FORTRAN PROGRAMS for Physics Students

(With THEORY and ALGORITHM)

Abhijit Kar Gupta

A Brief Introduction to writing FORTRAN Programs:

You may use **Force2** or any other freely downloadable (or not) software for FORTRAN compiler to be used on Microsoft Window. While in Linux, you have the compiler inbuilt there.

To write a Program:

	• Write a program here between 7 and 72 columns. • Columns 1 to 6 are reserved for special purposes. • Column 6 is particularly reserved for making a continuation of a line. (If a line exceeds 72 columns, then we may put a special symbol from the keyboard, such as *, !, $ *etc*. to link one line with the next. • In FORTRAN, each line is treated as a command to be executed. • Programs written in upper case letters (Capital Letters) or in lower case letters have no difference.	

Variables:

Name of a variable

- The name of a variable can be composed of letters, digits and underscore character.
- Names are not case sensitive.
- The first character of a variable must be a letter.
- The length of the name of a variable must not exceed 31 characters.

Type of a variable

- Integer (Integer values)
- Real (Floating point numbers)
- Complex (Complex Numbers)
- Logical (Logical Boolean Values)
- Character (Characters or Strings)

Note:

In general, the variable names starting with (I, j, k, l, m, n) are all treated as **integers**. Otherwise, if not declared separately, variable names starting with any other letters other than the above, are treated as **real**.

Mathematical Operations:

Addition	$x + y$
Subtraction	$x - y$
Multiplication	$x * y$
Division	x/y
Power	$x ** y$ (x^y)

System Functions:

There are some in-built functions in the system library which can be used readily by calling them anytime inside a program.

abs(x)	|x|
mod(x,y)	The remainder, while x is divided by y.
sqrt(x)	$\sqrt{x}$
sin(x), cos(x), tan(x), cot(x)	Sine, cosine *etc.* of x (in radian)
asin(x), acos(x), atan(x)…	Inverse trigonometric functions.
log(x), exp(x)	Logarithmic and Exponential functions

Do Loop Structure:

```
do n=in, fin, int
    Statement
enddo
```

in -> Initial value, fin -> Final value, int -> interval
For example, do n=1, 100, 2

Logical If Statement:

```
If (Condition) then
    Statement 1
Else
    Statement 2
endif
```

For a single statement: **If(condition) Statement**

Logical Conditions:

.lt.	Less than
.gt.	Greater than
.le.	Less than equal to
.ge.	Greater than equal to
.eq.	Equal to

Dimension:

To store the values of a variable, we declare 'dimension' at the start of the program.
For example, dimension x(10), y(15, 20) etc. in one and dimensional array.

Input/Output (I/O) Statements:

- read(*,*) or print*,
- write(*,*)

In the parenthesis (), the first star corresponds to writing/ reading on or from screen. The 2nd star corresponds to writing/ reading in free format.

If we write numbers in place of stars, for example, read(1, 7), it means the reading is from a file, opened with number 1. The second star indicates writing/ reading style is formatted (that means we design under the number 7, how to write/ read).

NOTE:
We can write loops inside loops, logical statements inside a logical statement. We can write a program for a function and can incorporate inside a main program. All these can be understood following the examples below.
In the following, we will mostly use FORTRAN 77 syntaxes. Once learned, other versions of FORTRAN language can be easily incorporated for clever use.

Simple Programs

Area of a Triangle: Heron's Formula

Theory:

For a triangle, if two sides a and b and the angle θ between them are given, the third side, $c = \sqrt{a^2 + b^2 - 2ab\cos\theta}$. The area of the triangle, $A = \sqrt{s(s-a)(s-b)(s-c)}$, where $s = (a+b+c)/2$.

Algorithm:

1. Input: a, b and θ (in deg.)
2. Covert the angle θ into radian.
3. Find the third side c from formula.
4. Calculate 's'.
5. Calculate area from formula.

FORTRAN Program:

```
C       Area of a Triangle: Heron's Formula
C
        write(*,*)'Give Two sides and the adjacent angle'
        read(*,*)a,b,theta
        x = 3.14*theta/180      ! Converting into radian
        c = sqrt(a**2+b**2-2*a*b*cos(x))
        s =(a+b+c)/2.
        area = sqrt(s*(s-a)*(s-b)*(s-c))
        write(*,*)'Area =', area
        stop
        end
```

Prime Numbers:

Theory:

Prime numbers are those which are not divisible by any number except 1 and the same number. For example, 3, 5, 7…. Here we check prime numbers between two numbers

Algorithm:

1. Do Loop starts from 2 up to n-1
2. Inside the loop, check if the number 'n' is divisible by any number
3. Set a counter to check how many times there are no divisors.
4. If the count = n-2, we find this is a prime number.

FORTRAN Program:

```
C       To find Prime Numbers
C
        write(*,*)'Prime Numbers'
        do 10 n=1,1000      !primes between this range
           do i=2,n-1
              if(mod(n,i).eq.0)go to 10
           enddo
           write(*,*)'Prime = ', n
10      continue
        stop
        end
```

Perfect Square Number:

Theory:

This is to check if a given number is a perfect square, for example, $25 = 5^2$. So we take a number and calculate its square root, the result will be an integer when perfect square, else a real number. If the result is a real number (and not an integer), the integer part squared again, will not match with the original number. In case of perfect square number, this matches.

Algorithm:

1. Input: the number
2. Take square root and keep the integer part.
3. Square the Integer part (in step 2).
4. Check if the squared number (in step 3) matches with the given number.

FORTRAN Program:

```
C     Check if a number is a perfect square
C
      write(*,*)'Give a number'
      read(*,*)n
      nsqrt = sqrt(real(n))
      nsq = nsqrt*nsqrt
      if(nsq.eq.n)then
         write(*,*)'Perfect Square'
      else
         write(*,*)'NOT a Perfect Square'
      endif
      stop
      end
```

TO CHECK LEAP YEAR:

Theory:

Earth takes 365 days, 5 hours, 48 minutes, and 45 seconds – to revolve once around the Sun. But our Gregorian calendar considers 365 days in a year. So a leap day is regularly added (to the shortest month February) in every 4 years to bring it in sync with the Solar year. After 100 years our calendar does have a round off by around 24 days.

In the Gregorian calendar three criteria must be taken into account to identify leap years:

- The year can be evenly divided by 4;
- If the year can be evenly divided by 100, it is NOT a leap year, unless;
- The year is also evenly divisible by 400. Then it is a leap year.

This means that in the Gregorian calendar, the years 2000 and 2400 are leap years, while 1800,1900, 2100, 2200, 2300 and 2500 are NOT leap years.

Algorithm:

1. Input: the number
2. Check if it is not divisible by 100 but divisible by 4
3. Check if it is divisible by 400
4. For steps 2 and 3, the Number is Leap year, otherwise it is not.

FORTRAN Program:

```
C     To Check Leap Year

      integer y
      write(*,*) 'enter year'
      read(*,*) y
      if(mod(y,100).ne.0.and.mod(y,4).eq.0)
      then
          write(*,*) 'leap year'
      elseif(mod(y,400).eq.0) then
          write(*,*) 'leap year'
      else
          write(*,*) 'not a leap year'
      endif
      end
```

Stirling's Formula:

Theory:

Stirling's formula to find out logarithm of factorial of a large number: $\ln n! \approx n \ln n - n$
We shall verify this. But since usually $n!$ would be a very large number for a big number, and so the computer would not be able to handle it and store beyond some limit. Thus we use $\ln n! = \sum_{k=1}^{n} \ln k$. [Here we avoid computing factorial!]

