MTH 408: Numerical Analysis
Semester 1, 2019-20

November 30, 2019

Contents

These notes have been prepared directly based on the material in [1]. All algorithms have been implemented with appropriate codes written for Mathematica 12 [2].

1 Solutions of equations in one variable

1.1 Review of results from single-variable calculus

Theorem 1.1.1. *Let $f : X(\subset \mathbb{R}) \to \mathbb{R}$, and let $x_0 \in X$. Then the following statements are equivalent.*

(a) f is continuous at x_0.

(b) If $\{x_n\}$ is a sequence in X such that $x_n \to x$, then $f(x_n) \to f(x)$.

Theorem 1.1.2. *Let $f : X(\subset \mathbb{R}) \to \mathbb{R}$. If f is continuous at $x_0 \in X$, then f is continuous at x_0.*

Theorem 1.1.3 (Rolles' Theorem)**.** *Suppose that $f \in C[a, b]$ and f is differentiable in (a, b). Then there exists $c \in (a, b)$ such that $f'(c) = 0$.*

Theorem 1.1.4 (Mean Value Theorem)**.** *Suppose that $f \in C[a, b]$ and f is differentiable in (a, b). Then there exists $c \in (a, b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

Theorem 1.1.5 (Extreme Value Theorem)**.** *If $f \in C[a, b]$, then there exists $c_1, c_2 \in [a, b]$ such that*

$$f(c_1) \leq f(x) \leq f(c_2), \, \forall x \in [a, b].$$

Theorem 1.1.6 (Intermediate Value Theorem)**.** *If $f \in C[a, b]$ and K is any number between $f(a)$ and $f(b)$, then there exists $c \in (a, b)$ such that $f(c) = K$.*

Theorem 1.1.7 (Mean Value Theorem for Integrals)**.** *Suppose that $f \in C[a, b]$, and g is Riemann Integrable function on $[a, b]$ which does not change sign in $[a, b]$. Then there exists $c \in (a, b)$ such that*

$$\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx.$$

Theorem 1.1.8 (Taylor's Theorem)**.** *Suppose that $f \in C^n[a,b]$ such that $f^{(n+1)}$ exists exists on $[a,b]$, and let $x_0 \in [a,b]$. Then for every $x \in [a,b]$, there exists $\xi(x)$ between x_0 and x with*

$$f(x) = P_n(x) + R_n(x),$$

where

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k \text{ and } R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1}.$$

Here, $P_n(x)$ is called the n^{th} Taylor polynomial of f about x_0 and $R_n(x)$ is called the remainder term associated with $P_n(x)$.

1.2 Bisection method

Let $f \in C[a,b]$ such that f has a root in $[a,b]$.

Purpose.

Finding the root p (or solution) of an equation of the form $f(x) = 0$, for a given function f.

Concept.

We iteratively bisect the given interval $[a,b]$ into subintervals and apply the Intermediate value theorem to determine the subinterval in which root exists, and approximate the root at each iteration by the midpoint of this subinterval. This generates a sequence $\{p_n\}$ such that $p_n \to p$.

Method.

- Set $a_1 = a$, $b_1 = b$, and let $p_1 = (a_1 + b_1)/2$.
- If $f(p_1) = 0$, then $p = p_1$, and we are done.
- If $f(p_1) \neq 0$, then $f(p_1)$ has the same sign as either $f(a_1)$ or $f(b_1)$.
    - If $f(p_1)$ and $f(a_1)$ have the same sign, then $p \in (p_1, b_1)$. Set $a_2 = p_1$ and $b_2 = b_1$.
    - If $f(p_1)$ and $f(a_1)$ have opposite signs, then $p \in (a_1, p_1)$. Set $a_2 = a_1$ and $b_2 = p_1$.

Algorithm

INPUT endpoints a, b; tolerance TOL; maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 1$; $FA = f(a)$.
Step 2 While i $\leq$ N do Steps $3 - 6$.
Step 3 Set $p = a + (b - a)/2$; $FP = f(p)$.
Step 4 If $FP = 0$ or $(b - a)/2 < TOL$ then
OUTPUT p;
STOP.
Step 5 Set $i = i + 1$.
Step 6 If $FA \cdot FP > 0$ then set $a = p$; $FA = FP$
else set $b = p$.
Step 7 OUTPUT ("Method failed after N_0 iterations", N_0);
STOP.

Code with output

```
Bisection[ACC_, Iter_] := Module[{i = 1, FA, FP, p, a = 0, b = 4},
FA = Func[a];
While[i <= Iter,
    p = (a + b)/2.0;
    FP = Func[p];
    If[FP == 0 || (b - a)/2.0 <= N[(10)^(-ACC - 1)],
        Print["Root up to the desired accuracy is ",
        SetAccuracy[p, ACC + 1]]; Break[],
        Print["The approximation after iteration ", i, " is ", p];
        i = i + 1;
        If[FA*FP > 0, a = p; FA = FP, b = p];
    ];
];
If[i > Iter, Print["Method Failed after ", i - 1, " iterations"]];
]

Func[x_] := x^2 - 3

Bisection[4, 30]

The approximation after iteration 1 is 2.
```

```
The approximation after iteration 2 is 1.
The approximation after iteration 3 is 1.5
The approximation after iteration 4 is 1.75
The approximation after iteration 5 is 1.625
The approximation after iteration 6 is 1.6875
The approximation after iteration 7 is 1.71875
The approximation after iteration 8 is 1.73438
The approximation after iteration 9 is 1.72656
The approximation after iteration 10 is 1.73047
The approximation after iteration 11 is 1.73242
The approximation after iteration 12 is 1.73145
The approximation after iteration 13 is 1.73193
The approximation after iteration 14 is 1.73218
The approximation after iteration 15 is 1.73206
The approximation after iteration 16 is 1.73199
The approximation after iteration 17 is 1.73203
The approximation after iteration 18 is 1.73204

Root up to the desired accuracy is 1.7320
```

Convergence

Theorem 1.2.1 (Convergence of Bisection method)**.** *Suppose that $f \in C[a,b]$ and $f(a)f(b) < 0$. Then the Bisection method generates a sequence $\{p_n\}$ approximating a zero p of f with*

$$|p_n - p| < \frac{b-a}{2^n}, \text{ for } n \geq 1.$$

1.3 Fixed-Point Iteration Method

Some preliminaries

Definition 1.3.1. A fixed point of a given function $f : X(\subset \mathbb{R}) \to \mathbb{R}$ is a point $p \in X$ such that $f(p) = p$.

Remark 1.3.2. A function $f : X(\subset \mathbb{R}) \to \mathbb{R}$ has a fixed point $p \in X$ if and only if the function $g(x) = f(x) - x$ has a zero at p.

Theorem 1.3.3. *Suppose that $g : [a,b] \to [a,b]$ is a continuous function.*

(i) Then g has at least one fixed point in $[a, b]$.

(ii) If in addition we assume that g' exists on (a, b) and there exists a positive constant $k < 1$ such that $|g'(x)| \leq k$, for all $x \in (a, b)$, then g has exactly one fixed point in $[a, b]$.

Purpose

Approximating the fixed point p of a function $g : [a, b] \to [a, b]$ that satisfies the hypotheses of Theorem 1.3.3.

Concept

This method is a direct application of Theorems 1.1.1 and 1.3.3.

Method

We start with an initial approximation p_0 and generate a sequence $\{p_n\}$ of successive approximations by taking $p_n = g(p_{n-1})$, for $n \geq 1$. If $p_n \to p$, then by Theorem 1.1.1, we have

$$p = \lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) = g(\lim_{n\to\infty} p_{n-1}) = g(p).$$

Algorithm

INPUT initial approx. p_0; tolerance TOL; maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 1$.
Step 2 While $i \leq N_0$ do Steps $3 - 6$.
Step 3 Set $p = g(p_0)$.
Step 4 If $|p - p_0| < TOL$ then
        OUTPUT p;
        STOP.
Step 5 Set $i = i + 1$.
Step 6 Set $p_0 = p$.
Step 7 OUTPUT ("The method failed after N_0 iterations", N_0);
STOP.

