Theoretical Computer Science: A Complete Guide

Computer Science

August 17, 2021

Elijah Thompson,
Physics and Math Honors

Solo Pursuit of Learning

Contents

Part I

Comp Phys 381

Chapter 1

Root Finding Methods

1.1.0 Bisection Method

Process 1.1.1. *To find the roots of a function f we first choose a pair of initial values x_B and x_U such that $f(x_B) < 0$ and $f(x_U) > 0$. Then we perform the following iterative procedure:*

1. *Define $x_C = \frac{x_B + x_U}{2}$, and evaluate $f(x_C)$.*
2. *If $f(x_C) > 0$ set $x_U = x_C$*
3. *If $f(x_C) < 0$ set $x_B = x_C$*
4. *Repeat until $|f(x_C)|$ is less than some chosen tolerance, ϵ.*

1.2.0 Newton-Raphson Method

Process 1.2.1. *Consider a real valued function f. Note that the Taylor expansion of f centered at a point x_0 and evaluated at $x_0 + \epsilon$ is*

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + \frac{1}{2}f''(x_0)\epsilon^2 + ... = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)\epsilon^n}{n!} \tag{1.2.1}$$

With the NR Method x_0 is the current estimate for the root of our function. We now truncate the series to terms linear in ϵ:

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + O(\epsilon^2) \tag{1.2.2}$$

where $O(\epsilon^2)$ indicates that the terms of order ϵ^2 and higher are omitted. We set $f(x_0 + \epsilon) = 0$. This gives

$$f(x_0) + f'(x_0)\epsilon = 0 \tag{1.2.3}$$

from which we obtain

$$\epsilon_0 = -\frac{f(x_0)}{f'(x_0)} \tag{1.2.4}$$

setting $\epsilon = \epsilon_0$. We then let $x_1 = x_0 + \epsilon_0$ and calculate a new ϵ_1. We extend this process and define it recursively as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{1.2.5}$$

for an indexing ineger $n \geqslant 0$.

Chapter 2

Non-Linear ODEs

Remark 2.0.1. In this chapter considered the differential equation described by

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = f(x, y, t) \tag{2.0.1}$$

2.1.0 Euler Method

Process 2.1.1. *We first choose initial conditions $x(0) = x_0$ and $y(0) = y_0$. Consider the Taylor series expansion of a function g about $a + h$ centered at a:*

$$f(a + h) = f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + O(h^3) \tag{2.1.1}$$

We apply this to $x(t)$ and $y(t)$ to obtain

$$x(t + \Delta t) = x(t) + \Delta t\frac{dx(t)}{dt} + O(\Delta t^2) \tag{2.1.2}$$

$$y(t + \Delta t) = y(t) + \Delta t\frac{dy(t)}{dt} + O(\Delta t^2) \tag{2.1.3}$$

where we omit terms of order Δt^2. We shall vary t steps discretely and label $\Delta t = t_{n-1} - t_n$, so our equations of motion become

$$x_{n+1} = x_n + y_n\Delta t \tag{2.1.4}$$

$$y_{n+1} = y_n + f(x_n, y_n, t_n)\Delta t \tag{2.1.5}$$

using Euler's Method for approximating integrals. In particular, for Euler's method we take

$$g(b) - g(a) = \int_a^b \frac{dg}{dt}dt \approx \sum_{i=1}^{n} \frac{dg(t_i)}{dt}\Delta t, t_1 = a, t_n = b - \Delta t \tag{2.1.6}$$

2.2.0 Trapezoid Rule

Process 2.2.1. *For the trapezoid rule we approximate an integral as follows*

$$(b-a)\left[\frac{\frac{dg(a)}{dt}+\frac{dg(b)}{dt}}{2}\right] \approx \int_a^b \frac{dg}{dt}dt = g(b)-g(a) \tag{2.2.1}$$

Then, applying this to our DE we obtain the iterative equation

$$x_{n+1} \approx x_n + \frac{\Delta t}{2}\left(y_n + y_{n+1}\right) \tag{2.2.2}$$

where $\Delta t = t_{n+1} - t_n$. We then approximate y_{n+1} using Euler's Method

$$y_{n+1} \approx y_n + \Delta t f(x_n, y_n, t_n) \tag{2.2.3}$$

Substituting this back into our previous approximation for x_{n+1} we have

$$x_{n+1} \approx x_n + \frac{\Delta t}{2}\left[y_n + \left(y_n + \Delta t f(x_n, y_n, t_n)\right)\right] \tag{2.2.4}$$