Algorithm:

1. Start a do-loop to sum the logarithm of the integers from 1 up to the given number.
2. Calculate the value from Stirling's formula.
3. Compare the results from (1) and (2).

FORTRAN Program:

```
C      Stirling's Formula
C
       open(1,file='stir.d')
       do n=10,10000,10
          sum=0.0
          do i=1,n
             sum=sum+log(real(i))
          enddo
          app=n*log(real(n))-n
          diff=sum-app
          write(*,*)n,sum,app
          write(1,*)n,diff/sum
       enddo
       stop
       end
```

Maximum and Minimum from a list of Numbers:

Algorithm:

1. Call the first number to be maximum (or minimum).
2. Compare the next number and find the max (or min) between the two. Then redefine the max (or min).
3. Repeat step 2.

FORTRAN Program:

```
C     To find Max and Min numbers from a list.
C
      write(*,*)'How many numbers?'
      read(*,*)n
      write(*,*)'Give the numbers'
      read(*,*)a              !First number
      amax=a
      amin=a
      do i=2,n
         read(*,*)a
         if(a.ge.amax)amax=a
         if(a.le.amin)amin=a
      enddo
      write(*,*)'Max = ', amax, 'Min = ', amin
      stop
      end
```

Note: To store the numbers through 'dimension':

```
C     To find Max, Min and the Range.
C
      dimension x(100)
      write(*,*)'How many numbers?'
      read(*,*)n
      write(*,*)'Give the numbers'
      read(*,*)(x(i),i=1,n)      !Implied do-loop
      amax=x(1)
      do i=1,n
         if(x(i).gt.amax)amax=x(i)
      enddo
C
      amin=x(1)
      do i=1,n
         if(x(i).lt.amin)amin=x(i)
      enddo
C
      range=amax-amin
      write(*,*)'Max, Min, Range'
      write(*,*)amax, amin, range
      stop
      end
```

Sorting of Numbers from a List: (Ascending and Descending)

Theory:

The task is to read from a list of numbers and compare between first two numbers and decide which is bigger (or smaller) and then compare with the next and so on. To do this we have to store the one number into a temporary variable and then orient the two.

Algorithm:

1. Read the numbers from a list
2. Compare the first two numbers. If the 1st one is greater than the 2nd, store the 2nd number into a temporary variable.
3. Now call the 2nd number equal to 1st number.
4. Assign the 1st number equal to the stored number (previous 2nd number).
5. Repeat 1-4 until the end of the list. We get the ascending order.

FORTRAN Program:

```
C       Ascending Order
C
        dimension a(100)
        write(*,*)'How many numbers?'
        read(*,*)n
        write(*,*)'Give the numbers'
        read(*,*)(a(i),i=1,n)
        do i=1,n
           do j=i+1, n
              if(a(i)> a(j))then
                 temp=a(j)
                 a(j)=a(i)
                 a(i)=temp
              endif
           enddo
        enddo
        write(*,*)'In ascending Order'
        write(*,*)(a(i),i=1,n)
        stop
        end
```

*For **descending** order, you just have to change '>' (greater than) in the **if**-statement to '<' (less than).

Digits of a decimal number:

Theory:

We have to read a number and find out its digits at different places. A decimal number is found by multiplying by the powers of 10 with the digit in place and then summing. So the digits of a decimal number can be found out by dividing the number by 10 and taking the remainder. Next time, divide the number by 10 and take the integer part and repeat the steps. For example, take the number 358, divide 358 by 10, the remainder is 8. Next time, consider $\frac{358}{10} = 35.8$ and take the integer part which is 35 and repeat the steps.

Algorithm:

1. Input: The Number, number of digits (n)
2. Start a do loop starting from 1 to n.
3. Find the remainder (Modulo 10).
4. Divide the number by 10 and retain the integer part for the next step.
5. Sum the digits as obtained.

FORTRAN Program:

```
C       To find the digits and their sum
C
        write(*,*)'Give the number'
        read(*,*)n
        nsum=0
        write(*,*)'Digits in reverse order'
        nsum=0
10      k=mod(n,10)
        n=n/10.
        nsum=nsum+k
        write(*,*)k
        if(n.gt.9)go to 10
        write(*,*)n
        write(*,*)'Sum of digits = ', nsum
        stop
        end
```

Factorial of a Number:

Theory: Factorial of a number, $n! = n \times (n-1) \ldots \times 2 \times 1$.

Algorithm:

1. Start a do loop for taking the product of numbers in increasing (or decreasing order)
2. Write the product which is the factorial of the given number

FORTRAN Program:

```
C       Factorial of a Number
        write(*,*)'Give the number'
        read(*,*)n
        nfac = 1
        do i=2,n
           nfac = nfac*i
        enddo
        write(*,*)'Factorial = ', nfac
        stop
        end
```

Strong Number:

Theory:

A strong number is something where the sum of factorials of its digits equals to the number itself. For example: $145 = 1! + 4! + 5!$ We here find out strong numbers between 1 and some number.

Algorithm:

1. Start a do-loop of numbers from 1 to some big number
2. The number variable is renamed when used.
3. Find each digit of the number and find factorial of that digit.
4. Sum the factorial number
5. Compare the sum with the original number

[Note: Here we have defined a function subroutine to calculate factorial as that has to be done repeatedly.]

FORTRAN Program:

```
C     Strong Number
C
      do k=1,100000
         n=k
         nsum=0
10       i=mod(n,10)
         nsum=nsum+nfac(i)
         n=n/10.
         if(n.gt.9)go to 10
         nsum=nsum+nfac(n)
         if(nsum.eq.k)write(*,*)'Strong Number ', k
      enddo
      stop
      end
C
C     Subroutine
C
      function nfac(i)
      nfac=1
      do k=2,i
         nfac=nfac*k
      enddo
      return
      end
```

*In the above program, we have defined a function in a 'subroutine' for our purpose. This is just to make our program simpler to handle.

Armstrong Number:

Theory:

Armstrong number is a number such that the sum of its digits raised to the third power is equal to the number itself. For example, 153 is an Armstrong number, since $1^3 + 5^3 + 3^3 = 153$.

Algorithm:

1. Set a counter.
2. For a three digit number, three nested do loop start for three integers.
3. Construct a number from three integers.
4. Take cubes of the integers and sum them inside the innermost loop.

5. Check if the sum matches with the constructed number.
6. Count how many Armstrong numbers are there between 0 and 999.

FORTRAN Program:

```
C       Armstrong Numbers between 0 and 999
C
        n=0
        do i=0,9
           do j=0,9
              do k=0,9
                 ijk=i*100+j*10+k
                 i3j3k3=i**3+j**3+k**3
                 if(ijk.eq.i3j3k3)then
                    write(*,*)ijk
                    n=n+1
                 endif
               enddo
            enddo
         enddo
         write(*,*)'Total Armstrong Numbers = ', n
         stop
         end
```

```
C       Armstrong Numbers between two Numbers
C
        Integer a, b
        write(*,*)'Give a and b'
        read(*,*)a,b
        do n=a,b
           k=n
           nsum=0
10         i=mod(k,10)
           k=k/10.
           nsum=nsum+i**3
           if(k.gt.9)go to 10
           nsum=nsum+k**3
           if(nsum.eq.n)'Armstrong Number ', n
        enddo
        stop
        end
```

Palindrome Number:

Theory:

A Palindrome is some word or number or a phrase which when read from the opposite end, appears the same (reflection symmetry, example, 34543, symmetry around 5 in the middle). We here read a decimal number and find out its digits at different places. After finding the digits we construct the number taking the digits in the reverse order. For example, given the number 456, we construct the new number $6 \times 10^2 + 5 \times 10^1 + 4 \times 10^0 = 654$ (this is not palindrome number).

