The Theoretics & Applications of Computational Mathematics in Our World

Introduction

“Computational mathematics is essentially the foundation of modern scientific computing”. These were the first words within the book ‘Introduction To Computational Mathematics 2nd Edition’ written by Yang (2014). This phrase can be discussed to be true as many mathematical branches combined with the assistance of computer science has been applied within various industries such as finance and healthcare. Three specific mathematical branches have been selected to discuss their theoretical concepts, implementation into real-world application and their practical efficiency when represented as Python code.

- Modular Arithmetic
- Limits & Differential Calculus
- Mathematical Programming & Optimization

The following three mathematical branches listed at the top have been selected.

Modular Arithmetic

This mathematical branch involves performing arithmetic onto integers while considering a certain value, defined as the modulus. There are many mathematical operations under modular arithmetic such as congruent modulo that have similar methods, but different expressions presented.

Congruent Modulo

A modular arithmetic mathematical operation that defines the relationship of two integers using their remainder. The difference of integers **a** and **b** are equal to the multiple of the modulus **kn**. When integers **a** & **b** are divisible by the remainder individually, we only take into account the modulus produced. This determines whether the integers are congruent modulo to the modulus if they share a common remainder.

“a-b=kn”

***a* ≡ *b* (mod *n*) when a-b is divisible by the modulus**

For example:

23 ≡ 8 (mod 5) where the modulus n = 5. Using this information, we can find the remainders the remainders of integers a = 23 and b= 8 and determine whether they are congruent modulo to modulus 5:

- 23 (mod 5) gives a remainder of 3 and 8 (mod 5) gives a remainder of 3. This means we could define this as **a(mod n) = b(mod n)**

- The difference between 23 and 8 is 15, where 15 can be divisible by 3 therefore we can say 23 and 8 are congruent modulo to modulus 5

Modular Exponentiation

Another modular arithmetic mathematical operation which calculates the remainder by using a number raised by an exponent divided by the modulus. Shown as **“a^b (mod n)”**

For example, to find *2^10* (mod *11*) where the modulus n = 11, a = 2 and b = 10:

- Calculate 2^10 which gives 1024
- Perform 1024 (mod 11) that provides the remainder 1. This means 2^10 (mod 11) = 1

Modular Inverse

Finds the reciprocal of a number which can be defined as **“1/a or a^-1”**. The multiplicative inverse is a number when multiplied with its original number provides the result of 1.

Rivest-Shamir-Adleman Algorithm

There are many applications of modular arithmetic, especially within cryptography and security however the most notable application of modular arithmetic which applies most of its mathematical operation is the Rivest-Shamir-Adleman Algorithm (RSA). It is an efficient cryptographic algorithm which uses asymmetric encryption using public and private keys and digital signatures. The decryption key is private, meaning the person with the correct decryption key can decipher the encrypted message however this key is not made in a way where it can be easily cracked by using the public encryption key (Milanov, 2009). Congruent modulo is implemented when generating these keys.

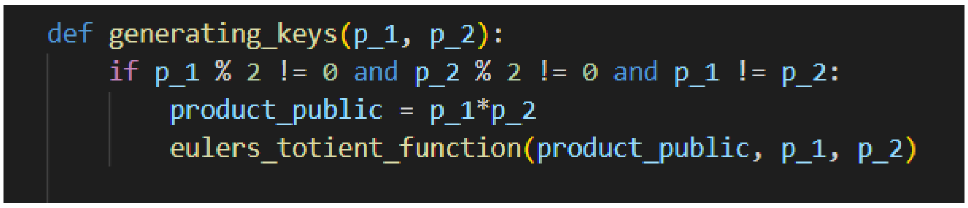
Generating the Keys:

- Two large distinct prime numbers **a** and **b** are chosen. This is because the product from these numbers can only be found by these specific two prime numbers and by itself and 1

- The product **n** between the two large prime numbers, defined as **“axb = n”** which will indirectly contribute to the public key.