We apply this again to y_{n+1} to obtain the approximation

$$y_{n+1} \approx y_n + \frac{\Delta t}{2}\left[f(x_n, y_n, t_n) + \delta t f(x_{n+1}, y_n + \Delta t f(x_n, y_n, t_n), t_{n+1})\right] \tag{2.2.5}$$

This pair of iterative equations constitute the application of the trapezoid rule to solving our DE.

2.3.0 Runge-Kutta

Process 2.3.1 (Second Order). *We first carry out a Taylor expansion of $\frac{dx(t)}{dt}$ about the mid-points of our interval, $\Delta t/2$:*

$$\frac{dx(t)}{dt} = \frac{dx(\Delta t/2)}{dt} + \frac{d^2x(\Delta t/2)}{dt^2}(t - \Delta t/2) + O\left(\frac{\Delta t^2}{2}\right) \tag{2.3.1}$$

We use this expansion to approximate the following integral:

$$\int_0^{\Delta t} \frac{x(t)}{dt}dt \approx \frac{dx(\Delta t/2)}{dt}\Delta t + \frac{d^2x(\Delta t/2)}{dt^2}\int_0^{\Delta t}(t - \Delta t/2)dt = \frac{dx(\Delta t/2)}{dt}\Delta t \tag{2.3.2}$$

Then, the rule to update x in an algorithmic notation is given by

$$x_{n+1} \approx x_n + y_{n+1/2}\Delta t + O\left(\frac{\Delta t^2}{2}\right) \tag{2.3.3}$$

and applying the Taylor expansion to $y_{n+1/2}$ we have

$$y_{n+1/2} \approx = y_n + f(x_n, y_n, t_n)\frac{\Delta t}{2} + O\left(\frac{\Delta t^2}{2}\right) \tag{2.3.4}$$

Substituing into our x_{n+1} rule we have

$$x_{n+1} \approx x_n + \left(y_n + f(x_n, y_n, t_n) \frac{\Delta t}{2} \right) \Delta t \tag{2.3.5}$$

The rule for updating y_n is given by

$$y_{n+1} = y_n + f(x_{n+1/2}, y_{n+1/2}, t_{n+1/2}) \Delta t \tag{2.3.6}$$

$$y_{n+1/2} = y_n + f(x_n, y_n, t_n) \frac{\Delta t}{2} \tag{2.3.7}$$

$$x_{n+1/2} = x_n + y_n \frac{\Delta t}{2} \tag{2.3.8}$$

Process 2.3.2 (Fourth Order). *In implementing we take a series of approximations for x_n and y_n then take a weighted average of the result. In particular, we take the approximations*

$$k_{1x,n} = \Delta t y_n \tag{2.3.9}$$

$$k_{1y,n} = \Delta t f(x_n, y_n, t_n) \tag{2.3.10}$$

$$k_{2x,n} = \Delta t \left(y_n + \frac{k_{1y,n}}{2} \right) \tag{2.3.11}$$

$$k_{2y,n} = \Delta t f \left(x_n + \frac{k_{1x,n}}{2}, y_n + \frac{k_{1y,n}}{2}, t_n + \Delta t/2 \right) \tag{2.3.12}$$

$$k_{3x,n} = \Delta t \left(y_n + \frac{k_{2y,n}}{2} \right) \tag{2.3.13}$$

$$k_{3y,n} = \Delta t f \left(x_n + \frac{k_{2x,n}}{2}, y_n + \frac{k_{2y,n}}{2}, t_n + \Delta t/2 \right) \tag{2.3.14}$$

$$k_{4x,n} = \Delta t \left(y_n + k_{3y,n} \right) \tag{2.3.15}$$

$$k_{4y,n} = \Delta t f \left(x_n + k_{3x,n}, y_n + k_{3y,n}, t_n + \Delta t \right) \tag{2.3.16}$$

$$x_{n+1} = x_n + \frac{k_{1x,n} + 2k_{2x,n} + 2k_{3x,n} + k_{4x,n}}{6} \tag{2.3.17}$$

$$y_{n+1} = y_n + \frac{k_{1y,n} + 2k_{2y,n} + 2k_{3y,n} + k_{4y,n}}{6} \tag{2.3.18}$$