Algorithm:

1. Input: The Number, number of digits (n)
2. Store the Number.
3. Start a do loop starting from 1 to n.
4. Find the remainder (Modulo 10).
5. Divide the number by 10 and retain the integer part for the next step.
6. Do the sum after multiplying the digit with the appropriate power of 10 for that place.
7. Check the sum with the stored number: If equal, Palindrome.

FORTRAN Program:

```
C       Palindrome Number
C
        write(*,*)'Give the number'
        read(*,*)num
        write(*,*)'Number of digits = ?'
        read(*,*)n
        num0=num              ! To store the Number
        nsum=0
        do i=1,n
           k=mod(num,10)
           num=num/10.
           nsum=nsum+nd*10**(n-i)
           write(*,*)nd
        endd
        if(nsum.eq.num0)write(*,*)'Palindrome'
        if(nsum.ne.num0)write(*,*)'Not Palindrome'
        stop
        end
```

Fibonacci Numbers:

Theory:

Fibonacci numbers are the series of numbers where each number is the sum of two previous numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89….

The rule: $a_{n+2} = a_n + a_{n+1}$

Algorithm: (Numbers up to 610, for example)

1. Input: Two initial numbers 0 and 1.
2. Start a do-loop. Obtain the 3rd number as the sum of first two numbers.
3. Now redefine the 2nd number as the 1st number and the 3rd number as the 2nd number.
4. Obtain the 3rd number as the sum of newly defined 1st and 2nd numbers and repeat inside the loop. Write down the numbers on the screen as they are obtained.
5. Check if the new number in the series is 610, then exist the loop and write the sum.

FORTRAN Program:

```
C     Fibonacci Numbers up to 610 (or any no. you like)
C
      n1=0
      n2=1
      n3=n2+n1
      write(*,*)'Fibonacci Numbers'
      write(*,*)n1
      write(*,*)n2
      write(*,*)n3
      do while(n3.lt.610)
         n1=n2
         n2=n3
         n3=n2+n1
         write(*,*)n3
      enddo
      write(*,*)'Golden Ratio= ', float(n3)/n2
      stop
      end
```

Golden Ratio = 1.6180…

GCD and LCM:

Theory:

This is to find Greatest Common Divisor (GCD) and Lowest Common Multiplier (LCM) of two positive integers by Euclid method. First, we divide the bigger number by the smaller number and consider the residue. If the residue is zero, then the smaller number is the GCD. If this is not, then we take the nonzero residue as the smaller number and previous smaller number to be the bigger number. This is continued until we get a zero residue. Then at the last step, the divisor is called the GCD. LCM is just obtained by dividing the product of two original numbers by GCD.

Algorithm:

1. Arrange the two numbers: bigger number = m, smaller number = n
2. Find residue by dividing "m" by "n" (through system function)
3. If the residue in 2 is zero, GCD = n
4. If the residue in 2 is not zero, redefine: m = n, n= residue
5. Repeat the steps 2 - 4.
6. Calculate LCM = m*n/GCD

```
C      GCD & LCM
C
       write(*,*)'Give m, n'
       read(*,*)m,n
       nprod=m*n
       if(m.lt.n)then
          k=m
          m=n
          n=k
       endif
10     i=mod(m,n)
       if(i.eq.0)then
          write(*,*)'GCD = ', n
       else
          m=n
          n=i
          go to 10
       endif
       write(*,*)'LCM = ', nprod/n
       stop
       end
```

Sum of an Infinite Series

Exponential Series:

Theory:

Taylor series expansion:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots\ldots\ldots\ldots$$

Algorithm:

1. Input: x, Tolerance value
2. Initial values: Sum = 0, Term = 1.0
3. Do Loop starts (n = 1, 2…)
4. New Term = Previous Term $\times \frac{x}{n}$
5. Sum the terms.
6. Check: Absolute value of a term when less than tolerance, come out.
7. Write the sum.

FORTRAN Program:

```
C     Exponential Series with tolerance value
C
      write(*,*)'Give x and tolerance value'
      read(*,*)x,tol
      sum=0.0
      term=1
      do n=1,100
         if(abs(term).lt.tol)go to 10
         sum=sum+term
         term=term*x/n
      end do
10    write(*,*)'Calculated value, Actual value'
      write(*,*)sum,exp(x)
      stop
      end
```

Cosine Series:

Theory:

Taylor series expansion:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Algorithm:

1. Input: x, tolerance value
2. Initial values: Sum = 0, Term $= 1.0$
3. Do Loop starts (n = 1, 2….)
4. New Term = Previous term$\times \left(-\frac{x^2}{2n(2n-1)}\right)$
5. Sum the terms
6. Check: Absolute value of a term when less than tolerance, come out.
7. Write the sum.

FORTRAN Program:

```
C     Sum of Cosine Series with tolerance value
C
      write(*,*)'Give x and tolerence value'
      read(*,*)x,tol
      sum=0.0
      term=1.0
      do n=1,100
         if(abs(term).lt.tol)go to 10
         sum=sum+term
         term=term*(-x**2/(2*n*(2*n-1)))
      enddo
10    write(*,*)'Calculated value, Actual Value'
      write(*,*)sum,cos(x)
      stop
      end
```

Sine Series:

Theory:

Taylor series expansion of:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{7}{7!} + \cdots$$

Algorithm:

1. Input: x, tolerance value
2. Initial values: Sum = 0, Term $= x$
3. Do Loop starts (n = 1, 2….)
4. New Term = Previous term$\times \left(-\frac{x^2}{2n(2n+1)}\right)$
5. Sum the terms
6. Check: Absolute value of a term when less than tolerance, come out.
7. Write the sum

FORTRAN Program:

```
C     Sum of Sine Series with tolerance value
C
      write(*,*)'Give x and tolerence value'
      read(*,*)x,tol
      sum=0.0
      term=x
      do n=1,100
         if(abs(term).lt.tol)go to 10
         sum=sum+term
         term=term*(-x**2/(2*n*(2*n+1)))
      enddo
10    write(*,*)'Calculated value, Actual Value'
      write(*,*)sum,sin(x)
      stop
      end
```

Conversion of Numbers

From Decimal to Binary: (Integer)

Theory:

Base 10 system -> Base 2 system.

Continue downwards, dividing each new quotient by two and writing the remainders to the right of each dividend. Stop when the quotient is 1.

For example, a decimal number, 156 (may be written as 156_{10}) is converted to a binary number 10011100 (may be written as 10011100_2)

2)156 0
2)78 0
2)39 1
2)19 1
2)9 1
2)4 0
2)2 0
1

Algorithm: (For Integer Numbers)

1. Input: a number (integer)
2. Divide by 2 (Modulo)
3. Store the remainder in memory.
4. Take the Quotient for the next step.
5. Repeat steps 2 & 3 until the Quotient is 1 or 0.
6. When the Quotient is 1 0r 0, store this as remainder.
7. Write the stored remainders (either 0 or 1) in reverse order

FORTRAN Program:

```
C       From Decimal to Binary
C
        dimension ib(100)
        write(*,*)'Give a Decimal Number'
        read(*,*)n
        do i=1,100
           if(n.le.1)then
              ib(i)= n
              go to 10
           endif
           ib(i)= mod(n,2)
           n=n/2
        enddo
10      write(*,*)'The Binary Number'
        write(*,*)(ib(k),k=i,1,-1)   !Implied do-loop
        stop
        end
```

Decimal to Binary (For Real Numbers, fractional part considered):

Theory:

For the conversion of fractional part, we have to multiply by 2 and collect the real part of the product.