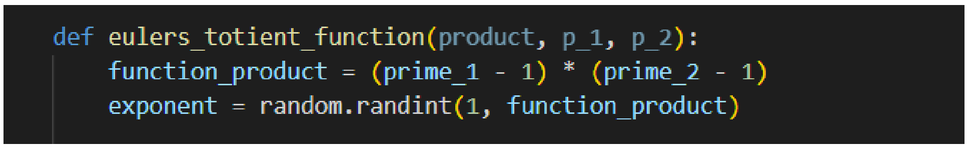
- Using the Euler's Totient function, we do $\boldsymbol{\phi(n)=(a-1)\times(b-1)}$

- Finally, an integer **q** is chosen that is between 1 and $\phi(n)$ where the integer **q** is only coprime with $\phi(n)$. This integer q is the exponent for the public encryption key that will involve modular multiplication.



```python
def generating_keys(p_1, p_2):
    if p_1 % 2 != 0 and p_2 % 2 != 0 and p_1 != p_2:
        product_public = p_1*p_2
        eulers_totient_function(product_public, p_1, p_2)
```

Figure 1 - Python Code for Generating Keys



```python
def eulers_totient_function(product, p_1, p_2):
    function_product = (prime_1 - 1) * (prime_2 - 1)
    exponent = random.randint(1, function_product)
```

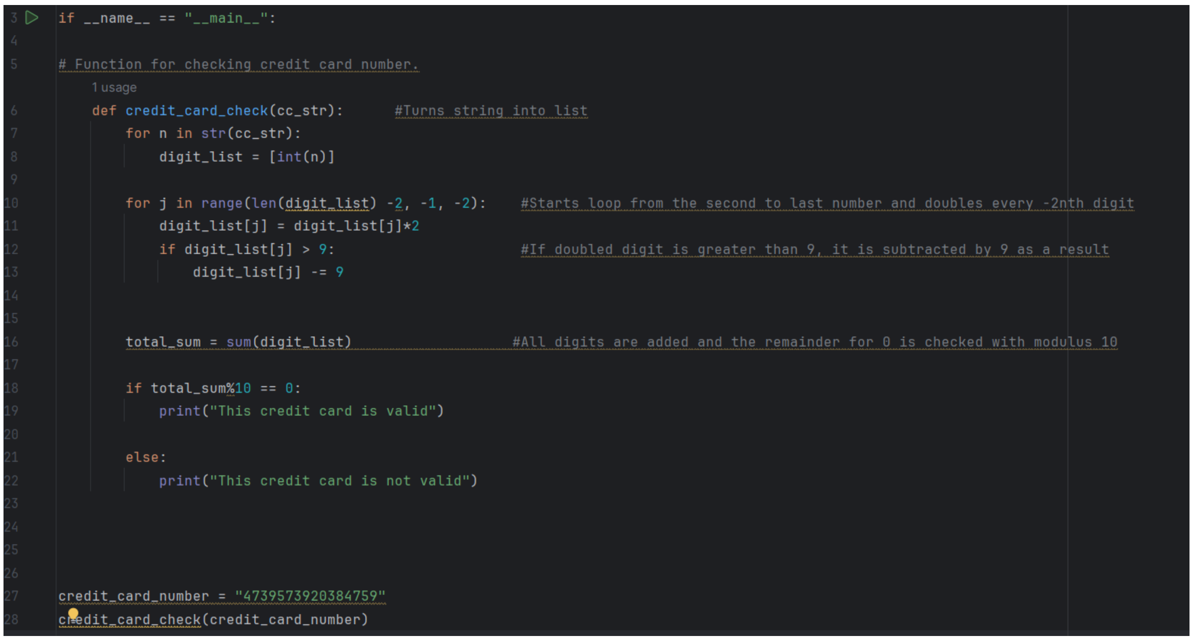
Figure 2 - Python Code of Euler's Totient Function

The efficiency of applying the key generation process of the RSA Algorithm into Python code varies depending on several factors. One factor is the size of the two randomly generated prime numbers which would cause more time taken to generate the keys and determining the size of the keys produced. Also, it would provide more computational power during execution as this specific program involves applying complex mathematical operations such as congruent modulo. Although increasing the size of the keys improves security, it would decrease availability as a result. Another factor is the application of the RSA Algorithm into Python itself. This is because Python is not as memory concise compared to low-level language such as C so performance would not be as optimised however Python contains many libraries relating to cryptography so their involvement could make up for performance.

Validation of Credit Card Numbers

The validation of credit card numbers is a real-world application of modular arithmetic which implements the Luhn Algorithm. This algorithm uses the second to last digit of a sequence of digits, doubles a digit every 2nth of a digit from right to left, provides the sum of the digits and check if the sum has a remainder of zero with modulus 10. If the sum of the altered 16 digits of the card has a remainder 0 with modulus 10, then it is a valid credit card.

