l and x_u

- If $f(x_l)f(x_m) < 0$, then the root lies between x_l and x_m; then $x_l = x_l$; $x_u = x_m$
- If $f(x_l)f(x_m) > 0$, then the root lies between x_m and x_u; then $x_l = x_m$; $x_u = x_u$
- If $f(x_l)f(x_m) = 0$, then the root is x_m; Stop the algorithm

(4) Find the new estimation of the root, $x_m = \frac{x_l+x_u}{2}$. Find the absolute relative approximate error, $|\epsilon_a| = \left|\frac{x_m^{new}-x_m^{old}}{x_m^{new}}\right| \times 100$ $[x_m^{new} \neq 0\ ,\ \ if\ x_m^{new} = 0, make\ it\ nonzero\ by\ adding\ epsilon(a\ very\ small\ positive\ number).]$

(5) Compare the absolute relative approximate error with the pre-specified relative error tolerance ϵ_s. Also, check if the number of iterations has exceeded the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user. If not, go to step 3.

iii) **Calculate significant digits are at least correct in the estimate root**

(1) Largest value of m for which $|\epsilon_a| \leq 0.5 \times 10^{2-m}$

(2) $m = \lfloor 2 - \log_{10}(2 \times |\epsilon_a|) \rfloor$

iv) **Pros:**

(1) The bisection method is always convergent. Since the method brackets the root, the method is guaranteed to converge.

(2) As iterations are conducted, the interval gets halved. So, one can guarantee the error in the solution of the equation.

v) **Cons:**

(1) This method will work only when f(x) changes sign. If a function f(x) is such that it just touches the x-axis it will be unable to find the lower and upper guesses. Bisection won't work in that case. We have to go for Newton-Raphson.

- If f(x) changes sign there will be odd (1, 3, 5 ...) number of roots
- If f(x) doesn't change sign there will be 0/1/2/4/6... roots

(2) For discontinuous functions, where there is a singularity and it reverses sign at the singularity, the bisection method may converge on the singularity. Because the function changes sign between two points. But no root exists. Bisection can't detect that.

(3) The convergence of the bisection method is slow as it is simply based on halving the interval.

(4) If one of the initial guesses is close to the root, the convergence is slower

(5) Have to guess two points. And function must change sign between the two points.

b) Newton-Raphson:

i) **Steps:**

(1) Evaluate $f'(x)$ symbolically

(2) Use an initial guess of the root, x_i. [It's better to choose the point on the side of x axis where the root is]

(3) To estimate the new value of the root,$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$ $[f'(x_i) \neq 0]$

(4) Find the absolute relative approximate error, $|\epsilon_a| = \left|\frac{x_{i+1}-x_i}{x_{i+1}}\right| \times 100$ [$x_{i+1} \neq$0, if $x_{i+1} = 0$, make it nonzero by adding epsilon (a very small positive number)]

(5) Compare the absolute relative approximate error with the pre-specified relative error tolerance ϵ_s. Also, check if the number of iterations has exceeded the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user. If not, go to step 3.

ii) **Pros:**

(1) Requires only one guess

(2) Converges fast (Quadratic Convergence)

iii) **Cons:**

(1) Division by zero: if we guess the point, where the slope is 0 (Root can't be found)

(2) Divergence at inflection points. After some more iterations converges to the exact root. So it's slow.

(3) Oscillations near local maximum and minimum (Program won't stop until max iteration limit exceeded)

(4) Root Jumping. Due to root jumping we may not get our expected root, rather we will get some other root. How can we avoid this?

2) SOLVING LINEAR EQUATIONS

Note: After forward elimination, if we have zero values in the main diagonal then we can't find the solution. (No solution / Infinitely many solutions) Explanation: Since product of all the values in the main diagonal of an upper triangular matrix is its determinant. So, if any value is zero, determinant is also zero. "*If the determinant of a matrix is zero, then the linear system of equations it represents has no solution. In other words, the system of equations contains at least two equations that are not linearly independent.*" So how will we know no solution or infinite solution? If right and left side of any equation is zero, then there are infinite solutions. If only left side of the equation is zero and right side is non-zero, then no solution exists.

a) Naïve Gaussian Elimination:

i) **Steps:**

(1) Forward Elimination

- Transform coefficient matrix into upper triangular matrix
- (n-1) steps of forward elimination

(2) Back Substitution

- Solve each equation starting from the last equation

ii) **Cons:**

(1) Division by zero

(2) Large round off error

b) Gaussian Elimination with Partial Pivoting:

i) **Steps:**

(1) Forward Elimination

- Transform coefficient matrix into upper triangular matrix
- (n-1) steps of forward elimination
- At the beginning of the k^{th} step of forward elimination, swap k^{th} row with the row which have maximum absolute value at k^{th} column. (This will avoid division by zero, as long as the maximum absolute value at k^{th} column is not zero itself.)