Algorithm:

1. Input: a number (real number)
2. Take the **integer** part. Divide by 2
3. Store the remainder in memory.
4. Take the Quotient for the next step.
5. Repeat steps 3 & 4 until the Quotient is 1 or 0.
6. When the Quotient is 1 or 0, store this as remainder.
7. Write the stored remainders (either 0 or 1) in reverse order.
8. Now take the **fractional part**, multiply by 2, store the integer part of real number obtained.
9. Subtract the above integer from the new real number and obtain a new fraction.
10. Repeat the steps from 8-10 until the fraction becomes 0 or stop after some steps for recurring fraction.
11. Write the integers

```
C       From Decimal to Binary
C
        dimension ib(100), ibc(100)
        write(*,*)'Give a Decimal Number'
        read(*,*)x
        intx=x                ! Integer part
        fracx=x-intx          ! Fractional part
        do i=1,1000           ! Loop for Integer part
           if(intx.le.1)then
             ib(i)=intx
             go to 10
           endif
           ib(i)= mod(intx,2)
           intx=intx/2
        enddo
10      do j=1,1000           ! Loop for Fractional part
           fracx=fracx*2
           ibc(j)=fracx
           fracx=fracx-ibc(j)
           if(fracx.eq.0.0.or.j.eq.8)go to 20
        enddo
20      write(*,*)'Binary Number = ',
                  (ib(k),k=i,1,-1),'.',(ibc(k),k=1,j)
        stop
        end
```

From Decimal to Octal:

Theory:

Base 10 system -> Base 8 system.

We have to divide by 8 and we stop when the quotient is less than 8.

The **octal** (*base 8*) numeral system has 7 possible values, represented as 0, 1, 2, 3, 4, 5, 6 and 7 for each place-value.

Continue downwards, dividing each new quotient by 8 and store the remainders**.** Stop when the quotient is less than 8. For example, a decimal number, 156 (may be written as 156_{10}) is converted to a Octal number 234 (may be written as 234_8)

8)156 4
8)19 3
2

Algorithm: (For Integer Numbers)

1. Input: a number (integer)
2. Divide by 8 (Modulo)
3. Store the remainder in memory.
4. Take the Quotient for the next step.
5. Repeat steps 2 & 3 until the Quotient is less than 8.
6. When the Quotient less than 8, store this as remainder.
7. Write the stored remainders (between 0 & 7) in reverse order

FORTRAN Program:

```
C       From Decimal to Octal
C
        dimension ib(100)
        write(*,*)'Give a Decimal Number'
        read(*,*)n
        do i=1,100
           if(n.le.7)then
              ib(i)=n
              go to 10
           endif
           ib(i)= mod(n,8)
           n=n/8
        enddo
10      write(*,*)'The Octal Number'
        write(*,*)(ib(k),k=i,1,-1)  ! Implied do-loop
        stop
        end
```

Decimal to Octal (For Real Numbers, fractional part considered):

Theory:

For the conversion of fractional part, we have to multiply by 2 and collect the real part of the product.

Algorithm:

1. Input: a number (real number)
2. Take the **integer** part. Divide by 8
3. Store the remainder in memory.
4. Take the Quotient for the next step.
5. Repeat steps 3 & 4 until the Quotient is less than 8.
6. When the Quotient is less than 8, store this as remainder.
7. Write the stored remainders (between 0 and 7) in reverse order
8. Now take the **fractional part**, multiply by 8, store the integer part real number obtained.
9. Subtract the above integer from the new real number and obtain a new fraction.
10. Repeat the steps from 8-10 until the fraction becomes 0 or stop after some steps if recurring.
11. Write the stored integers

```
C       From Decimal to Octal
C
        Dimension io(100), ioc(100)
        write(*,*)'Give a Decimal Number'
        read(*,*)x
        intx=x                    ! Integer part
        fracx=x-intx              ! Fractional part
        do i=1,1000               ! Loop for integer part
           if(intx.le.7)then
           io(i)=intx
             go to 30
           endif
           io(i)=mod(intx,8)
           intx=intx/8
        enddo
30      do j=1,1000               ! Loop for fractional part
           fracx=fracx*8
           ioc(j)=fracx
           fracx=fracx-ioc(j)
           if(fracx.eq.0.0.or.j.ge.8)go to 40
        enddo
40      write(*,*)'Octal Number = ',
                  (io(k),k=i,1,-1),'.',(ioc(k),k=1,j)
        stop
        end
```

From Binary to Decimal:

Theory:

To convert a binary number: List the powers of 2 from right to left. Start at 2^0, next increase the exponent by 1 for each power. Stop when the amount of elements in the list is equal to the amount of digits in the binary number. For example, the number 10011 has 5 digits, so the corresponding decimal number = $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19$

Algorithm:

1. Input: Binary number, total number of digits
2. Define the binary number as a 'character' so as to read each digit in it.
3. Find each digit of the number (either 0 or 1) and its position (n) counted from right to left.
4. The digit (0 or 1) found from step 3 is multiplied by 2^{n-1}
5. Obtain the sum in a do-loop.
6. Write the sum which is the decimal number.

FORTRAN Program:

```
C
C     From Binary to Decimal
C
      write(*,*)'Give a Binary Number'
      read(*,*)number
      write(*,*)'Total Number of Digits?'
      read(*,*)n
      nsum=0
      do i=1,n
         nd=mod(number,10)
         number=number/10.
         nsum=nsum+nd*2**(i-1)
      enddo
      write(*,*)'Decimal Number = ',nsum
      stop
      end
```

From Octal to Decimal:

Theory:

To convert a octal number: List the powers of 8 from right to left. Start at 8^0, next increase the exponent by 1 for each power. Stop when the amount of elements in the list is equal to the amount of digits in the binary number. For example, the number 234 has 3 digits, so the corresponding decimal number = $2 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$ = 156

Algorithm:

1. Input: Octal number, total number of digits
2. Define the Octal number as a 'character' so as to read each digit in it.
3. Find each digit of the number (0, 1, 2, 3, 4, 5, 6 or 7) and its position (n) counted from right to left.
4. The digit (0 to 7) found from step 3 is multiplied by 8^{n-1}
5. Obtain the sum in a do-loop.
6. Write the sum which is the decimal number.

FORTRAN Program:

```
C
C     From Octal to Decimal
C
      write(*,*)'Give an Octal Number'
      read(*,*)number
      write(*,*)'Total Number of Digits?'
      read(*,*)n
      nsum=0
      do i=1,n
         nd=mod(number,10)
         number=number/10.
         nsum=nsum+nd*8**(i-1)
      enddo
      write(*,*)'Decimal Number = ',nsum
      stop
      end
```

Matrices

Addition of Two Matrices:

Theory:

For the addition of two matrices, the rows and columns of the two matrices have to be equal. The (i,j)th element of a matrix is added with the same (i,j)th element of another to obtain the (i,j)th element of the new matrix: $C_{ij} = A_{ij} + B_{ij}$.

Algorithm:

1. Input and output Matrix elements are stored in memory.
2. Input: Dimension of Matrices, Matrix elements
3. Do loop for two indices, i and j, loops are nested.
4. The (i,j)-th element of a matrix is added with the same (i,j)th element of another to obtain the (i,j)th element of the new matrix: $C_{ij} = A_{ij} + B_{ij}$.
5. Write down the matrix elements with implied do loop.