Code with output

```
(*Courtesy of Srabana Biswas and Divyasree C R*)
FPI[ACC_, Iter_] := Module[{i = 1, p, p0 = (a + b)/2.0},
While[i <= Iter,
    p = Func[p0];
    If[Abs[p - p0] <= N[(10)^(-ACC - 1)],
        Print["Fixed point up to the desired accuracy is ",
        SetAccuracy[p, ACC + 1]]; Break[],
        Print["The approx value after iteration ", i, " is ", p];
        i = i + 1; p0 = p
     ];
];
If[i > Iter, Print["Method Failed after ", i - 1, " iterations"]];
]

Func[x_] := 6^(-x); a = 0; b = 1;

FPI[2, 50]

The approx value after iteration 1 is 0.408248
The approx value after iteration 2 is 0.481195
The approx value after iteration 3 is 0.422238
The approx value after iteration 4 is 0.469283
The approx value after iteration 5 is 0.431347
The approx value after iteration 6 is 0.461686
The approx value after iteration 7 is 0.437259
The approx value after iteration 8 is 0.456822
The approx value after iteration 9 is 0.441086
The approx value after iteration 10 is 0.453699
The approx value after iteration 11 is 0.443561
The approx value after iteration 12 is 0.451692
The approx value after iteration 13 is 0.445159
The approx value after iteration 14 is 0.450401
The approx value after iteration 15 is 0.44619
The approx value after iteration 16 is 0.449569
The approx value after iteration 17 is 0.446856
The approx value after iteration 18 is 0.449033
```

```
The approx value after iteration 19 is 0.447285
The approx value after iteration 20 is 0.448688
The approx value after iteration 21 is 0.447561

Fixed point up to the desired accuracy is 0.45
```

Convergence

Theorem 1.3.4. *Suppose that $g : [a, b] \to [a, b]$ is a continuous function. If g' exists on (a, b) and there exists a positive constant $k < 1$ such that $|g'(x)| \leq k$, for all $x \in (a, b)$, then for any $p_0 \in [a, b]$, the sequence $\{p_n\}$ defined by $p_n = g(p_{n-1})$, for $n \leq 1$, converges to a unique fixed point $p \in [a, b]$.*

Corollary 1.3.5. *If a function g satisfies the hypothesis of Theorem 1.3.4, then the bounds for the error involved in using the p_n to approximate p are given by*

$$|p_n - p| \leq k^n \max\{p_0 - a, b - p_0\}$$

and

$$p_n - p| \leq \frac{k^n}{1 - k}|p_1 - p_0|, \text{ for } n \geq 1.$$

Remark 1.3.6. Corollary 1.3.5 implies that the rate of convergence of the sequence $\{p_n\}$ is dependent on bound on the first derivative (i.e k). More precisely, smaller k would mean faster convergence of $\{p_n\}$.

So the trick is to manipulate the root-finding problem into fixed-point problem $g(x) = x$ that satisfies the hypothesis of Theorem 1.3.4 so that g' is small.

1.4 Newton-Raphson Method

Purpose

Approximating the root of a given function $f \in C^2[a, b]$.

Concept

We start with an initial approximation p_0 such that $f'(p_0) \neq 0$ and use the 2^{nd} Taylor polynomial (as in Theorem 1.1.8) to find successive approximations.

Method

Starting with the initial approximation p_0, we obtain a sequence $\{p_n\}$ of successive approximations by using the iterative formula

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \text{ for } n \geq 1.$$

Algorithm

INPUT initial approximation p_0; tolerance TOL;
maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 1$.
Step 2 While $i \leq N_0$ do Steps $3 - 6$.
Step 3 Set $p = p_0 - f(p_0)/f'(p_0)$.
Step 4 If $|p - p_0| < TOL$ then
OUTPUT (p);
STOP.
Step 5 Set $i = i + 1$.
Step 6 Set $p_0 = p$.
Step 7 OUTPUT ("The method failed after N_0 iterations", N_0);
STOP

Code with output

```
(*Courtesy of Koustav Mondal*)
NewtonRaphson[acc_, Iter_] := Module[{i = 1, p0 = (a + b)/2.0, p1},
While[i <= Iter,
    p1 = p0 - (Func[p0]/Func'[p0]);
    If[Func[p1] == 0 || Abs[p1 - p0] <= N[(10)^(-acc - 1)],
        Print["Root upto desired accuracy is ",
        SetAccuracy[p1, acc + 1]]; Break[],
        Print["The approximation after iteration ", i, " is ", p1];
        i = i + 1; p0 = p1
      ];
];
If[i > Iter, Print["method failed after ", i - 1, " iterations"]];
]
```

```
Func[x_] := Sin[x] - Exp[-x]; a = 3; b = 5;

NewtonRaphson[4, 50]

The approximation after iteration 1 is 2.77997
The approximation after iteration 2 is 3.11406
The approximation after iteration 3 is 3.09638
The approximation after iteration 4 is 3.09636

Root upto desired accuracy is 3.0964
```

Convergence

Theorem 1.4.1. *Let $f \in C^2[a,b]$. If $p \in (a,b)$ such that $f(p) = 0$ and $f'(p) \neq 0$, then there exists $\delta > 0$ such that the Newton-Raphson method generates a sequence $\{p_n\}$ converging to p, for any initially chosen approximation $p_0 \in [p_0 - \delta, p_0 + \delta]$.*

1.5 Secant Method

Purpose

Approximating the root of a given function $f \in C[a,b]$.

Concept

One of the drawbacks of the Newton-Raphson method is that it requires the value of the derivative to be computed at each iteration. The Secant Method uses the divided-difference formula for the derivative (of a function) to overcome this drawback.

Method

Starting with the initial approximation p_0, we obtain a sequence $\{p_n\}$ of successive approximations by using the iterative formula

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}, \text{ for } n \geq 1.$$

Algorithm

INPUT initial approximations p_0, p_1; tolerance TOL;
maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 2$; $q_0 = f(p_0)$; $q_1 = f(p_1)$.
Step 2 While $i \leq N_0$ do Steps $3 - 6$.
Step 3 Set $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$.
Step 4 If $|p - p_1| < TOL$ then
OUTPUT (p);
STOP.
Step 5 Set $i = i + 1$.
Step 6 Set $p_0 = p_1$; $q_0 = q_1$; $p_1 = p$; $q_1 = f(p)$.
Step 7 OUTPUT ("The method failed after N_0 iterations", N_0);
STOP.

Code with output

```
(*Courtesy of R. Aswin*)
Secant[ACC_, Iter_] := Module[{i = 1, FA, FB, x, y, k, FX, FY, p, FP},
  FA = Func[a]; FB = Func[b];
  While[i <= Iter,
     If[i == 1, x = a; y = b; FX = FA; FY = FB];
        If[FY != FX, k = ((y - x)/(FY - FX))*1.0;
           p = y - FY*k; FP = Func[p];
           If[FP == 0 || Abs[(p - y)/2.0] <= N[10^(-ACC - 1)],
             Print["The root with desired accuracy is ",SetAccuracy[p, ACC + 1]];
             Break[],
             Print["The approximated root after ", i, "th iteration is " , p];
             i = i + 1;
             x = y;
             FX = FY;
             y = p;
             FY = FP;
           ]
       ]
   ];
  If[i > Iter, Print["The method failed after ", i, "iterations"]];
]
```

```
Func[x_] := x^2 - 5; a = 2; b = 3;

Secant[4, 40]

The approximated root after 1th iteration is 2.2
The approximated root after 2th iteration is 2.23077
The approximated root after 3th iteration is 2.23611
The approximated root after 4th iteration is 2.23607

The root with desired accuracy is 2.2361
```

1.6 Regula Falsi Method

Purpose

Approximating the root of a given function $f \in C[a, b]$.

Concept

This is a variant of the Secant method in which approximations are generated in the same manner, but it includes a test at each iteration to ensure that the root always lies between two successive approximations.

Method

Choose initial approximations p_0 and p_1 so that $f(p_0)f(p_1) < 0$.