(2) Back Substitution

- Solve each equation starting from the last equation

c) Finding determinant using Gaussian Elimination:

i) Steps

(1) Forward Elimination
(2) Determinant = Product of all the values in the diagonal

3) <u>INTERPOLATION</u>

Note: Higher order polynomial doesn't guarantee more accurate result.

a) Newton's Divided Difference Polynomial Method

i) Steps

(1) x=The point that needs to be interpolate [Must be in the range. If not in the range, then Extrapolation→Regression]
(2) $Choose\ nearest\ n+1\ points\ of\ x, that\ also\ bracket\ x.$ [n=order of polynomial]

- $While\ choosing\ the\ points\ we\ need\ to\ take\ the\ left\ and\ right\ point\ of\ the\ given\ point. Then\ the\ closest\ n-1\ points.$

(3) $f[x_i] = f(x_i) = y_i$
(4) $b_i = f[x_i, x_{i-1}, \dots, x_0] = \frac{f[x_i, x_{i-1}, \dots, x_1] - f[x_{i-1}, x_{i-2}, \dots, x_0]}{x_i - x_0}$ $\quad [Constant]$
(5) $f_n(x) = b_0 + b_1(x-x_0) + \cdots + b_n(x-x_0)(x-x_1)\dots(x-x_{n-1}) = \sum_{i=0}^{n} b_i \prod_{j=o}^{i-1}(x-x_j)$
(6) $f_n(x) = 0^{th}\ order + 1^{st}\ order + \cdots + n^{th}\ order$
(7) $\prod_{j=o}^{i-1}(x-x_j)\ [i^{th}\ order\ polynomial]$

ii) Pros:

(1) Just a new term is added with the change in degree

iii) Cons:

(1) It's hard to find the constants

b) Lagrange

i) Steps

(1) x=The point that needs to interpolate [Must be in the range. If not in the range, then Extrapolation→Regression]
(2) $Choose\ nearest\ n+1\ points\ of\ x, that\ also\ bracket\ x$

- $While\ choosing\ the\ points\ we\ need\ to\ take\ the\ left\ and\ right\ point\ of\ the\ given\ point. Then\ the\ closest\ n-1\ points.$

(3) $L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$ $\quad [Weighting\ function, n^{th}\ order\ polynomial]$
(4) $f_n(x) = \sum_{i=0}^{n} L_i(x) y_i$
(5) $f_n(x) = n^{th}\ order + n^{th} order + \cdots + n^{th}\ order$

ii) Pros:

(1) Coefficient can be found easily

iii) Cons:

(1) All the terms changes with the change in degree

4) <u>INTEGRATION</u>

Note: More subsegments, less error.

a) Trapezoidal Rule

i) Steps

(1) $Single\ segment, \int_a^b f(x)dx = \frac{(b-a)}{2}[f(a) + f(b)]$
(2) $Multiple\ segment, \int_a^b f(x)dx = \frac{b-a}{2n}\left[f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)\right]$

ii) True error

(1) $Single\ segment, E_t = -\frac{(b-a)^3}{12} f''(\alpha)\ \ [a < \alpha < b]$
(2) $Multiple\ segment, E_t = -\frac{(b-a)^3}{12n^2} \frac{\sum_{i=1}^{n} f''(\alpha_i)}{n}\ \ [a + (i-1)h < \alpha_i < a + ih]$

- $n \to E_t$
- $2n \to \frac{E_t}{2^2}$
- $As\ the\ number\ of\ segments\ are\ doubled, the\ true\ error\ gets\ approximately\ quartered.$

iii) Pros

(1) Can work with both odd and even number of sub segments

iv) Cons

(1) Converges slower

b) Simpsons 1/3rd rule

i) Steps

(1) $Double\ segment, \int_a^b f(x)dx = \frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$
(2) $Multiple\ segment, \int_a^b f(x)dx = \frac{(b-a)}{3n}\left[f(a) + 4\sum_{\substack{i=1 \\ i=odd}}^{n-1} f(a+ih) + 2\sum_{\substack{i=2 \\ i=even}}^{n-2} f(a+ih) + f(b)\right]$

ii) True error

(1) $Double\ segment, E_t = -\frac{(b-a)^5}{2880} f^4(\alpha)\ \ [a < \alpha < b]$
(2) $Multiple\ segment, E_t = -\frac{(b-a)^5}{180n^4} \frac{\sum_{i=1}^{\frac{n}{2}} f^4(\alpha_i)}{\frac{n}{2}} = -\frac{(b-a)^5}{90n^4} \frac{\sum_{i=1}^{\frac{n}{2}} f^4(\alpha_i)}{n} [a + 2(i-1)h < \alpha_i < a + 2ih]$