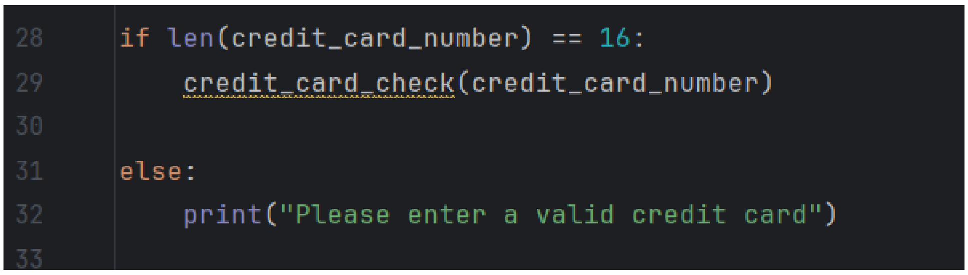
For example, we want to define whether 4830-7593-7583-5769 is a valid credit card number



```python
if __name__ == "__main__":

# Function for checking credit card number.
    1 usage
    def credit_card_check(cc_str):      #Turns string into list
        for n in str(cc_str):
            digit_list = [int(n)]

        for j in range(len(digit_list) -2, -1, -2):    #Starts loop from the second to last number and doubles every -2nth digit
            digit_list[j] = digit_list[j]*2
            if digit_list[j] > 9:                      #If doubled digit is greater than 9, it is subtracted by 9 as a result
                digit_list[j] -= 9


        total_sum = sum(digit_list)                   #All digits are added and the remainder for 0 is checked with modulus 10

        if total_sum%10 == 0:
            print("This credit card is valid")

        else:
            print("This credit card is not valid")




credit_card_number = "4739573920384759"
credit_card_check(credit_card_number)
```

Figure 3 – Python Code for Credit Card Validation

The length of the credit card number changes the efficiency of this code. A typical credit card has 16 digits so the loop will go through 16 iterations with simple calculations of multiplication and addition. This code specifically does not check the length of the credit card which could be added to provide simple validation to show whether it is a valid credit card



```python
if len(credit_card_number) == 16:
    credit_card_check(credit_card_number)

else:
    print("Please enter a valid credit card")
```

Figure 4 – Python Code with added validation

Limits & Differential Calculus

Another notorious mathematical branch which is widely known that comes under calculus but focuses on differentiation. Differentiation is the study the rate of change of a variable as it decreases (or increases).

Using Differential Calculus on a Straight Line

To summarise differential calculus, it is splitting an item into smaller and smaller pieces where it’s rate of change can be calculated as the item is split into an infinite number of small pieces. It can be used to show the rate of change of the straight line on a graph as it increases or decreases. This is the derivative of the line. To find the derivative, we must acquire the derivative. The gradient of the straight line can be found by dividing the change of the vertical coordinates of the straight line(y) by the change of the horizontal coordinates of the straight line(x).

To find the derivative (**dy/dx** or **f'(x)** or also defined as the function representing the rate of change of one variable with respect to another variable, we apply differential calculus. Rules are applied to functions to find their derivatives based on how they are expressed. Some differential calculus rules include:

- **Constant Rule** -> when y = k, the derivative is 0 (for example, x = 35 so f'(x) = 0)
- **Power Rule** -> when y = x^n, the derivative is f'(y) = nx^n-1
- **Product Rule** -> where f(x) = u & g(x) = v, f'(x) = u' & g'(x) = v': d/dx[f(x)g(x)] = uv' + vu'

- **Quotient Rule** -> where f(x) = u & g(x) = v, f'(x) = u' & g'(x) = v'd/dx[f(x)g(x)] = vu' - uv'/v^2

- **Chain Rule** -> where f(g(x)), f'(g(x)) x g'(x)

This is useful when finding the derivative of curves on a graph as they have changing gradients along with other mathematical problems involving logarithms, exponentials and trigonometry.