- If $f(p_2)f(p_1) < 0$, then a root lies between p_1 and p_2. Choose p_3 as the x-intercept joining $(p_1, f(p_1))$ and $(p_2, f(p_2))$.
- If not, choose p_3 as the x-intercept of the line joining $(p_0, f(p_0))$ and $(p_2, f(p_2))$, and then interchange the indices on p_0 and p_1.

Algorithm

INPUT initial approximations p_0, p_1; tolerance TOL; maximum N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 2$; $q_0 = f(p_0)$; $q_1 = f(p_1)$.

Step 2 While $i \leq N_0$ do Steps $3-7$.
Step 3 Set $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$.
Step 4 If $|p - p_1| < TOL$ then
        OUTPUT (p);
        STOP.
Step 5 Set $i = i + 1$; $q = f(p)$
Step 6 If $qq_1 < 0$ then set $p_0 = p_1$; $q_0 = q_1$.
Step 7 Set $p_1 = p$; $q_1 = q$.
Step 8 OUTPUT ("Method failed after N_0 iterations", N0);
STOP.

Code with output

```
(*Courtesy of Amanjit Sarma*)
RegulaFalsi[ACC_, Iter_] := Module[{i = 1, p, p0 = P0, p1 = P1},
While[i <= Iter,
       p = p1 - (Func[p1]*(p1 - p0))/(Func[p1] - Func[p0]);
       If[Func[p] == 0 || Abs[p - p1] <= N[(10)^(-ACC - 1)],
           Print["Root up to the desired accuracy is ",
           SetAccuracy[p, ACC + 1]]; Break[],
           Print["The approximation after iteration ", i, " is ", p];
           i = i + 1;
           If[Func[p]*Func[p1] < 0, p0 = p1; Func[p0] = Func[p1]];
                   p1 = p;
                   Func[p1] = Func[p];
              ];
       ];
If[i > Iter, Print["Method Failed after ", i - 1, " iterations"]];
]

Func[x_] := Log[x - 1] + Cos[x - 1]; P0 = 1.3; P1 = 2.0;

RegulaFalsi[5, 30]

The approximation after iteration 1 is 1.52061
The approximation after iteration 2 is 1.41837
The approximation after iteration 3 is 1.40114
The approximation after iteration 4 is 1.3983
The approximation after iteration 5 is 1.39784
```

```
The approximation after iteration 6 is 1.39776
The approximation after iteration 7 is 1.39775
The approximation after iteration 8 is 1.39775
Root up to the desired accuracy is 1.39775
```

1.7 Order of convergence

Definition 1.7.1. Suppose $\{p_n\}$ is a sequence of reals such that $p_n \to p$ with $p_n \neq p$, for all n. If positive constants λ and α exist with

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|^\alpha} = \lambda,$$

then the convergence of the sequence $\{p_n\}$ is said to be of *order* α, with *asymptotic error* λ. In particular:

(a) If $\alpha = 1$ and $\lambda < 1$, then $\{p_n\}$ is said to *linearly* convergent.

(b) If $\alpha = 2$, then $\{p_n\}$ is said to *quadratically* convergent.

Example 1.7.2. The sequence $p_n = (0.5)^n$ is linearly convergent, while the sequence $(0.5)^{2^n-1}$ is quadratically convergent.

Theorem 1.7.3 (Order of convergence of FPI method)**.** *Let $g : [a, b] \to [a, b]$ be a continuous function. Suppose g' is continuous on (a, b), and a positive constant $k < 1$ exists with $|g'(x)| \leq k$, for all $x \in (a, b)$. If $g'(p) \neq 0$, then for any $p_o \neq p$, the sequence $\{p_n\}$ defined by $p_n = g(p_{n-1}$, for $n \geq 1$, converges linearly to a unique fixed point.*

Theorem 1.7.4. *Let p be a solution to the fixed point problem $x = g(x)$. Suppose $g'(p) = 0$ and g'' is continuous with $|g'(x)| < M$ on an open interval $I \ni p$. Then there exists a $\delta > 0$ such that for $p_0 \in [p-\delta, p+\delta]$, the sequence $\{p_n\}$ defined by $p_n = g(p_{n-1})$, for $n \geq 1$, converges at least quadratically to p. Moreover, for sufficiently large n,*

$$|p_{n+1} - p| < \frac{M}{2}|p_n - p|^2.$$

Theorem 1.7.5. *The Newton-Raphson method converges at least quadratically.*

1.8 Aitken's Δ^2 method

Purpose

Given a sequence $\{p_n\}$ of approximations converging linearly to p, this method constructs a new sequence that converges more rapidly to p.

Concept and method

We assume that $p_n - p$, $p_{n+1} - p$, and $p_{n+2} - p$ have the same sign, and for n sufficiently large

$$\frac{p_{n+1} - p}{p_n - p} \approx \frac{p_{n+2} - p}{p_{n+1} - p}.$$

We define a new sequence $\{\tilde{p}_n\}$ given by

$$\tilde{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+1} - 2p_{n+1} + p_n}.$$

If $\Delta p_n = p_{n+1} - p_n$ and $\Delta^2 p_n = \Delta(\Delta p_n)$, for $n \geq 2$, then

$$\tilde{p}_n = p_n - \frac{(\Delta p_n)^2}{\Delta^2 p_n}. \tag{1}$$

Convergence

Theorem 1.8.1. *Given a sequence $\{p_n\}$ of approximations converges linearly to p and $\lim_{n\to\infty} \frac{p_{n+1} - p}{p_n - p} < 1$, the Aitken's Δ^2 sequence $\tilde{p}_n$ converges faster to p in sense that*

$$\lim_{n\to\infty} \frac{\tilde{p}_n - p}{p_n - p} = 0.$$

1.9 Steffenson's method

Purpose

Applying the Δ^2 to a linearly converging sequence of approximations obtained by the fixed-point iteration method to obtain a new sequence that converges quadratically.

Concept and method

Let $\{p_n\}$ be a linear converging sequence of approximation obtained from the fixed-point iteration method. The Steffenson's method constructs the same first four terms as the Δ^2 method, namely:

$$p_0, p_1, p_2, \text{ and } \tilde{p}_0.$$

Now we assume that $\tilde{p}_0$ is a better approximation to p than p_2. So we apply the fixed-point iteration method to $\tilde{p}_0$ instead of p_2 to obtain the modified sequence:

$$\begin{array}{lll} p_0^{(0)} = p_0, & p_1^{(0)} = g(p_0^{(0)}) = p_1, & p_2^{(0)} = g(p_1^{(0)}) = p_2, \\ p_0^{(1)} = \{\Delta^2\}(p_0^{(0)}), & p_1^{(1)} = g(p_0^{(1)}), & p_2^{(1)} = g(p_1^{(1)}), \\ \vdots & \vdots & \vdots \end{array}$$

Here, $\{\Delta^2\}$ indicates that Equation (1) has been used. In other words, every third term in sequence is generated using Equation 1, while the other terms are generated using the fixed-point iteration method.

Algorithm

INPUT initial approximation p_0; tolerance TOL;
maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $i = 1$.
Step 2 While $i \leq N_0$ do Steps $3 - 6$.
Step 3 Set $p_1 = g(p_0)$; $p_2 = g(p_1)$;
$p = p_0 - (p_1 - p_0)^2/(p_2 - 2p_1 + p_0)$.
Step 4 If $|p - p_0| < TOL$ then
OUTPUT (p);
STOP.
Step 5 Set $i = i + 1$.
Step 6 Set $p_0 = p$.
Step 7 OUTPUT ("Method failed after N_0 iterations", N_0);
STOP.

Convergence

Theorem 1.9.1. *Suppose that $x = g(x)$ has the solution p with $g(p) = 1$. If there exists a $\delta > 0$ such that $g \in C^3[p - \delta, p + \delta]$, then the Steffensen's method gives quadratic convergence for any $p_0 \in [p - \delta, p + \delta]$.*

1.10 Horner's method

Background

Definition 1.10.1. A *polynomial of degree n over* $\mathbb{C}$ is a function $P : \mathbb{C} \to \mathbb{C}$ that has the form

$$P(x) = \sum_{i=0}^{n} a_{n-i} x^{n-i},$$

where the *coefficients* $a_i \in \mathbb{C}$ and $a_n \neq 0$.