Chapter 3

Numerical Fourier Analysis

3.1.0 Fourier Series

Definition 3.1.1 (Fourier Series). *Any periodic function $f(t)$ with period $T = 2\pi/\omega$ can be represented as a* **Fourier Series**

$$f(t) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(n\omega t) + b_n \sin(n\omega t)] \tag{3.1.1}$$

The frequency ω is known as the **fundamental frequency** *and $\omega_n = n\omega$ for $n > 1$ are the* **harmonics**. *The result of Fourier-analysing a signal is a set of values for these coefficients for all n.*

Process 3.1.2. *The Fourier coefficients for a periodic function f are evaluated using the orthogonality properties of sines and cosines:*

$$\frac{2}{T}\int_0^T \sin(n\omega t)\sin(k\omega t)dt = \delta_{nk} \tag{3.1.2}$$

$$\frac{2}{T}\int_0^T \cos(n\omega t)\sin(k\omega t)dt = 0 \tag{3.1.3}$$

$$\frac{2}{T}\int_0^T \cos(n\omega t)\cos(k\omega t)dt = \delta_{nk} \tag{3.1.4}$$

Applying these orthogonality properties we obtain the following equations for the coefficients:

$$a_0 = \frac{1}{T}\int_0^T f(t)dt \tag{3.1.5}$$

$$a_k = \frac{2}{T}\int_0^T f(t)\cos(k\omega t)dt, k \in \{1, 2, 3, ...\} \tag{3.1.6}$$

$$b_k = \frac{2}{T}\int_0^T f(t)\sin(k\omega t)dt, k \in \{1, 2, 3, ...\} \tag{3.1.7}$$

3.2.0 Simpson's Rule

Process 3.2.1. *Simpson's rule is a method of numerical integration. In particular, to approximate the integral* $\int_a^b f(x)dx$ *we split the interval* $[a,b]$ *into* n *steps of length* $h = (b-a)/n$, *where* $n \in 2\mathbb{Z}$. *The approximation is taken as*

$$\int_a^b f(t)dt \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right] \tag{3.2.1}$$

where $x_j = a + jh$ *for* $j \in \{0,1,2,...,n-1,n\}$. *In particular* $x_0 = a$ *and* $x_n = b$.

3.3.0 Fourier Integral

Remark 3.3.1. For a non-periodic function $f(t)$ we require a ***Fourier integral*** over a continuous range of frequencies. The Fourier integral may be viewed as a limit of a Fourier series in the limit $T \to \infty$.

Process 3.3.1. *For a non periodic function* $f(t)$ *its Fourier integral is given by*

$$f(t) = \int_0^\infty [a(\omega)\cos(\omega t) + b(\omega)\sin(\omega t)]\, d\omega \tag{3.3.1}$$

and the coefficient equations become functions of ω:

$$a(\omega) = \frac{1}{\pi}\int_{-\infty}^{\infty} f(t)\cos(\omega t)dt \tag{3.3.2}$$

$$b(\omega) = \frac{1}{\pi}\int_{-\infty}^{\infty} f(t)\sin(\omega t)dt \tag{3.3.3}$$

Using Euler's identity $e^{i\omega t} = \cos(\omega t) + i\sin(\omega t)$ *we can rewrite this as*

$$f(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} F(\omega)e^{i\omega t}d\omega \tag{3.3.4}$$

$$F(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \tag{3.3.5}$$

where $F(\omega)$ *is the* **Fourier Transform** *of* $f(t)$. *If the signal has dimensions of energy, then its Fourier Transform has units of Power, and its magnitude* $|F(\omega)|$ *is a measure of the total power in the signal at frequency* ω:

$$|F(\omega)| = \sqrt{\mathbb{R}e\{F(\omega)\}^2 + \mathbb{I}m\{F(\omega)\}^2} = \sqrt{\pi}2\sqrt{a^2(\omega) + b^2(\omega)} \tag{3.3.6}$$

3.4.0 Discrete Fourier Transform

Definition 3.4.1. *In practice we approximate the Fourier integral and its other corresponding forms using finite summations, known as the* **Discrete Fourier Transform**. *Let* $f(t)$ *be a non-periodic function that we have* N *samples of at intervals* h *going from* $t = 0$ *to* $t = (N-1)h$.