FORTRAN Program:

```
C     Addition of Two Matrices
C
      dimension a(10,10),b(10,10),c(10,10)
      write(*,*)'Rows & Columns of Matrices'
      read(*,*)m,n
      write(*,*)'Matrix elements of A?'
      do i=1,m
         read(*,*)(a(i,j),j=1,n)
      enddo
      write(*,*)'Matrix elements of B?'
      do i=1,m
         read(*,*)(b(i,j),j=1,n)
      enddo
      do i=1,m
      do j=1,n
         c(i,j)= a(i,j)+b(i,j)
      enddo
      enddo
      Write(*,*)'The added Matrix'
      do i=1,m
         write(*,*)(c(i,j),j=1,n)
      enddo
      stop
      end
```

Product of Two Matrices:

Theory:

$C_{ij} = (AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}$, for the Product of two matrices, the number of columns of the first matrix has to be the same as the number of rows of the 2nd matrix.

Algorithm:

1. Input and output Matrix elements are stored in memory.
2. Input: Dimensions of Matrices, Matrix elements
3. Do loop for three indices: i, j, k, loops are nested
4. Inside the k-loop, the (i,k)th element of 1st matrix is multiplied with the (k,j)th element of 2nd matrix, and then sum is taken.
5. Write down the matrix elements with implied do loop.

FORTRAN Program:

```
C     Product of Two Matrices
C
      dimension a(10,10),b(10,10),c(10,10)
      write(*,*)'Rows & Columns of Matrix A'
      read(*,*)m,l
      write(*,*)'Matrix elements of A?'
      do i=1,m
         read(*,*)(a(i,j),j=1,l)
      enddo
      write(*,*)'Rows & Columns of Matrix B'
      read(*,*)l,n
      write(*,*)'Matrix elements of B?'
      do i=1,l
         read(*,*)(b(i,j),j=1,n)
      enddo
      do i=1,m
      do j=1,n
         c(i,j)= 0.0
         do k=1,l
            c(i,j)= c(i,j)+a(i,k)*b(k,j)
         enddo
      enddo
      enddo
      Write(*,*)'The Product Matrix'
      Do i=1,m
         write(*,*)(c(i,j),j=1,n)
      enddo
      stop
      end
```

Transpose of a Matrix:

Theory:

For the transpose of a matrix, the rows of the original matrix become the columns of the transposed matrix and vice versa. So what we do is that we read the matrix elements and find the elements of the transposed matrix: $b_{ij} = a_{ji}$

Algorithm:

1. Matrix elements are stored in memory
2. Input: rows and columns, matrix elements
3. Do loop for i and j: $b_{ij} = a_{ji}$
4. Transpose matrix elements written in implied do loop

FORTRAN Program:

```
C     Transpose of a Matrix
C
      dimension a(10,10),b(10,10)
      write(*,*)'No. of rows and columns?'
      read(*,*)m,n
      write(*,*)'Matrix elements?'
      do i=1,m
         read(*,*)(a(i,j),j=1,n)
      enddo
C
      do i=1,n
      do j=1,m
         b(i,j)=a(j,i)
      enddo
      enddo
C
      Write(*,*)'The Transposed Matrix'
      Do i=1,n
         write(*,*)(b(i,j),j=1,m)
      enddo
      stop
      end
```

Numerical Analysis

Integration by Simpson's 1/3rd Rule:

Theory:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$ [See APPENDIX I for the detailed theory and calculations.]

Algorithm:

1. The function $f(x)$ is defined in a subroutine.
2. Input: Lower limit (a), Upper limit (b)
3. Calculate h: $h = \frac{b-a}{2}$
4. Evaluate: $f(a)$, $f(b)$, $f(\frac{a+b}{2})$
5. Calculate the sum: $f(a) + 4 \times f\left(\frac{a+b}{2}\right) + f(b)$
6. Multiply step 5 by $h/3$

FORTRAN Program:

```
C      Simpson's 1/3 Rule
C
       write(*,*)'Lower limit, Upper Limit'
       read(*,*)a,b
       h=(b-a)*0.5
       x=(a+b)*0.5
       sum=f(a)+4*f(x)+f(b)
       s=h/3*sum
       write(*,*)'Value of Integral= ', s
       stop
       end
C
C      Subroutine for function
       function f(x)
       f=x**2
       return
       end
```

Integration by Simpson's 1/3rd Rule (Modified):

Theory: $\int_a^b f(x)dx$

$$= \frac{h}{3}\left[f(x_0) + 4\{f(x_1) + f(x_3) + \ldots + f(x_{n-1})\} + 2\{f(x_2) + f(x_4) + \ldots + f(x_{n-2})\} + f(x_n) \right]$$

Algorithm:

1. The function $f(x)$ is defined in a subroutine.
2. Input: Lower limit (a), Upper limit (b), sub-intervals (n).
3. Calculate: $h = \frac{b-a}{n}$
4. Evaluate: $f(a) + f(b)$
5. Do Loop over n starts. Evaluate the function: for every odd sub-interval multiply by 4, for every even sub-interval multiply by 2 and then add all of them.
6. Finally, multiply the sum by $h/3$

FORTRAN Program:

```
C      Simpson's 1/3 Rule (Modified)
C
       write(*,*)'Lower limit, Upper Limit,sub-intervals'
       read(*,*)a,b,n
       h=(b-a)/n
       sum=f(a)+f(b)
       d=4
       do k=1,n-1
          x=a+k*h
          sum=sum+d*f(x)
          d=6-d
       enddo
       s=h/3*sum
       write(*,*)'Value of Integral= ', s
       stop
       end
C
C      Subroutine for function
C
       function f(x)
       f=x**2
       return
       end
```

Least Square Method: Linear Fit

Theory: Slope = $\frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$, Intercept = $\frac{\sum x^2 \sum y - \sum x \sum xy}{n\sum x^2 - (\sum x)^2}$

[See APPENDIX I for the detailed theory and calculations.]

Algorithm:

1. Input: x, y
2. Inside the Do Loop, run the sums to evaluate: $\sum x, \sum y, \sum x^2, \sum xy$
3. Calculation of slope and intercept
4. Few data points generated with obtained slope and intercept, the straight line drawn is superposed over the original data points.

```
C     Least Square fit
C
      open(1,file='xy.dat')
      open(2,file='fit.dat')
      write(*,*)'Number of Points?'
      read(*,*)n
      sumx=0.0
      sumy=0.0
      sumsqx=0.0
      sumxy=0.0
      write(*,*)'Give data in the form: x,y'
      do i=1,n
         read(*,*)x,y
         write(1,*)x,y
         sumx=sumx+x
         sumy=sumy+y
         sumsqx=sumsqx+x*x
         sumxy=sumxy+x*y
      enddo
      deno=n*sumsqx-sumx*sumx
      slope=(n*sumxy-sumx*sumy)/deno
      b=(sumsqx*sumy-sumx*sumxy)/deno
      write(*,*)'Slope, Intercept= ',slope,b
C
      write(*,*)'Give lower and upper limits of X'
      read(*,*)xmin, xmax
      x=xmin
      dx=(xmax-xmin)/2.0
      do i=1,3
         y=slope*x+b
         write(2,*)x,y
         x=x+dx
      enddo
      stop
      end
```

Note: The following programs are based on solving simple first order differential equations, basically by Euler's scheme. Runge-Kutta method of different orders can be used when necessary. But we avoided that here for this set of elementary programs.