Theorem 1.10.2 (Fundamental theorem of algebra)**.** *If $P(x)$ is a polynomial of degree $n \geq 1$ over $\mathbb{C}$, then $P(x) = 0$ has at least one root in $\mathbb{C}$.*

Corollary 1.10.3. *If $P(x)$ is a polynomial of degree $n \geq 1$ over $\mathbb{C}$, then there exist unique constants $x_1, \ldots, x_k \in \mathbb{C}$ and unique integers $m_1, \ldots, m_k$ such that $\sum_{i=1}^{k} m_i = n$ and*

$$P(x) = a_n \prod_{i=1}^{k} (x - x_i)_i^m.$$

Corollary 1.10.4. *Let $P(x)$ and $Q(x)$ be polynomials of degree $\leq n$. If $x_1, \ldots, x_k$, where $k > n$, are distinct numbers with $P(x_i) = Q(x_i)$, for $1 \leq i \leq k$, then $P(x) = Q(x)$, for all x.*

Purpose

Let $P(x) = \sum_{i=0}^{n} a_{n-i} x^{n-i}$ be a polynomial over reals. This method uses the usual synthetic division to evaluate the polynomial and its derivatives at a given $x_0 \in \mathbb{R}$.

Concept and method

Theorem 1.10.5. *Let* $P(x) = \sum_{i=0}^{n} a_{n-i}x^{n-i}$ *be a polynomial over reals. Define* $a_n := b_n$ *and*

$$b_k := a_k + b_{k+1}x_0, \text{ for } k = n-1, n_2, \ldots, 1, 0.$$

Then $b_0 = P(x_0)$. *Moreover, if*

$$Q(x) = \sum_{i=0}^{n-1} b_{n-i}x^{n-i-1},$$

then

$$P(x) = (x - x_0)Q(x) + b_0.$$

Remark 1.10.6. Since $P'(x) = Q(x)$, in the Newton-Raphson method, $P(x)$ and $P'(x)$ can be evaluated similarly.

Example 1.10.7. Let $P(x) = 2x^4 - 3x^2 + 3x - 4$. We wish to evaluate P and P' at $x_0 = -2$, and the next term x_1 of the Newton-Raphson sequence. By Horner's method, we have

$$\begin{array}{r|ccccc} & 2 & 0 & -3 & 3 & -4 \\ -2 & & -4 & 8 & -10 & 14 \\ \hline & 2 & -4 & 5 & -7 & 10 \end{array}.$$

Thus, we have that

$$P(x) = (x+2)(2x^3 - 4x^2 + 5x - 7) \text{ and } P(-2) = 10.$$

Moreover, by Theorem 1.10.5 and Remark 1.10.6, we know that $P'(-2) = Q(-2)$. So, by applying the method once again, we obtain

$$\begin{array}{r|cccc} & 2 & -4 & 5 & -7 \\ -2 & & -4 & 16 & -42 \\ \hline & 2 & -8 & 21 & -49 \end{array},$$

from which it follows that

$$P'(-2) = Q(-2) = -49.$$

Hence, we have that

$$x_1 = x_0 - \frac{P(x_0)}{P'(x_0)} \approx 1.796.$$

Algorithm

INPUT degree n; coefficients $a_0, a_1, \ldots, a_n$ of $P(x)$; x_0.
OUTPUT $y = P(x_0)$; $z = P'(x_0)$
Step 1 Set $y = a_n$; $z = a_n$.
Step 2 For $j = n-1, n-2, \ldots, 1$
Set $y = x_0 y + a_j$;
Set $z = x_0 z + y$;
Step 3 Set $y = x_0 y + a_0$.
Step 4 OUTPUT(y,z);
STOP.

1.11 Müller's method

Background

Theorem 1.11.1. *Let $P(x) = \sum_{i=0}^{n} a_{n-i} x^{n-i}$ be a polynomial over reals. If $z = a + bi$ is a complex root of $P(x)$ of multiciplicity m, then $\bar{z} = a - bi$ is also a zero of of $P(x)$ of multiciplicity m, and*

$$(x^2 - 2ax + a^2 + b^2)^m \mid P(x).$$

Purpose

Let $f(x)$ be a polynomial over reals. This method is used for approximating a roots of the equation $f(x) = 0$, particularly the complex roots.

Concept

The Müller's method uses three initial approximations p_0, p_1, and p_2 and then determines the next term p_3 (of the sequence of approximations) by considering the intersection of x-axis with the parabola through $(p_0, f(p_0))$, $(p_1, f(p_1))$, and $(p_2, f(p_2))$.

Method

The method begins by considering the quadratic polynomial

$$P(x) = a(x - p_2)^2 + b(x - p_2) + c,$$

where a, b, c are determined by the system of equations:

$$\begin{aligned} f(p_0) &= a(p_0 - p_2)^2 + b(p_0 - p_2) + c \\ f(p_1) &= a(p_1 - p_2)^2 + b(p_1 - p_2) + c \\ f(p_2) &= c \end{aligned}$$

Solving for b and a, we have

$$\begin{aligned} b &= \frac{(p_0 - p_2)^2[f(p_1) - f(p_2)] - (p_1 - p_2)^2[f(p_0) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)} \\ a &= \frac{(p_1 - p_2)[f(p_0) - f(p_2)] - (p_0 - p_2)[f(p_1) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)} \end{aligned}$$

Supplying the values of a, b, and c in the quadratic formula $P(x) = 0$, we obtain

$$p_3 = p_2 - \frac{2c}{b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}}.$$

Now p_4 is determined by reinitializing the procedure using p_1, p_2 and p_3.

Remark 1.11.2. If the initial approximations are all real, then successive approximation are also real. So, in order to approximate complex roots, one must ensure that at least one of the initial approximations is a complex number.

Algorithm

INPUT p_0, p_1, p_2; tolerance TOL; maximum iterations N_0.
OUTPUT approximate solution p or message of failure.
Step 1 Set $h_1 = p_1 - p_0$; $h_2 = p_2 - p_1$;
$\delta_1 = (f(p_1) - f(p_0))/h_1$; $\delta_2 = (f(p_2) - f(p_1))/h_2$;
$d = (\delta_2 - \delta_1)/(h_2 + h_1)$; $i = 3$.
Step 2 While $i \leq N_0$ do Steps 3–7.
Step 3 $b = \delta_2 + h_2 d$; $D = (b_2 - 4f(p_2)d)^{1/2}$.
Step 4 If $|b - D| < |b + D|$ then set $E = b + D$
else set $E = b - D$.
Step 5 Set $h = -2f(p_2)/E$; $p = p_2 + h$.
Step 6 If $|h| < TOL$ then
OUTPUT(p);

STOP.

Step 7 Set $p_0 = p_1$; $p_1 = p_2$; $p_2 = p$;
$h_1 = p_1 - p_0$; $h_2 = p_2 - p_1$;
$\delta_1 = (f(p_1) - f(p_0))/h_1$; $\delta_2 = (f(p_2) - f(p_1))/h_2$;
$d = (\delta_2 - \delta_1)/(h_2 + h_1)$; $i = i + 1$.

Step 8 OUTPUT("Method failed after N_0 iterations", N_0);
STOP.

2 Interpolation

2.1 Lagrange's interpolating polynomial

Background

Theorem 2.1.1 (Weierstrass approximation theorem)**.** *Suppose that $f \in C[a, b]$. Then for each $\epsilon > 0$, there exists a polynomial P such that*

$$|f(x) - P(x)| < \epsilon, \text{ for all } x \in [a, b].$$

Example 2.1.2. The function $f(x) = e^x$ can be approximated by the Taylor polynomials

$$P_n(x) = \sum_{k=1}^{n} \frac{x^k}{k!}$$

about $x_0 = 0$.