- $n \to E_t$
- $2n \to \frac{E_t}{2^4}$
- $As\ the\ number\ of\ segments\ are\ doubled, the\ true\ error\ gets\ approximately \frac{1}{2^4}.$

iii) Pros

(1) Converges faster

iv) Cons

(1) Can't work with odd number of sub segments

5) <u>REGRESSION</u>

Note: If specific model is not given, we have to guess the model from the graph of given data.

a) Linear

i) Cases

(1) $y = a_0 + a_1 x$

- $S_r = \sum_{i=1}^{n} E_i^2 = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i)^2$
- $a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$ $\quad or\ a_o = \bar{y} - a_1\bar{x}$

- $a_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$

(2) $y = a_1 x$

- $S_r = \sum_{i=1}^{n} E_i^2 = \sum_{i=1}^{n}(y_i - a_1 x_i)^2$
- $a_1 = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$

b) Non-Linear

i) Exponential Model

(1) Steps

- $y = ae^{bx}$
- $S_r = \sum_{i=1}^{n} E_i^2 = \sum_{i=1}^{n}\left(y_i - ae^{bx}\right)^2$
- $a = \frac{\sum_{i=1}^{n} y_i e^{bx}}{\sum_{i=1}^{n} e^{2bx_i}}$
- $\sum_{i=1}^{n} y_i x_i e^{bx_i} - \frac{\sum_{i=1}^{n} y_i e^{bx}}{\sum_{i=1}^{n} e^{2bx_i}} \sum_{i=1}^{n} x_i e^{2bx_i} = 0$ [Solve this using numerical methods for solving nonlinear equation like **Bisection**]

(2) Converting to Linear

- $y = ae^{bx}$
- $\ln y = \ln a + bx$
- $Y = \ln y, X = x, a_0 = \ln a\ , a_1 = b$
- $Y = a_0 + a_1 X$
- $a = e^{a_0}, b = a_1$
- $y = e^{a_0} e^{a_1 x}$

ii) Polynomial Model

(1) Steps

- $y = a_0 + a_1 x + \cdots + a_m x^m$
- $S_r = \sum_{i=1}^{n} E_i^2 = \sum_{i=1}^{n}(y_i - a_0 - a_1 x_i - \cdots - a_m x_i^m)^2$
- $\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \cdots & \sum_{i=1}^{n} x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^m y_i \end{bmatrix}$
- Find $a_0, a_1, \dots, a_m$ using numerical method of solving linear equation like **Gaussian Elimination.**

iii) Saturation Growth Model

(1) Convert to Linear

- $y = \frac{ax}{b+x}$
- $\frac{1}{y} = \frac{1}{a} + \left(\frac{b}{a}\right) x$
- $Y = \frac{1}{y}\ , X = x\ , a_0 = \frac{1}{a}\ , a_1 = \frac{b}{a}$
- $Y = a_0 + a_1 X$
- $a = \frac{1}{a_0}, b = \frac{a_1}{a_0} = a_1 * a$
- $y = \frac{\frac{x}{a_0}}{\frac{a_1}{a_0} + x}$

iv) Power Model

(1) Convert to Linear

- $y = ax^b$
- $\ln y = \ln a + b \ln x$
- $Y = \ln y\ , X = \ln x\ , a_0 = \ln a\ , a_1 = b$
- $Y = a_0 + a_1 X$
- $a = e^{a_0}\ , b = a_1$
- $y = e^{a_0} x^{a_1}$

6) <u>DIFFERENTIATION</u>

a) First Derivative

Note: Δx is positive. The smaller this is, the more accurate differentiation.

i) Forward Difference Approximation

(1) $f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$

ii) Backward Difference Approximation

(1) $f'(x) \approx \frac{f(x) - f(x-\Delta x)}{\Delta x}$

7) <u>Absolute Relative Approximate Error</u>

a) $|\epsilon_a| = \left|\frac{newValue - oldValue}{newValue}\right| \times 100$