These are the lecture notes for CSC349A Numerical Analysis taught by Rich Little in the Spring of 2018. They roughly correspond to the material covered in each lecture in the classroom but the actual classroom presentation might deviate significantly from them depending on the flow of the course delivery. They are provide as a reference to the instructor as well as supporting material for students who miss the lectures. They are simply notes to support the lecture so the text is not detailed and they are not thoroughly checked. Use at your own risk. They are complimentary to the handouts. Many thanks to all the guidance and materials I received from Dale Olesky who has taught this course for many years and George Tzanetakis.

1 Midterm Logistics

The following information is important for preparing for the midterm exam.

- The midterm is 50 minutes long
- The exam is closed book (see below regarding formula sheet)
- Only simple, scientific calculators (the ones you use for math classes) are allowed. If you bring anything programmable or with a large screen and or internet access you will not be allowed to use it.
- You can bring a single letter size (8 by 11) piece of paper with formulas and notes (it can be double sided)
- The material covered corresponds to parts 1,2 of the textbook and Handouts 1 to 11.
- In terms of topics these are condition, stability, error, Taylor polynomial, floating point arithmetic (part 1).
- Roots of equations (Bisection, Newton and Secant) and Horner's algorithm and rates of convergence (part 2).
- In addition you should study all the assignments you have completed and the corresponding problems from the sample exam questions.

2 Roots of Polynomials

A polynomial of order (degree) n can be written as

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

as well as

$$f(x) = a_n (x - r_1)^{m_1} (x - r_2)^{m_2} \ldots (x - r_k)^{m_k} \text{ with } \sum_{j=1}^{k} m_j = n$$

if $f(x)$ has k distinct roots (real or complex) and r_j is a zero of multiplicity $m_j \geq 1$. If the coefficients a_i are real, then any complex roots occur in conjugate pairs, $\lambda \pm \mu i$ where $i = \sqrt{-1}$.

3 Motivation

Many dynamical systems (e.g. mechanical devices, electrical circuits) are modelled by a linear ordinary differential equation for example :

$$a_2 \frac{d^2 y}{dt^2} + a_1 \frac{dy}{dt} + a_o y = F(t)$$

where the forcing function $F(t)$ represents the effect of the external world on the system.

The homogeneous (general) solution (i.e when $F(T) = 0$) is $y = e^{rt}$. Substituting we have:

$$a_2 r^2 e^{rt} + a_1 r e^{rt} + a_0 e^{rt} = 0 \implies a_2 r^2 + a_1 r + a_0 = 0.$$

This is called the **characteristic** polynomial. Its roots are the eigenvalues and these determine the behavior of the physical system.

One approach to computing the roots of a polynomial $f(x)$ is to use the Newton/Raphson method.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Main issues:

- Efficient evaluation of $f(x_i)$ and $f'(x_i)$.
- How to implement Newton to compute all n roots of $f(x)$
- How to compute complex roots

4 Horner's Algorithm (Nested Multiplication, Synthetic Division)

Given a polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ and a value x_0, this algorithm is used to efficiently evaluate $f(x_0)$ and $f'(x_0)$. To illustrate the basic idea, consider the case $n = 4$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \tag{1}$$

can be rewritten in the form:

$$f(x) = a_0 + x * (a_1 + x * (a_2 + x * (a_3 + x * a_4))) \tag{2}$$

Evaluate of (1) at x_0 requires 7 multiplications and 4 additions, whereas (2) requires only 4 multiplications and 4 additions. The general case (for a polynomial of order n):
form (1) requires $2n - 1$ multiplications and n additions,
from (2) requires n multiplications and n additions
An algorithm to evaluate $f(x_0)$, assuming that $f(x) = \sum_{i=0}^{n} a_i x^i$ is written in the **"nested" form**, as in (2):

$$\begin{aligned}
b_n &= a_n \\
b_{n-1} &= a_{n-1} + b_n x_0 \\
b_{n-2} &= a_{n-2} + b_{n-1} x_0 \\
\cdots & \\
b_0 &= a_0 + b_1 x_0 \\
b_0 &= f(x_0)
\end{aligned}$$

or in more compact form:

$$b_k = a_k + b_{k+1} x_0 \quad \text{for } k = n-1, n-2, \ldots, 1, 0$$

NOTE that execution of this algorithm requires **exactly** n multiplications and n additions.