Remark 2.1.3. The main drawback of using the Taylor polynomials to approximate continuous functions is that the information used in approximating is concentrated around a single point x_0. Consequently, these are inaccurate approximations.

Purpose

Given a continuous function f whose values at $n+1$ distinct numbers $x_0, \ldots, x_n$ are known, this method constructs a polynomial of degree n that approximates f and passes through $(x_0, f(x_0)), \ldots (x_n, f(x_n))$.

Concept and method

Definition 2.1.4. Suppose that f is a continuous function whose values at $n+1$ distinct numbers $x_0, \ldots, x_n$ are given. Then we define the n^{th} *Lagrange*

interpolating polynomial through $(x_0, f(x_0)), \ldots (x_n, f(x_n))$ by

$$P_n(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x),$$

where

$$L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}.$$

Theorem 2.1.5. *Suppose that f is a continuous function whose values at $n+1$ distinct numbers $x_0, \ldots, x_n$ are given. Then the n^{th} Lagrange interpolating polynomial $P_n(x)$ through the points $(x_i, f(x_i))$ is the unique polynomial of degree at most n such that $f(x_k) = P(x_k)$, for $0 \leq k \leq n$.*

Code with output

```
LagIntPoly[Func_[y_], a_, b_, n_] :=
Module[{k, inputs, values, i, polycoeffs = {}, poly, LPoly},
inputs = RandomReal[{a, b}, {n + 1}];
values = Func[inputs];
Do[
        poly = 1;
        For[i = 1, i <= n + 1, i++,
                If[i != k,
                    poly = poly*((x - inputs[[i]])/(inputs[[k]] - inputs[[i]]))
                   ];
            ];
        AppendTo[polycoeffs, poly];
, {k, Range[1, n + 1]}];
LPoly = Sum[values[[i]]*polycoeffs[[i]], {i, 1, n + 1}];
Return[LPoly];
]

In[6]:= Simplify[LagIntPoly[Sin[y], 0, 2*Pi, 4]]

Out[6]= -8.76133 + 11.2767 x - 4.58442 x^2 + 0.718955 x^3 - 0.0381436 x^4
```

Convergence and error

Theorem 2.1.6 (Generalized Rolles theorem)**.** *Suppose that $f \in C[a,b]$ is n times differentiable on (a,b). If $f(x) = 0$ at $n+1$ distinct points $a = x_0 < x_1 < \ldots < x_n \leq b$, then there exists $c \in (x_0, x_n) \subset (a,b)$ such that $f^{(n)}(c) = 0$.*

Theorem 2.1.7. *Suppose that $x_0, \ldots x_n$ are distinct numbers in the interval $[a,b]$ and $f \in C^{n+1}[a,b]$. Then for each $x \in [a,b]$, there exists a number $\xi(x) \in (a,b)$ such that*

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)\ldots(x - x_n),$$

where $P(x)$ is the n^{th} Lagrange interpolating polynomial through the points $(x_i, f(x_i))$.

2.2 Newton divided difference polynomial

Purpose

Let $P_n(x)$ be the n^{th} Lagrange interpolating polynomial of f whose values at $n+1$ distinct numbers (or *nodes*) x_i are given. This method uses the divided differences of f to express P_n in the form

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)\ldots(x - x_{n-1}).$$

Method

Definition 2.2.1. Suppose that f is a continuous function whose values at $n+1$ nodes $x_0, \ldots, x_n$ are given.

(a) The *zeroth divided difference* of f with respect to the node x_i, denoted by $f[x_i]$, is defined by

$$f[x_i] = f(x_i), \text{ for } 0 \leq i \leq n.$$

(b) The *first divided difference* of f with respect to x_i, x_{i+1}, denoted by $f[x_i, x_{i+1}]$, is defined by

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}, \text{ for } 0 \leq i \leq n-1.$$

(c) In general, the k^{th} *divided difference* of f relative to $x_i, x_{i+1}, \ldots, x_{i+k}$, denoted by $f[x_i, x_{i+1}]$ is defined by

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_0}, \text{ for } 0 \leq i \leq n-k.$$

This process ends with the n^{th} divided difference

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}.$$

Theorem 2.2.2 (Newton's divided difference formula)**.** *Let $P_n(x)$ be the n^{th} Lagrange interpolating polynomial of f with nodes x_i, for $0 \leq i \leq n$. Then*

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k](x - x_0) \ldots (x - x_{k-1}).$$

Theorem 2.2.3 (NDD formula for equally spaced nodes)**.** *Let $P_n(x)$ be the n^{th} Lagrange interpolating polynomial of f with nodes x_i, for $0 \leq i \leq n$ that are equally spaced. Let $h = x_{i+1} - x_i$, for $0 \leq i \leq n-1$, and let $x = x_0 + sh$. Then*

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} \binom{s}{k} k! n^k f[x_0, \ldots, x_k].$$

Theorem 2.2.4 (Newton's forward difference formula)**.** *Let $P_n(x)$ be the n^{th} Lagrange interpolating polynomial of f with nodes x_i, for $0 \leq i \leq n$ that are equally spaced. Let $h = x_{i+1} - x_i$, for $0 \leq i \leq n-1$, and let $x = x_0 + sh$. Then*

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0).$$

Definition 2.2.5. Given a sequence $\{p_n\}$ of numbers, we define the *backward difference* ∇ by

$$\nabla p_n = p_n - p_{n-1}, \text{ for } n \geq 1, \text{ and}$$
$$\nabla^k p_n = \nabla(\nabla^{k-1} p_n), \text{ for } k \geq 2.$$

Theorem 2.2.6 (Newton's backward difference formula)**.** *Let $P_n(x)$ be the n^{th} Lagrange interpolating polynomial of f with equally spaced nodes x_i re-ordered backwards, that is, for $n \geq i \geq 0$. Let $h = x_i - x_{i-1}$, for $n \geq i \geq 1$, and let $x = x_n + sh$. Then*

$$P_n(x) = f(x_n) + \sum_{k=1}^{n} (-1)^n \binom{-s}{k} \nabla^k f(x_n).$$

Theorem 2.2.7. *Suppose that $f \in C^n[a,b]$ and $x_0, \ldots, x_n$ are distinct numbers in $[a,b]$. Then there exists $\xi \in (a,b)$ with*

$$f[x_0, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

Newton's divided difference generation table

x	$f(x)$	1^{st} divided difference	2^{nd} divided difference	3^{rd} divided difference
x_0	$f[x_0]$			
		$f[x_0, x_1]$		
x_1	$f[x_1]$		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f[x_2]$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f[x_3]$			

2.3 Hermite interpolation

Background

Definition 2.3.1. For $0 \leq i \leq m$, let x_i be a numbers in $[a,b]$, and let m_i be a non-negative integer. Suppose that $f \in C^m[a,b]$, where $m = \max_{0 \leq i \leq n} m_i$. Then the *osculating polynomial* approximating f is the polynomial P of least degree such that

$$f(x_i) = P(x_i) \text{ and } \frac{d^k P(x_i)}{dx^k} = \frac{d^k f(x_i)}{dx^k}, \text{ for } 0 \leq i \leq n \text{ and } 0 \leq k \leq m_i.$$

Remark 2.3.2. Let P be an osculating polynomial for f on $[a,b]$ as in Definition 2.3.1.

(a) $\deg(P) \leq \sum_{i=1}^{n} m_i + n.$

(b) When $n = 0$, P is the m_0^{th} Taylor polynomial for f at x_0.

(c) For each i, when $m_i = 0$, P is the n^{th} Lagrange Interpolating polynomial on nodes $x_0, \ldots, x_n$.

Definition 2.3.3. Let P be an osculating polynomial for f on $[a, b]$ as in Definition 2.3.1. When $m_i = 1$ for each i, then P is called the *Hermite interpolating polynomial* of f.

Purpose

Suppose that f is a continuous function whose values at $n+1$ nodes $x_0, \ldots, x_n$ are given. The goal is to provide an iterative method for generating a Hermite polynomial approximating f.