Projectile Motion

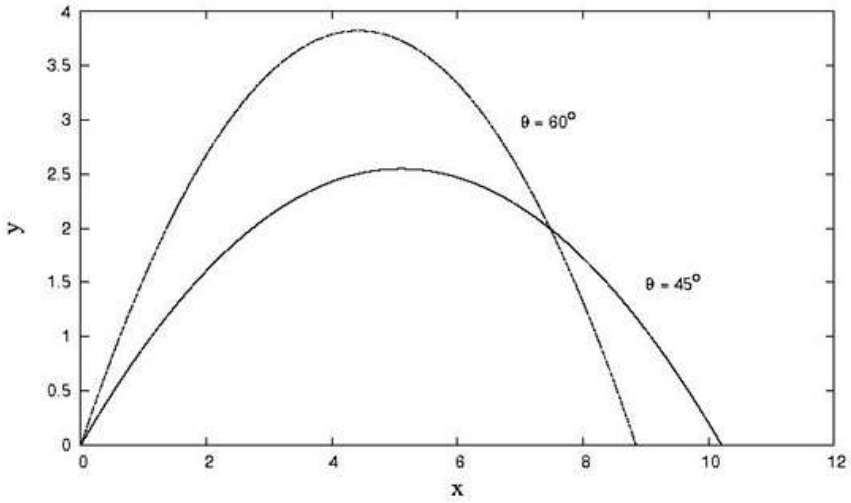
Theory:

Consider a particle projected at an angle θ from origin $(0, 0)$, with the horizontal direction being x-axis and the vertical direction y-axis. The force on the particle, $\vec{F} = m\vec{g}$. Components: $F_x = m\frac{dv_x}{dt} = 0, F_y = m\frac{dv_y}{dt} = -g$. For the purpose of computation, we take $g = 980$; Mass, $m = 1$. Projection angle, θ and initial speed v_0 are given. All the quantities are in arbitrary units.

Algorithm:

1. Input: initial velocity v_0, angle of projection θ. The interval of time is set.
2. Initial values: interval of time dt, Force components $F_x = 0, F_y = -g$
3. Components of initial velocity: $v_x = v_0 \cos\theta$, $v_y = v_0 \sin\theta$
4. Change in velocity components: $dv_x = F_x.dt = 0$, $dv_y = F_y.dt = -gdt$
5. New velocity components: $v_x = v_x + dv_x$, $v_y = v_y + dv_y$
6. Change in position coordinates: $dx = v_x.dt$, $dy = v_y.dt$
7. New position coordinates: $x = x + dx$, $y = y + dy$
8. Write the position coordinates (x, y) in a file and then plot to see.

The trajectory of the particle projected at an angle (indicated in the figs.) and with some initial velocity



FORTRAN Program:

```
C     Projectile Motion of a particle
C
      open(1,file='proj.dat')
      write(*,*)'Angle (in degree) and initial speed?'
      read(*,*)theta,v0
      dt=0.01
      g=980
      theta=3.14/180*theta
      fx=0
      fy=-g
      vx=v0*cos(theta)
      vy=v0*sin(theta)
      x=0
      y=0
      do while(y.gt.0.0)
         dvx=fx*dt
         dvy=fy*dt
         vx=vx+dvx
         vy=vy+dvy
         dx=vx*dt
         dy=vy*dt
         x=x+dx
         y=y+dy
         write(1,*)x,y
      enddo
      stop
      end
```

Motion under Central Force

Theory:

Consider the Central Force, $F = -k/r^2$ and Newton's 2nd law of motion: $= m\frac{dv}{dt}$. For computational purpose, we take $k = 1, m = 1$. Thus we have, $\frac{dv}{dt} = -\frac{1}{r^2}$. We solve for v and then therefrom derive the position. Initial values for the velocity components, v_x and v_y are provided. We choose the time interval (dt) appropriately. All the quantities are in arbitrary units.

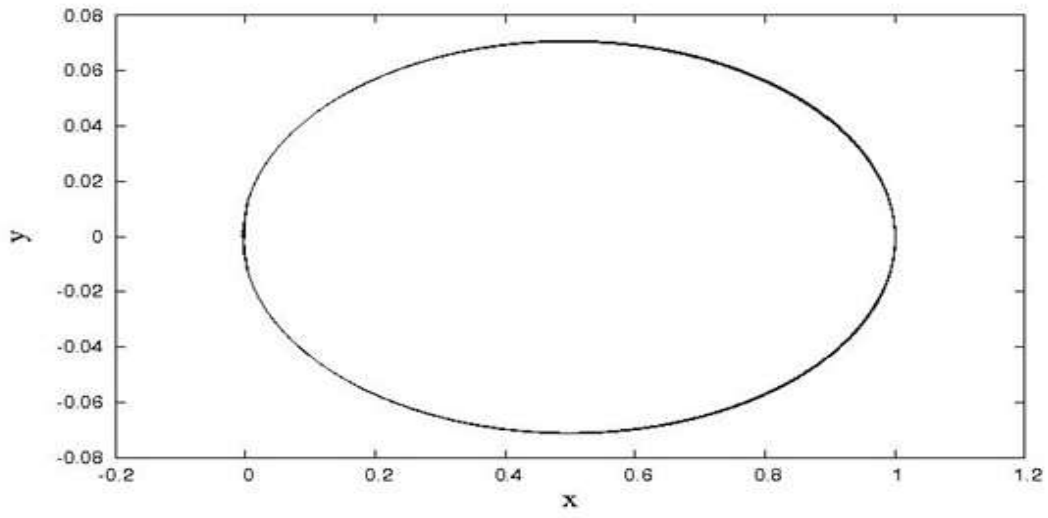
Algorithm:

1. Initial position coordinates x, y are given. Initial velocity components v_x and v_y are given.
2. Time interval dt is chosen.
3. Components of Force: $F_x = -\frac{x}{r}.\frac{1}{r^2},\ F_y = -\frac{y}{r}.\frac{1}{r^2}$
4. $r^2 = x^2 + y^2 \Rightarrow r = \sqrt{r^2}$
5. Change in velocity components: $dv_x = F_x.dt = -\frac{x}{r}.\frac{1}{r^2}.dt, dv_y = F_y.dt = -\frac{x}{r}.\frac{1}{r^2}.dt$
6. New velocity components: $v_x = v_x + dv_x, v_y = v_y + dv_y$
7. New position coordinates: $x = x + v_x.dt, y = y + v_y.dt$
8. Write the position coordinates (x, y) in a file and then plot to see.

FORTRAN Program:

```
C     Motion under a Central Force
C
      open(1,file='central.dat')
      dt=0.01
      vx=0.0
      vy=1.0
      x=1.0
      y=0.0
      ncount=0
      do i=1,10000
        r2=x*x+y*y
        r=sqrt(r2)
        f=-1/r2
        fx=x/r*f
        fy=y/r*f
        vx=vx+fx*dt
        vy=vy+fy*dt
        x=x+dt*vx
        y=y+dt*vy
        n=mod(i,100)
        if(n.eq.0)write(1,*)x,y
      enddo
      stop
      end
```

The Elliptical path of a particle under Central Force



Harmonic Oscillator

Theory:

Consider the Oscillator equation in the form: $m\frac{dv}{dt} = -kx$. For computational purpose, we choose mass $m = 1$ and the force constant $k =$ a positive number (all in arbitrary units). The equation is solved to obtain the velocity v and therefrom we obtain the position of the particle.