4.1 Algorithm for evaluating $f'(x_0)$

Let $b_n, b_{n-1}, \ldots, b_0$ be defined as above, the now define:

$$Q(x) = b_1 + b_2x + b_3x^2 + \cdots + b_nx^{n-1}$$

then

$$\begin{aligned} &(x - x_0)Q(x) + b_0 \\ &= (x - x_0)(b_1 + b_2x + b_3x^2 + \cdots + b_nx^{n-1}) + b_0 \\ &= (b_0 - b_1x_0) + (b_1 - b_2x_0)x + \cdots + (b_{n-1} - b_nx_0)x^{n-1} + b_nx^n \\ &= a_0 + a_1x + \cdots + a_{n-1}x^{n-1} + a_nx^n \\ &= f(x) \end{aligned}$$

Differentiating with respect to x gives

$$f'(x) = Q(x) + (x - x_0)Q'(x)$$

which implies that

$$f'(x_0) = Q(x_0)$$

Thus, to evaluate $f'(x_0)$, one first needs to evalute $f(x_0)$ as above, which gives the coefficients $b_n, b_{n-1}, \ldots, b_0$, and then evaluate $Q(x_0)$. The most efficient way to evalute $Q(x_0)$, is to use the nested form for the polynomial $Q(x)$. The following algorithm evaluates both $f(x)$ and $f'(x_0) = Q(x_0)$ using *nested multiplication* to evaluate both of the polynomials.

HORNER'S ALGORITHM

Given values $a_0, a_1, \ldots, a_n$ and x_0, compute:

$$
\begin{aligned}
b_n &= a_n & c_n &= b_n \\
b_{n-1} &= a_{n-1} + b_n x_0 & c_{n-1} &= b_{n-1} + c_n x_0 \\
b_{n-2} &= a_{n-2} + b_{n-1} x_0 & c_{n-2} &= b_{n-2} + c_{n-1} x_0 \\
\ldots & & & \\
b_0 &= a_0 + b_1 x_0 & c_1 &= b_1 + c_2 x_0 \\
\text{Then} & & & \\
b_0 &= f(x_0) & c_1 &= f'(x_0)
\end{aligned}
$$

EXAMPLE

Let $n = 4$ and

$$f(x) = x^4 - 2x^3 + 2x^2 - 3x + 4$$

Using Horner's algorithm to evaluate $f(1)$ and $f'(1)$:

$$
\begin{aligned}
b_4 &= 1 & c_4 &= 1 \\
b_3 &= -2 + (1)(1) = -1 & c_3 &= -1 + (1)(1) = 0 \\
b_2 &= 2 + (-1)(1) = 1 & c_2 &= 1 + (0)(1) = 1 \\
b_1 &= -3 + (1)(1) = -2 & c_1 &= -2 + (1)(1) = -1 \\
b_0 &= 4 + (-2)(1) = 2 & &
\end{aligned}
$$

giving $f(1) = b_0 = 2$ and $f'(1) = c_1 = -1$.

Note that the explicit form of $f'(x)$, namely

$$f'(x) = 4x^3 - 6x^2 + 4x - 3$$

is not obtained; only the *value* of $f'(1)$ is computed. Since $Q(x)$ depends on the value of x_0, which is equal to 1 above, all computations must be re-done in order to evaluate $f'(x)$ at a different value of x.

5 Polynomial Deflation

Having computed one zero, say r_1 of a polynomial $f(x)$ having n zeros $r_1, r_2, \ldots, r_n$ the deflated polynomial is

$$\hat{f}(x) = \frac{f(x)}{x - r_1}$$

Note that $\hat{f}(x)$ is a polynomial of order $n-1$ having roots

$$r_2, \ldots, r_n$$

$\hat{f}(x)$ can be easily determined from Horner's algorithm.