Concept and method

Theorem 2.3.4 (Hermite polynomial)**.** *Let $f \in C^1[a, b]$ and $x_0, \ldots, x_n \in [a, b]$ be distinct numbers. Then the unique polynomial of least degree agreeing with f and f' at $x_0, \ldots, x_n$ is the Hermite polynomial of degree at most $2n+1$ given by*

$$H_{2n+1} = \sum_{j=0}^{n} f(x_j) H_{n,j}(x) + \sum_{j=0}^{n} f'(x_j) \widehat{H}_{n,j}(x),$$

where $L_{n,j}$ is j^{th} Lagrange coefficient polynomial of degree n,

$$H_{n,j}(x) = [1 - 2(x - x_j) L'_{n,j}(x_j)] L^2_{n,j}(x) \text{ and } \widehat{H}_{n,j}(x) = (x - x_j) L^2_{n,j}(x).$$

Moreover, if $f \in C^{2n+2}[a, b]$, then

$$f(x) = H_{2n+1}(x) + \frac{(x - x_0)^2 \ldots (x - x_n)^2}{(2n + 2)!} f^{(2n+2)}(\xi(x)),$$

for some $\xi(x) \in (a, b)$.

Theorem 2.3.5 (Hermite polynomial using divided differences)**.** *For $0 \leq i \leq n + 1$, let x_i be $n + 1$ distinct numbers at which the values of f and f' at x_i are given. Define a new sequence by*

$$z_{2i} = z_{2i+1} = x_i, \text{ for } 1 \leq i \leq n,$$

and construct the divided difference table for the z_i with the additional substitution that

$$f[z_{2i}, z_{2i+1}] = f'(z_{2i}) = f'(x_i), \text{ for } 0 \leq i \leq n.$$

Then the Hermite polynomial takes the form

$$H_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, \ldots, z_k](x - z_0) \ldots (x - z_{k-1}).$$

3 Matrix methods

3.1 Gauss elimination method

Purpose

Given an invertible real matrix $A = (a_{ij})_{n\times n}$ and $b = [b_1 \ \dots \ b_n]^T \in \mathbb{R}^n$, we want to solve the system of linear equations $Ax = b$ for $x = [x_1 \ \dots \ x_n]^T \in \mathbb{R}^n$.

Method

Given a system of linear equation $Ax = b$ (as above), we form the *augmented matrix* $\bar{A} = [A, b]$ define by

$$\bar{A} := \left[\begin{array}{ccc|c} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \end{array}\right].$$

Performing elementary row operation we reduce $\bar{A}$ to a matrix of the form

$$\bar{\bar{A}} := \left[\begin{array}{cccc|c} \tilde{a}_{11} & \dots & & \tilde{a}_{1n} & \tilde{a}_{(1)(n+1)} \\ 0 & \tilde{a}_{22} & & \tilde{a}_{2n} & \tilde{a}_{(2)(n+1)} \\ \vdots & & & \vdots & \vdots \\ 0 & \dots & 0 & \tilde{a}_{nn} & \tilde{a}_{(n)(n+1)} \end{array}\right].$$

Hence, we obtain the following solution.

$$\begin{aligned} x_n &= \frac{a_{(n)(n+1)}}{a_{nn}} \\ x_{n-1} &= \frac{a_{(n-1)(n+1)} - a_{(n-1)(n)}x_n}{a_{(n-1)(n-1)}} \\ \vdots &\quad \vdots \\ x_i &= \frac{a_{(i)(n+1)} - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}, \text{ for } n-1 \geq i \geq 1. \end{aligned}$$

Note that the procedure fails if $\tilde{a}_{ii} = 0$, for some i.

3.2 Gauss elimination method with partial pivoting

Purpose

Using the diagonal elements a_{ii} as pivoting elements in the Gauss elimination method can introduce significant roundoff errors, particularly when the pivots a_{ii} are significantly smaller (in absolute value) when compared with other elements in the i^{th} column below the diagonal element.

Method (Partial pivoting)

Select an element in the same column below the diagonal that has the largest absolute value. More specifically, we choose the smallest $p \geq k$ such that

$$|a_{pk}| = \max_{k \leq i \leq n} |a_{ik}|$$

and swap the k^{th} row with the p^{th} row.

Method (Scaled partial pivoting)

In this method, we first define a scalar factor s_i for each row as

$$s_i = \max_{1 \leq j \leq n} |a_{ij}|.$$

Then we choose the least integer p with

$$\frac{|a_{p1}|}{s_p} = \max_{1 \leq k \leq n} \frac{|a_{k1}|}{s_k}$$

and swap the first row with the p^{th} row. In a similar manner, before eliminating the variable x_i (through row operations), we select the smallest $p \geq i$ with

$$\frac{|a_{pi}|}{s_p} = \max_{i \leq k \leq n} \frac{|a_{ki}|}{s_k}$$

and swap the i^{th} row with the p^{th} row.

3.3 Jacobi and Gauss-Siedel methods

Purpose

These are iterative methods for approximating the solution to a system of equations $Ax = b$.

Method (Jacobi)

First, we solve the i^{th} equation in $Ax = b$ for x_i, to obtain

$$x_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{-a_{ij}x_j}{a_{ii}} + \frac{b_i}{a_{ii}}, \text{ for } 1 \leq i \leq n.$$

Then for each $k \geq 1$, the k^{th} iteration of x_i is given by the formula

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[\sum_{\substack{j=1 \\ j \neq i}}^{n} (-a_{ij}x_j^{(k-1)}) + b_i \right], \text{ for } 1 \leq j \leq n.$$

Method (Gauss-Siedel)

For each $k \geq 1$, the k^{th} iteration of x_i is given by the formula

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} (a_{ij}x_j^{(k-1)}) + b_i \right], \text{ for } 1 \leq i \leq n.$$

3.4 Matrix norms

Definition 3.4.1. A *matrix norm* on $M_n(\mathbb{R})$ is a function

$$\| \cdot \| : M_n(\mathbb{R}) \to \mathbb{R}$$

satisfying the following properties for all matrices $A, B \in M_n(\mathbb{R})$ and all $\alpha \in \mathbb{R}$.

(a) $\|A\| \geq 0$.

(b) $\|A\| = 0 \iff A = O_n$.

(c) $\|\alpha A\| = |\alpha| \|A\|$.

(d) $\|A + B\| \leq \|A\| + \|B\|$.

(e) $\|AB\| \leq \|A\| \|B\|$.

Theorem 3.4.2. *If $\|\cdot\|$ is a vector norm in $\mathbb{R}^n$, then*

$$\|A\| = \max_{\|x\|=1} \|Ax\|$$

is a matrix norm.

The matrix norm induced by a vector norm as in Theorem 3.4.2 is called a *induced matrix norm.*

Corollary 3.4.3. *For any vector $v \neq 0$ and $A \in M_n(\mathbb{R})$, we have*

$$\|Av\| \leq \|A\| \cdot \|v\|.$$

Definition 3.4.4. The ℓ_p and ℓ_∞ norms of a vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ are defined by

$$\|x\|_p = \left(\sum_{i=1}^{n} x_i^p \right)^{1/p} \text{ and } \|x\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Theorem 3.4.5. *For each $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$, we have*

$$|x \cdot y| \leq \|x\|_2 \, \|y\|_2.$$

Theorem 3.4.6. *For any $A \in M_n(\mathbb{R})$, we have*

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|.$$

Definition 3.4.7. The *spectral radius* $\rho(A)$ of a matrix A is defined by

$$\rho(A) = \max\{|\lambda| : \lambda \text{ is an eigenvalue of } A\}.$$

Theorem 3.4.8. *If $A \in M_n(\mathbb{R})$, then*

(i) $\|A\|_2 = [\rho(A^T A)]^{1/2}$, *and*

(ii) $\rho(A) \leq \|A\|$, *for any induced matrix norm $\|\cdot\|$.*

Definition 3.4.9. A matrix $A \in M_n(\mathbb{R})$ is said to be *convergent* if

$$\lim_{k \to \infty} A^k = O_n.$$

Theorem 3.4.10. *Let $A \in M_n(\mathbb{R})$. Then the following statements are equivalent.*

(i) A *is a convergent.*

(ii) $\lim_{k\to\infty} \|A^k\| = 0$, *for some induced matrix norm* $\|\cdot\|$.