We define a discrete timeline by $t_m = mh$ for $m \in \{0, 1, 2, ..., N-1\}$. The time $\tau = Nh$ will become the period of our approximated function under reconstruction, and we need τ to be the longest time over which we are interested in the behaviour of $f(t)$. We also assume

$$f(t) = f(t+\tau) \iff f(t_m) = f(t_{m+N}) \iff f_m = f_{m+N} \tag{3.4.1}$$

The lowest frequency in the DFT will be $\nu_1 = 1/\tau = 1/(Nh)$, and this will be the fundamental frequency of our reconstructed function. The frequency spectrum is given by

$$\Lambda := \left\{ \nu_n = \frac{n}{Nh} = n\nu_1 | n \in \mathbb{N} \right\} \tag{3.4.2}$$

We then discretize the integrals for the function and its Fourier transform as

$$f_m = frac1N \sum_{n=0}^{N-1} F_n e^{i2\pi\nu_n t_m} = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{i2\pi mn/N} \tag{3.4.3}$$

$$F_n = \sum_{m=0}^{N-1} f_m e^{-i2\pi\nu_n t_m} = \sum_{m=0}^{N-1} f_m e^{-2\pi mn/N} \tag{3.4.4}$$

Note that it can be shown that $F_{N/2-n} = \overline{F}_{N/2+n}$ for $n \in \{0, 1, ..., N/2\}$. The highest frequency component is thus $F_{N/2-1}$, corresponding to a frequency of

$$\nu_{max} = (N/2-1)/Nh = 1/(2h) - 1/(Nh) \approx 1/(2h), \text{ if } N \text{ large} \tag{3.4.5}$$

This is also known as the **nyquist frequency** *$\nu_{Nyquist}$.*

If the function has a component with frequency $\nu > \nu_{Nyquist}$ there are less than two sample points per period. This implies that there will be one or more frequencies less than $\nu_{Nyquist}$ for which the amplitude equals the true amplitude at the sample points, but these lower frequencies are not in the signal although they will appear in the frequency spectrum - this is phenomenon known as **aliasing**. *The power spectrum of the DFT is often plotted as all values*

$$P_n = |F_n|^2 = \mathbb{R}e\{F_n\}^2 + \mathbb{I}m\{F_n\}^2 \tag{3.4.6}$$

Process 3.4.2. *In application we use the following summations for the components of F_n*

$$\mathbb{R}e\{F_n\} = \sum_{m=0}^{N-1} f_m \cos\left(\frac{2\pi mn}{N}\right) \tag{3.4.7}$$

$$\mathbb{I}m\{F_n\} = \sum_{m=0}^{N-1} f_m \sin\left(\frac{2\pi mn}{N}\right) \tag{3.4.8}$$

We then reconstruct the original signal as

$$f_m = \frac{1}{N} \sum_{n=0}^{N-1} \left\{ \mathbb{R}e\{F_n\} \cos\left(\frac{2\pi mn}{N}\right) + \mathbb{I}m\{F_n\} \sin\left(\frac{2\pi mn}{N}\right) \right\} \tag{3.4.9}$$

Chapter 4

Curve-fitting and Optimization

4.1.0 Least Squares

Process 4.1.1. *Assume we have some sequence of measurements at times t_i*

$$y_i = y(t_i) \tag{4.1.1}$$

and for some presumed model of the relationship

$$y = f(t; p) \tag{4.1.2}$$

expressed in terms of the independent variable t and the model parameters p. We define the distance between a data point and our model by

$$\delta_i = y_i - y_i \tag{4.1.3}$$

A general measure of the distance is

$$\sum_i |\delta_i|^d \tag{4.1.4}$$

For $d = 2$ we obtain the **chi-squared** *measure*

$$\chi^2 = \sum_i |y_i - y_i(p_1, ..., p_K)|^2 \tag{4.1.5}$$

The best fit is assumed to minimize χ^2 with respect to the model parameters

$$\frac{\partial \chi^2}{\partial p_l} = \sum_i 2|y_i - y_i(p_1, ..., p_K)| \frac{\partial y_i}{\partial p_l} \tag{4.1.6}$$

We also usually use the **reduce chi-squared** *value*

$$\chi_N^2 = \frac{\chi^2}{N} \tag{4.1.7}$$

In general we want χ^2 to scale with the uncertainty in our measurements, so we wish to minimize

$$\sum \left(\frac{expected - observed}{uncertainty} \right)^2 \tag{4.1.8}$$

Remark 4.1.1. This minimization can be done numerically using ***scipy.optimize*** package's ***minimize*** method. This package also has a ***curve_fit*** method for non-linear data sets.