6 Newton's algorithm with Horner and Polynomial Deflation

Outline of a procedure to compute a zero of a polynomail $f(x)$ using Newton's method and Horner's algorith:

- Let x_0 be an initial approximation to a zero of $f(x)$
- for i = 1 to imax
  use Horner's algorithm to evaluate $f(x_{i-1})$ and $f'(x_{i-1})$
  set $x_i \leftarrow x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$
  if $|1 - \frac{x_{i-1}}{x_i}| < \epsilon$ exit
  end
  output failed to converge in imax iterations

Polynomial Deflation Suppose that the values $x_0, x_1, x_2, \ldots$ computed above converge in N iterations. Then x_N is the final computed approximation to some zero, say r_1 of $f(x)$. Now the final computation in the above procedure with Newton's method (after N iterations) is:

$$x_N \leftarrow x_{N-1} - \frac{f(x_{N-1})}{f'(x_{N-1})}$$

If $b_n, b_{n-1}, \ldots, b_0$ are the values computed by Horner's algorithm to evalute $f(x_{N-1})$ that is, in the last step of the above procedure (when $i = N$), then from page 2 of Handout number 13 it follows that:

$$f(x) = (x - x_{N-1})Q(x) + b_0 \tag{3}$$

where

$$Q(x) = b_1 + b_2 x + b_3 x^2 + \cdots + b_n x^{n-1} \tag{4}$$

On letting $x = x_{N-1}$ in (3), we obtain:

$$b_0 = f(x_{N-1}) \approx 0 \text{ since } x_{N-1} \approx x_N \approx \text{ the zero } r_1 \text{ of } f(x)$$

Therefore from (3),

$$f(x) \approx (x - x_{N-1})Q(x)$$

and consequently

$$Q(x) \approx \frac{f(x)}{x - x_{N-1}}$$

That is, the polynomial $Q(x)$ defined in (4) above, is the **deflated polynomial**, it is a polynomial of degree $n - 1$, whose zeroes are equal to those of $f(x)$, except for the zero at $x_{N-1} \approx r_1$. Note that the coefficients $b_1, b_2, \ldots, b_n$ of $Q(x)$ are determined from the last application (when $i = N$) of Horner's algorithm in the procedure at the beginning of these notes.

Example See Handout 14 page 3 - An illustration of the application of Newtons method and Horners algorithm to compute a zero of a polynomial $f(x) = x^4 - 0.2x^3 + 1.8x^2 - 0.6x - 3.6$.

With $x_0 = 2$, Horner's gives:

$$\begin{aligned} b_4 &= 1 & c_4 &= 1 \\ b_3 &= 1.8 & c_3 &= 3.8 \\ b_2 &= 5.4 & c_2 &= 13 \\ b_1 &= 10.2 & c_1 &= 36.2 \\ b_0 &= 16.8 \end{aligned}$$

and Newton's method gives $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{16.8}{36.2} = 1.535912$
You can see the calculations for x_2, x_3, x_4 on Handout 14. Finally,

$$x_5 = x_4 - \frac{f(x_4)}{f'(x_4)} = 1.2000000015$$

Thus, $r_1 \approx x_5 = 1.2000000015$. (Note, the true root is $r_1 = 1.2$.) Now, we deflate the polynomial and do it again. So,

$$Q(x) = 3.00215 + 3.000084x + 1.000038x^2 + x^3$$

Note: If several zeros of $f(x)$ are approximated as above, and several deflations are carried out giving a sequence of deflated polynomials of degrees $n-1, n-2, n-3, \ldots$, then the successive computed zeros tend to become less and less accurate.

Root Polishing

Aply Newton's method to approximate deflated polynomial $Q(x)$, giving a value $\hat{r}$. The value $\hat{r}$ approximates some root r_2 of $f(x)$, but will not be fully accurate. Use $\hat{r}$ as the initial approximation for Newton's method applied to $f(x)$. This will converge very quickly (1 or 2 iterations) to the fully accurate root r_2 (as $\hat{r}$ is very clsoe to r_2).

6.1 Computation of complex roots of polynomail f(x)

One approach is to use Newton's method with complex arithmetic. This requires a complex-valued initial value x_0. Usually needs a very good initial approximation to a complex root for convergence.

Example Let $f(x) = 16x^4 - 40x^3 + 5x^2 + 20x + 6$ with $x_0 = -1 + i$ and $\varepsilon = 10^{-4}$.

In MATLAB,

```
>> Newton(-1+i,1e-4,20,'Complex','ComplexPrime')
 iteration approximation
      0 -1.0000000000000000
      1 -0.7019416036757078
      2 -0.5128917887704155
      3 -0.4104573929932645
      4 -0.3682443627399943
      5 -0.3571805008646267
      6 -0.3560743236521379
```

```
     7 -0.3560617632835127
     8 -0.3560617617473319

ans =

  -0.3561 + 0.1628i
```