(iii) $\lim_{k\to\infty} \|A^k\| = 0$, *for all induced matrix norms* $\|\cdot\|$.

(iv) $\rho(A) < 1$.

(v) $\lim_{k\to\infty} A^k x = \boldsymbol{0}$, *for every* $x \in \mathbb{R}^n$.

3.5 Successive over-relaxation (SOR) method

Definition 3.5.1. Suppose $\tilde{x} \in \mathbb{R}^n$ is an approximate solute of the linear system $Ax = b$. Then *residual vector* for $\tilde{x}$ with respect to this system is $r = b - A\tilde{x}$.

Purpose

In iterative methods such Jacobi and Gauss-Siedel methods, a residual vector is associate with each iteration (of a component) of the solution vector x. We want to generate a sequence of approximations that will cause the residual vectors to converge rapidly to zero.

Method

Let D, U and L denote the $n \times n$ matrices obtained by replacing the diagonal, upper-triangular, and lower triangular entries of the matrix O_n with the corresponding entries of matrix A. The SOR method has the form

$$x^{(k)} = T_\omega x^{(k-1)} + c_\omega, \text{ where } 1 < \omega < 2,$$
$$T_\omega = (D - \omega L)^{-1}[(1-\omega)D + \omega U] \text{ and } c_\omega = \omega(D - \omega L)^{-1}b.$$

Convergence

Theorem 3.5.2. *If $a_{ii} \neq 0$ for all i, then $\rho(T_\omega) \geq |\omega - 1|$. Consequently, the SOR method converges only if $0 < \omega < 2$.*

3.6 Power method

Let $A \in M_n(\mathbb{R})$ be a matrix with n eigenvalues $\lambda_1, \ldots, \lambda_n$ such that

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$$

with an associated collection of linearly independent eigenvectors $\{v_1, \ldots, v_n\}$.

Purpose

This is an iterative technique to estimate the dominant eigenvalue λ_1 of the matrix A and also estimate an eigenvector for λ_1.

Method

We begin by choosing a unit vector $x^{(0)}$ relative to the $\|\cdot\|_\infty$ norm and a component $x_{p_0}^{(0)}$ of $x^{(0)}$ with

$$x_{p_0}^{(0)} = 1 = \|x^{(0)}\|_\infty.$$

For $m \geq 1$, we define sequences of vectors $\{x^{(m)}\}_{m=0}^\infty$ and $\{y^{(m)}\}_{m=1}^\infty$, and a sequence of scalars $\{\mu^{(m)}\}_{m=1}^\infty$ inductively by

$$\begin{aligned} y^{(m)} &= Ax^{(m-1)}, \\ \mu^{(m)} &= y_{p_{m-1}}^{(m)}, \\ x^{(m)} &= \frac{y^{(m)}}{y_{p_m}^{(m)}} = \frac{A^m x^{(0)}}{\prod_{k=1}^{m} y_{p_k}^{(k)}}, \end{aligned}$$

where at each step, p_m is used to represent the smallest integer for which

$$|y_{p_m}^{(m)}| = \|y^{(m)}\|_\infty.$$

Note that

$$\lim_{m\to\infty} \mu^{(m)} = \lambda_1 \text{ and } \lim_{m\to\infty} x^{(m)} = w,$$

where w is an eigenvector associated with λ_1.

3.7 Housholder's method

Background

Definition 3.7.1. Let $w \in \mathbb{R}^n$ with $w^T w = 1$. Then the $n \times n$ matrix

$$P_w = I_n - 2ww^T$$

is called a *Householder's transformation.*

Theorem 3.7.2. *A Householder's transformation $P_w = I_n - 2ww^T$ is both symmetric and orthogonal. Consequently, $P_w = P_w^{-1}$.*

Purpose

This method is used is used to find a symmetric tridiagonal matrix that is similar to a given symmetric matrix $A \in M_n(\mathbb{R})$.

Method

- Let $P^{(1)} = P_{w^{(1)}}$, where $w^{(1)} = (w_1, \ldots, w_n)$ with

$$\begin{aligned} \alpha &= -\text{sgn}(a_{21}) \left(\sum_{j=2}^{n} a_{j1}^2 \right)^{1/2}, \\ r &= \left(\frac{1}{2}\alpha(\alpha - a_{21}) \right)^{1/2}, \\ w_1 &= 0, \\ w_2 &= \frac{a_{21} - \alpha}{2r}, \text{ and} \\ w_j &= \frac{a_{j1}}{2r}, \text{ for } 3 \leq j \leq n. \end{aligned}$$

  With this choice, we define

$$A^{(2)} = P^{(1)} A P^{(1)}.$$

- This process is repeated for $2 \leq k \leq n-2$ as follows:

$$\begin{aligned}
\alpha &= -\text{sgn}(a^{(k)}_{(k+1)k}) \left(\sum_{j=k+1}^{n} (a^{(k)}_{jk})^2 \right)^{1/2}, \\
r &= \left(\frac{1}{2} \alpha (\alpha - a^{(k)}_{(k+1)k}) \right)^{1/2}, \\
w^{(k)}_1 &= w^{(k)}_2 = \ldots = w^{(k)}_k = 0, \\
w^{(k)}_{k+1} &= \frac{a^{(k)}_{(k+1)k} - \alpha}{2r}, \\
w^{(k)}_j &= \frac{a^{(k)}_{jk}}{2r}, \text{ for } k+2 \leq j \leq n, \\
P^{(k)} &= I_n - 2w^{(k)}(w^{(k)})^T, \text{ and} \\
A^{(k+1)} &= P^{(k)} A^{(k)} P^{(k)}.
\end{aligned}$$

- Continuing in this manner, the tridiagonal symmetric matrix $A^{(n-1)}$ is formed, where

$$A^{(n-1)} = P^{(n-2)} \ldots P^{(1)} A P^{(1)} \ldots P^{(n-2)}.$$

3.8 QR Algorithm

Without loss of generality, the method assumes (after application of Householder's method) that the matrix A is a tridiagonal matrix whose diagonal and off-diagonal entries are given by:

$$[a_{11}\, a_{22} \ldots a_{nn}] = [a_1\, a_2 \ldots a_n] \text{ and}$$
$$[a_{12}\, a_{23} \ldots a_{(n-1)n}] = [a_{21}\, a_{32} \ldots a_{n(n-1)}] = [b_2\, b_3 \ldots b_n],$$

respectively.

Purpose

This is a reduction technique to determine all eigenvalues of a symmetric matrix A simultaneously.

Method

Assuming that $b_j \neq 0$ for any j, the QR method proceeds by forming a sequence $A = A^{(1)}, A^{(2)}, \ldots$ as follows.

- The matrix $A^{(1)} = A$ is factored as $A^{(1)} = Q^{(1)}R^{(1)}$, where $Q^{(1)}$ is orthogonal and $R^{(1)}$ is upper-triangular.
- The matrix $A^{(2)} = R^{(1)}Q^{(1)}$.
- In general, $A^{(i)} = Q^{(i)}R^{(i)}$ where $Q^{(i)}$ is orthogonal and $R^{(i)}$ is upper-triangular, and $A^{(i+1)} = R^{(i)}Q^{(i)}$.
- As $A^{(i+1)}$ is tridiagonal and similar to $A^{(i)}$, it has the same eigenvalues as A. Proceeding inductively, $A^{(i+1)}$ tends to a diagonal matrix whose diagonal entries are the eigenvalues of A.

Construction of the $Q^{(i)}$ and the $R^{(i)}$

Definition 3.8.1. A *rotation matrix* $\mathcal{P}_\theta^{ij} = (p_{rs})_{n\times n}$ differs from the identity matrix in at most four elements. These four elements are of the form

$$p_{ii} = p_{jj} = \cos(\theta) \text{ and } p_{ij} = -p_{ji} = \sin(\theta),$$

for some θ and some $i \neq j$.