4.2.0 Finite Differences

Process 4.2.1. *Consider a real-valued function $f(x)$ and its Taylor series about some point $x = a$:*

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{4.2.1}$$

Consider a set of points, x_i, such that $x_{i+1} = x_i + \Delta$. Then we have that

$$f(x_{i+n}) = f(x_i) + f'(x_i)n\Delta + \frac{f''(x_i)}{2!}(n\Delta)^2 + \frac{f'''(x_i)}{3!}(n\Delta)^3 + O(\Delta^4) \tag{4.2.2}$$

Define $f(x_{i+n}) =: f_{i+n}$. Then for neighboring points we have

$$f_{i+1} = f_i + f_i'\Delta + \frac{f_i''}{2!}\Delta^2 + \frac{f_i'''}{3!}\Delta^3 + O(\Delta^4) \tag{4.2.3}$$

$$f_i = f_i \tag{4.2.4}$$

$$f_{i-1} = f_i - f_i'\Delta + \frac{f_i''}{2!}\Delta^2 - \frac{f_i'''}{3!}\Delta^3 + O(\Delta^4) \tag{4.2.5}$$

Subtracting the expression for two neighboring points we have

$$f_{i+1} - f_i = f_i'\Delta + \frac{f_i''}{2!}\Delta^2 + \frac{f_i'''}{3!}\Delta^3 + O(\Delta^4) \tag{4.2.6}$$

Dividing by Δ we obtain the following **forward difference** *estimate of the first derivative*

$$\frac{f_{i+1} - f_i}{\Delta} = f_i' + \frac{f_i''}{2!}\Delta + \frac{f_i'''}{3!}\Delta^2 + O(\Delta^3) \approx f_i' + O(\Delta) \tag{4.2.7}$$

Similarly we obtain the **backward difference** *estimate*

$$f_i' \approx \frac{f_i - f_{i-1}}{\Delta} + O(\Delta) \tag{4.2.8}$$

We can also cancel all even terms in the expansion by

$$f_{i+1} - f_{i-1} = 2f_i'\Delta + 2\frac{f_i'''}{3!}\Delta^3 + O(\Delta^5) \tag{4.2.9}$$

which gives us the **centered difference** *estimate*

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2\Delta} + O(\Delta^2) \tag{4.2.10}$$

By adding terms we can cancel all odd terms and obtain

$$f_{i+1} + f_{i-1} = 2f_i + 2\frac{f_i''}{2!}\Delta^2 + O(\Delta^4) \tag{4.2.11}$$

We then obtain an expression for the **second difference** *estimate*

$$f_i'' = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta^2} + O(\Delta^2) \tag{4.2.12}$$

Appendices

.1.0 Lambda Functions

Definition .1.1. *Lambda functions are in practice one-line functions which cannot contain commands or more than one expression. In particular, a Lambda function can be created to and assigned to a variable in Python by*

$$g = \textbf{lambda}\ args_{array} : function\ rule \tag{.1.1}$$

.2.0 List Comprehension

Definition .2.1. *List comprehension is a method of defining and filling a list all in one step. In general, list comprehension can be implemented in Python by*

$$list = [item\ for\ item\ in\ old\ list\ if\ P(item) == True] \tag{.2.1}$$

.3.0 ODE-int Solve

Definition .3.1. *The* **scipy.integrate.odeint** *method can be used to numerically solve a system of differential equations. Define a method which takes the input vector of the system, a timeline, as well as any other needed parameters. Then, implement odeint by*

$$result = odeint(system\_function, y0, t, args = (arg\_tuple)) \tag{.3.1}$$

where $y0$ is an initial state vector.

Appendices