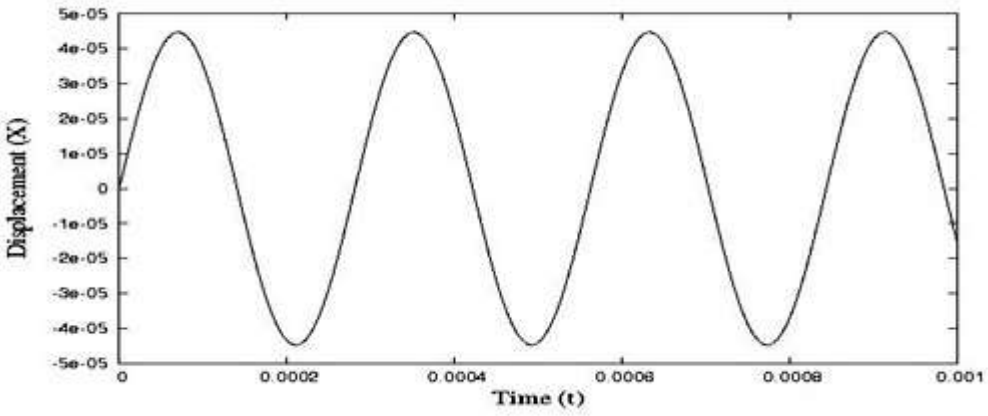
Algorithm:

1. Give the initial position (x), initial time (t) and velocity (v). Set the time interval dt.
2. Value of spring constant k is chosen appropriately, usually a small positive number.
3. $\frac{dv}{dt} = -kx \quad \Rightarrow dv = -k.x.dt$
4. Updating of velocity: $v = v + dv$
5. Change in position: $dx = v.dt$
6. Updating of position: $x = x + dx$
7. Updating of time: $t = t + dt$
8. Write the time and position $(t,\ x)$ in a file and then plot to see.

FORTRAN Program:

```
C       Harmonic Oscillator
C
        open(1,file='harmonic.dat')
        x=0.0
        t=0
        v=1.0
        dt=0.01
        k=5
        do i=1,1000
          dv=-k*x*dt
          v=v+dv
          dx=v*dt
          x=x+dx
          t=t+dt
          write(1,*)t,x,v
        enddo
        stop
        end
```

The Sine Curve obtained from the solution of Linear Harmonic Oscillator Equation.



**Equation for SHM [$m\frac{dv}{dt} = -kx$] is solved numerically in the above with the simple Euler scheme. One can add a damping term (proportional to velocity) and tune the parameters to obtain well known results and phase diagrams. However, it would be interesting to play with the parameters and gain some insight through such numeric. See APPENDIX II.

Fourth Order Runge-Kutta (RK4) Method:

For the first order differential equation, $\frac{dy}{dx} = f(x,y)$ with the initial value at (x_n, y_n), the following steps are performed in order to find the value at the next grid point.

For the grid point x_{n+1}, we compute

$k_1 = hf(x_n, y_n)$

$k_2 = hf(x_n + \frac{h}{2}, y_n + k_1/2)$

$k_3 = hf(x_n + \frac{h}{2}, y_n + h/2)$

$k_4 = hf(x_n + h, y_n + h)$

Then the value of y_{n+1} is given by $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$.

RK4 for Van der Pole Oscillator:

$$\ddot{x} + \epsilon(x^2 - 1)\dot{x} + x = 0$$

$\frac{dx}{dt} = y = f(t,x,y)$

$\frac{dy}{dt} = -x - \epsilon(x^2 - 1)\dot{x} = g(x,y,t)$

Fortran Program:

```
open(1,file='shmrk4.dat')
C
      t1=0
      x1=0
      y1=0.1
      h=0.01
C
      write(1,*)t1,x1,y2
      do i=1,10000
         a1=f(t1,x1,y1)
         b1=g(t1,x1,y1)
         a2=f(t1+0.5*h,x1+0.5*h*a1,y1+0.5*h*b1)
         b2=g(t1+0.5*h,x1+0.5*h*a1,y1+0.5*h*b1)
         a3=f(t1+0.5*h,x1+0.5*h*a2+y1+0.5*h*b2)
         b3=g(t1+0.5*h,x1+0.5*h*a2,y1+0.5*h*b2)
         a4=f(t1+h,x1+h*a3+y1+h*b3)
         b4=g(t1+h,x1+h*a3,y1+h*b3)
C
```

```
          x2=x1+h/6*(a1+2*a2+2*a3+a4)
          y2=y1+h/6*(b1+2*b2+2*b3+b4)
          t2=t1+h
          write(1,*)x2,y2,t2
           x1=x2
           y1=y2
           t1=t2
        enddo
c
        stop
        end
C
        function f(t,x,y)
        f=y
        return
        end
 C
        function g(t,x,y)
        eps=10
        g=-x-eps*(x**2-1)*y
        return
        end
```

RK4 for Chaos Theory: Lorenz Equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

The Values of the parameters, Lorez used, $\sigma = 10,\ \rho = 28$ and $\beta = 8/3$

We can solve this set of coupled 1st order equations by RK4 method.

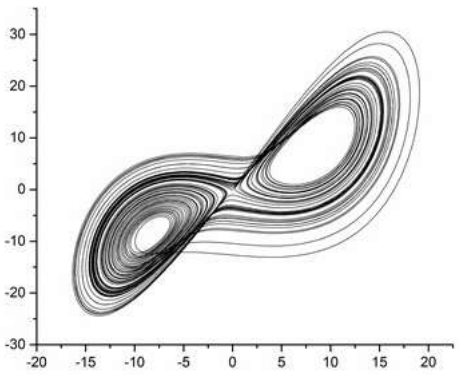
FORTRAN Prog for RK4 Method:

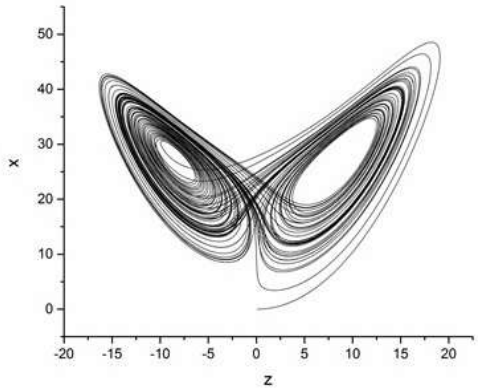
```
C     LORENZ EQUATIONS by RK4
      implicit real*8(a-h,o-z)
      open(1,file='rk4chaos.dat')
```

```
C
C     Initial Values
C
      t1=0
      x1=0
      y1=1.0
      z1=0
C
      h=0.01
      h2=h/2
      h6=h/6
C
      write(1,*)y2,z2,x2,t2
C
      do i=1,10000
         a1=f1(x1,y1,z1)
         b1=f2(x1,y1,z1)
         c2=f3(x1,y1,z1)
C
         x11=x1+h2*a1
         y11=y1+h2*b1
         z11=z1+h2*c1
C
         a2=f1(x11,y11,z11)
         b2=f2(x11,y11,z11)
         c2=f3(x11,y11,z11)
C
         x12=x1+h2*a2
         y12=y1+h2*b2
         z12=z1+h2*c2
C
         a3=f1(x12,y12,z12)
         b3=f2(x12,y12,z12)
         c3=f3(x12,y12,z12)
C
         x12=x1+h*a3
         y12=y1+h*b3
         z12=z1+h*c3
C
          a4=f1(x13,y13,z13)
          b4=f2(x13,y13,z13)
```

```
           c4=f3(x13,y13,z13)
C
           x2=x1+h6*(a1+2*2+2*a3+a4)
           y2=y1+h6*(b1+2*b2+2*b3+b4)
           z2=z1+h6*(c1+2*c2+2*c3+c4)
           t2=t1+h
C
           write(1,*)x2,y2,z2,t2
C
C          Update…
C
           x1=x2
           y1=y2
           z1=z2
           t1=t2
         enddo
C
         stop
         end
C
         function f1(x,y,z)
         implicit real*8(a-h,o-z)
         sigma=10
         f1=sigma*(y-x)
         return
         end
C
         function f2(x,y,z)
         implicit real*8(a-h,o-z)
         rho=28
         f2=x*(rho-z)-y
         return
         end
C
         function f3(x,y,z)
         implicit  real*8(a-h,o-z)
         beta=8/3.0
         f3=x*y-beta*z
          return
          end
```

* Plotting x against y and z against z, we get the attractors.