Remark 3.8.2. A rotation $\mathcal{P}_\theta^{ij}$ has the following properties.

(a) The matrix $A\mathcal{P}_\theta^{ij}$ (resp. $\mathcal{P}_\theta^{ij}A$) differs from A only in the i^{th} and j^{th} columns (resp.

(b) For any $i \neq j$, the angle θ can be chosen so that the product $(\mathcal{P}_\theta^{ij}A)_{ij} = 0$.

(c) $\mathcal{P}_\theta^{ij}$ is orthogonal.

The construction the $Q^{(i)}$ and the $R^{(i)}$ follows these steps:

- First, the factorization of $A = A^{(1)}$ as a product $A^{(1)} = Q^{(1)}R^{(1)}$ uses $n-1$ rotation matrices

  $$R^{(1)} = P_n P_{n-1} \ldots P_2 A^{(1)},$$

  where the P_i are rotation matrices are chosen in the following manner.

1. Choose $P_2 = \mathcal{P}_{\theta_2}^{12}$, where $\theta_2 = \arctan(b_2/a_1)$, and let
$$A_2^{(1)} = P_2 A^{(1)}.$$
Note that $(A_2^{(1)})_{21} = 0$.
2. In general, P_k is chosen so that the $(k, k-1)$ entry in
$$A_k^{(1)} = P_k A_{k-1}^{(1)}$$
is zero, which would imply that the $(k-1, k+1)$ entry is nonzero.
3. Let $(A_k^{(1)})_{kk} = x_k$ and $(A_k^{(1)})_{(k+1)k} = b_{k+1}$. Choose $P_{k+1} = \mathcal{P}_{\theta_k}^{k(k+1)}$ with $\theta_k = \arctan(b_{k+1}/x_k)$, and let
$$A_{k+1}^{(1)} = P_{k+1} A_k^{(1)}.$$
Note that $(A_{k+1}^{(1)})_{(k+1)k} = 0$.
4. Proceeding in the manner in the sequence $P_2, \ldots, P_n$ produces the upper triangular matrix
$$R^{(1)} = A_n^{(1)}.$$
Moreover, the other half of the QR factorization is
$$Q^{(1)} = P_2^T \ldots P_n^T,$$
as $Q^{(1)} R^{(1)} = A^{(1)}$. Note that $Q^{(1)}$ is orthogonal.
5. Finally, we define $A^{(2)} = R^{(1)} Q^{(1)}$ and proceed to the next iteration.

4 Numerical integration

4.1 Trapezoidal and Simpson's rules

Background

Theorem 4.1.1 (Quadrature formula)**.** *Let $\{x_0, \ldots, x_n\}$ be distinct nodes from an interval $[a, b]$, and let f be Riemann integrable on $[a, b]$. Then*
$$\int_a^b f(x)\,dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{1}{(n+1)!} \int_a^b \prod_{i=0}^{n} (x - x_i) f^{(n+1)}(\xi(x))\,dx,$$

where $\xi(x) \in [a, b]$ *for each* x *and for* $1 \leq i \leq n$,

$$a_i = \int_a^b L_i(x)\, dx.$$

Purpose

To find an approximation to the integral $\int_a^b f(x)\, dx$.

Method

Corollary 4.1.2 (Trapezoidal rule)**.** *Substituting* $n = 1$, $x_0 = a$, $x_1 = b$ *and* $h = b - a$ *in quadrature formula in Theorem 4.1.1, yields the Trapezoidal rule given by:*

$$\int_a^b f(x)\, dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12} f''(\xi),$$

where $\xi \in (x_0, x_1)$.

Corollary 4.1.3 (Simpson's rule)**.** *Substituting* $n = 2$, $x_0 = a$, $x_1 = a + h$, $x_2 = b$ *and* $h = (b - a)/2$ *in quadrature formula in Theorem 4.1.1, yields the Simpson's rule given by:*

$$\int_a^b f(x)\, dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90} f^{(4)}(\xi),$$

where $\xi \in (x_0, x_2)$

4.2 Closed Newton-Cotes formula

Purpose

Gives a generalized formula for equally spaced nodes in $[a, b]$.

Method

Corollary 4.2.1 (Closed Newton-Cotes formula)**.** *Substituting* $x_0 = a$, $x_i = a + ih$, *for* $1 \leq i \leq n - 1$, $x_n = b$ *and* $h = (b - a)/n$ *in quadrature formula in*

Theorem 4.1.1, yields the $(n+1)$-point closed Newton-Cotes formula given by:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} a_i f(x_i),$$

where

$$a_i = \int_a^b L_i(x)\,dx = \int_{x_0}^{x_n} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx.$$

Theorem 4.2.2. *Suppose that $\sum_{i=0}^{n} a_i f(x_i)$ denotes the $(n+1)$-point closed Newton-Cotes formula as in Corollary 4.2.1. Then there exists $\xi \in (a,b)$ for which*

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n} a_i f(x_i) + h^{n+3}\frac{f^{(n+2)}(\xi)}{(n+2)!}\int_0^n t^2(t-1)\ldots(t-n)\,dt,$$

if n is even and $f \in C^{n+2}[a,b]$, and

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n} a_i f(x_i) + h^{n+2}\frac{f^{(n+1)}(\xi)}{(n+1)!}\int_0^n t(t-1)\ldots(t-n)\,dt,$$

if n is odd and $f \in C^{n+1}[a,b]$.

Note that the Newton-Cotes formula in Corollary 4.2.1 for $n = 1$ (resp. $n = 2$) yields the Trapezoidal (resp. Simpson's) rule.

Corollary 4.2.3 (Simpson's $(3/8)^{th}$ rule)**.** *The Newton-Cotes formula for $n = 3$ yields the Simpson's $(3/8)^{th}$ rule given by:*

$$\int_a^b f(x)\,dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] - \frac{3h^5}{80}f^{(4)}(\xi).$$

4.3 Gaussian quadrature

Purpose

Chooses nodes for evaluation in an optimal rather an equally spaced way using the roots of the Legendre polynomials to approximate $\int_{-1}^{1} f(x)\,dx$.

Background

Definition 4.3.1. For $n \geq 0$, the n^{th} *Legendre polynomial* $P_n(x)$ is a monic polynomial of degree n that satisfies the the condition

$$\int_{-1}^{1} P(x)P_n(x)\,dx = 0,$$

for every polynomial $P(x)$ of degree $< n$.

Remark 4.3.2. For $n \geq 1$, the polynomial $P_n(x)$ has n distinct roots in $(-1, 1)$ that are symmetric about the origin. The first five Legendre polynomials are

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_2(x) &= x^2 - \frac{1}{3}, \\ P_4(x) &= x^3 - \frac{3}{5}x, \text{ and} \\ P_5(x) &= x^4 - \frac{6}{7}x^2 + \frac{3}{35}. \end{aligned}$$

Theorem 4.3.3. *Suppose that $r_{n1}, r_{n2}, \ldots, r_{nn}$ are the roots of $P_n(x)$ and that for each $1 \leq i \leq n$, the number c_{ni} are defined by*

$$c_{ni} = \int_{-1}^{1} \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}\,dx. \qquad (*)$$

If $P(x)$ is any polynomial of degree less than $2n$, then

$$\int_{-1}^{1} P(x)\,dx = \sum_{i=1}^{n} c_{ni}P(x_i).$$

Method

For $k \geq 1$, let $\{r_{k1}, \ldots, r_{kk}\}$ be the roots of $P_k(x)$ in $(-1, 1)$ and $\{c_{k1}, \ldots, c_{kk}\}$ be the coefficients appearing in $(*)$ of Theorem 4.3.3. Then Gaussian quadrature with $n = k$ nodes yields an approximation to $\int_{-1}^{1} f(x)\,dx$ given by

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{k} c_{ki}f(r_{ki}).$$

References

[1] RL Burden and J Douglas Faires. Numerical analysis 9th edition. *Thompson Brooks/Cole*, 2011.

[2] Wolfram Research, Inc. Mathematica, Version 12.0. Champaign, IL, 2019.