Finding and Using Limits

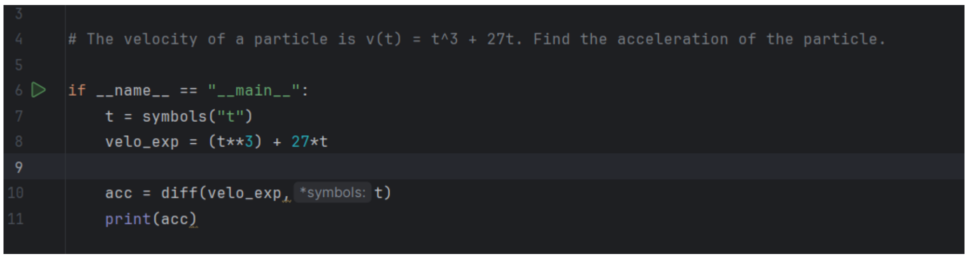
Limits describes how a function acts as it tends to a certain point within a sequence or series. They are involved in differential calculus. There are methods to find limits such as direct substitution. This uses the substituted value given to find the maximum limit of the function.

To find the limit of this function of **lim (3x - 1) as x -> 2 (tends to):**

- Using direct substitution, we can substitute 2 into x within the function to find it's limit where we get **lim (3(2) -1)** which gives us 5 therefore the limit is 5

Finding Quantities in Motion Physics

Differential calculus has many real-world applications, mostly notably with analysing change between variables. For example, it is used within the scientific community, especially in the field of physics, engineering and biology. Differentiation can be used to find vector quantities relating to motion such as velocity and acceleration. Acceleration can be found by differentiating the function representing velocity as it is the derivative of velocity with respect to time. Velocity can be found by using differential calculus on displacement to find it's derivative as velocity is the derivative. This is applied when a mass of a body is in motion.



```
# The velocity of a particle is v(t) = t^3 + 27t. Find the acceleration of the particle.

if __name__ == "__main__":
    t = symbols("t")
    velo_exp = (t**3) + 27*t

    acc = diff(velo_exp, *symbols: t)
    print(acc)
```

Figure 5 – Python Code on Simple Differentiation

This simple program involves simple symbolic differentiation which uses the SymPy library. This library is very efficient for symbolic differentiation, offering the accurate derivative of the given expression however the computational time taken for the program to execute will increase as more complex expressions are given. The performance of this program may vary depending on the given expression and how it is represented.

Calculating Changes in Sales for Businesses

Differential calculus can be applied to finance along with several mathematical branches like optimization. It is used to find maximum sales by using the derivative of the profit function. The profit formula could be represented as any expression however it is commonly represented as P **= TR + TC** (Profit = Total Revenue – Toal Costs). To find the derivative of the profit function, we use differential calculus. We make the derivative of the profit function equal 0, rearrange the derived variables so they equate to each other and find the value of sales is at its maximum showed in the original profit equation.

Mathematical Programming & Optimization

The final mathematical branch which will be mentioned operates on dealing with finding the optimal solution within a group of alternatives. It is used to label the values which will deliver minimum or maximum results. There are five terms involved when dealing with mathematical optimisation:

- Objective Function -> the function that is optimised with the changing variable(s).
- Decision Variables -> the variables altered to minimise or maximise objective function
- Constraints -> conditions needed to be followed while optimising the objective function
- Feasible Region -> set of all possible combinations of decision variables that satisfy constraints
- Optimal Solution -> set of values for the decision variables that provides the maximum (or minimum) value with the objective function within the constraints

An example, the following objective function and constraints could be applied:

Maximise $5X + 8$ with constraints $X < 100$, $X > 0$, $2X < 40$:

- Using the constraints, we know that the value of X is less than 100 but greater than 0. It is also known that the double of X is less than 40 so we can do $39/2 = 19.5$. This makes 19.5 the maximum value of X

Portfolio Optimisation

Optimisation collaborates with sectors which strive to produce maximum results. In finance, optimisation can be implemented to gather possible maximum profits from financial assets such as stocks and cryptocurrencies.

Supply Chain Management

It is involved with calculating changes in sales within businesses with differentiation and like portfolio optimisation, it involves optimising the amount of goods for profit while minimizing expenses.

Conclusion

In conclusion, there are many real-world applications of computational mathematics significantly affecting the way we live as a society. It is useful and efficient for humanity to apply, especially as technology evolves and Python is commonly implemented as the main programming language for data science and other mathematical practices.

Reference List

- Yang, X.-S. (2014). *Introduction to Computational Mathematics*. World Scientific Publishing Company.

- Milanov, E., (2009). The RSA algorithm. *RSA laboratories*, pp.1-11.

- Karloff, H. (2023). Modular Arithmetic. In: Mathematical Thinking. Compact Textbooks in Mathematics. Birkhäuser, Cham. https://doi.org/10.1007/978-3-031-33203-6_3