Real Roots by Bisection Method

Theory:

Consider an equation, $f(x) = 0$. If a real root exists between$[a, b]$, we search that root by bisecting the interval; the midpoint being $x_m = (a + b)/2$. If $f(x_m) = 0$, x_m is the root. If not so, we choose either $[a,\ x_m]$ or $[x_m,\ b]$. We check whether $f(x)$ has opposite signs at the end points (over any of the intervals). Then we take that new interval and bisect again. Thus we keep searching the root according to some accuracy (we call it tolerance).

Algorithm:

1. Function $f(x)$ defined in subroutine
2. Input: $a,\ b,\ \epsilon$
1. If $f(a).f(b) > 0$, No Root. Stop
2. Else set $x_m = 0.5 \times (a + b)$
3. If $f(x_m) = 0$, x_m is the root. Stop
4. If $f(a).f(x_m) < 0$, set $b = x_m$. Else set: $a = x_m$.
5. Check if $|b - a| < \epsilon$. Root = $0.5 \times (a + b)$. Stop
6. Else go to step 3

Example:

Consider the equation: $x^3 - 3x - 5 = 0$. Let us find a real root in the interval, $[2, 20]$.
The following is the result of the iterations that is obtained by running the program.

FORTRAN Program:

```
C       Bolzano Bisection Method

        write(*,*)'Lower Limit, Upper Limit, Tolerance'
        read(*,*)a,b,eps
C
C       Testing if the root exists in this range
C
        if(f(a)*f(b).gt.0.0)then
           write(*,*)'No Root'
           stop
        endif
C
C       Bisection Loop starts
C
        do while(abs(b-a).ge.eps)
           xm=0.5*(a+b)
           if(f(xm).eq.0.0)go to 10
           if(f(a)*f(xm).lt.0.0)then
              b=xm
           else
              a=xm
           endif
        enddo
10      write(*,*)'Root = ',xm
        stop
        end
C
        function f(x)
        f=x**3-3*x-5
        return
        end
```

No. Iterations	x_m
1	11.000000
2	6.5000000
3	4.2500000
4	3.1250000
5	2.5625000
6	2.2812500
7	2.1406250
8	2.2109375
9	2.2460938
10	2.2636719
11	2.2724609

No. Iterations	x_m
12	2.2768555
13	2.2790527
14	2.2779541
15	2.2785034
16	2.2787781
17	2.2789154
18	2.2789841
19	2.2790184
20	2.2790356
21	2.2790270

Roots by Newton-Raphson Method:

Theory:

If we can find the derivative $f'(x)$ such that $f'(x) \neq 0$ and also know that $f(x)$ is monotonic in $[a, b]$, we have: $x_{n+1} = x_n - \frac{f(x)}{f'(x)}$

Starting from an approximate value x_0, we can generate the sequence $x_1,\ x_2,\ x_3$ ……

Algorithm:

1. Define $f(x)$ and derivative $f'(x)$ in subroutine
2. Input: Guess value of x, Tolerance, Max. no. of Iterations
3. Iteration in do loop: $x_{n+1} = x_n - f(x)/f'(x)$
4. Update: $x_{n+1} \rightarrow x_n$
5. Check if $|x_{n+1} - x_n| < tol$, then either x_{n+1} or x_n is the root. Stop.
6. Else continue until max iterations.

FORTRAN Program:

```
C       Newton-Raphson Method

        write(*,*)'Initial approx, Tolerance, Max iteration'
        read(*,*)x,tol,maxit
        do i=1,maxit
           x0=x
           x=x-f(x)/f1(x)
           if(abs(x-x0).lt.tol)then
              root=x
              write(*,*)'Root =', root
              stop
           endif
           if(i.eq.maxit)write(*,*)'No Solution'
        enddo
        stop
        end
C
        function f(x)
        f=x**3-3*x-5
        return
        end
C
        function f1(x)
        f1=3*x**2-3
        return
        end
```

APPENDIX I

Least Square Fit:

Theory:

The following is a theory for data to be fitted with a straight line.

The equation of a straight line is $y = mx + b$

Consider the data points (x_1, y_1), (x_2, y_2).....etc.

Error is defined as $\varepsilon(m,b) = \sum_{i=1}^{n}(mx_i + b - y_i)^2$.

For the best fit, this error should be minimum.

Therefore, $\frac{\partial \varepsilon}{\partial m} = 0$ and $\frac{\partial \varepsilon}{\partial b} = 0$.

Now, $\frac{\partial \varepsilon}{\partial m} = \frac{\partial}{\partial m}\left[\sum_{i=1}^{n}(mx_i + b - y_i)^2\right] = \sum_{i=1}^{n}\frac{\partial}{\partial m}(mx_i + b - y_i)^2$

$$= \sum_{i=1}^{n} 2.(mx_i + b - y_i)\frac{\partial}{\partial m}(mx_i + b - y_i)$$

$$= 2\sum_{i=1}^{n}(mx_i + b - y_i)x_i = 2m\sum_{i=1}^{n}x_i^2 + 2b\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}y_i x_i = 0 \qquad (1)$$

Similarly, $\frac{\partial \varepsilon}{\partial b} = 0 = m\sum_{i=1}^{n}x_i + nb = \sum_{i=1}^{n}y_i$ (2)

From (1) and (2),

$$\begin{pmatrix} n & \sum_{i=1}^{n}x_i \\ \sum_{i=1}^{n}x_i & \sum_{i=1}^{n}x_i^2 \end{pmatrix}\begin{pmatrix} b \\ m \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n}y_i \\ \sum_{i=1}^{n}y_i x_i \end{pmatrix}$$

Slope, $m = \frac{n\sum_{i=1}^{n}x_i y_i - \sum_{i=1}^{n}y_i \sum_{i=1}^{n}x_i}{n\sum_{i=1}^{n}x_i^2 - (\sum_{i=1}^{n}x_i)^2}$ and Intercept, $b = \frac{\sum_{i=1}^{n}y_i \sum_{i=1}^{n}x_i^2 - \sum_{i=1}^{n}x_i \sum_{i=1}^{n}x_i y_i}{n\sum_{i=1}^{n}x_i^2 - (\sum_{i=1}^{n}x_i)^2}$.

Example:

For the data points (1,2), (2,3), (3,4), (4,5)

$n = 4$, $\sum_{i=1}^{n}x_i = 1+2+3+4 = 10$, $\sum_{i=1}^{n}y_i = 2+3+4+5 = 14$

$\sum_{i=1}^{n}x_i y_i = 1\times2 + 2\times3 + 3\times4 + 4\times5 = 40$, $\sum_{i=1}^{n}x_i^2 = 1\times1 + 2\times2 + 3\times3 + 4\times4 = 30$

$$\therefore m = \frac{4\times 40 - 14\times 10}{4\times 30 - 10^2} = \frac{160-140}{120-100} = \frac{20}{20} = 1,\ \ b = \frac{14\times 30 - 10\times 40}{4\times 30 - 10^2} = \frac{420-400}{120-100} = \frac{20}{20} = 1.$$

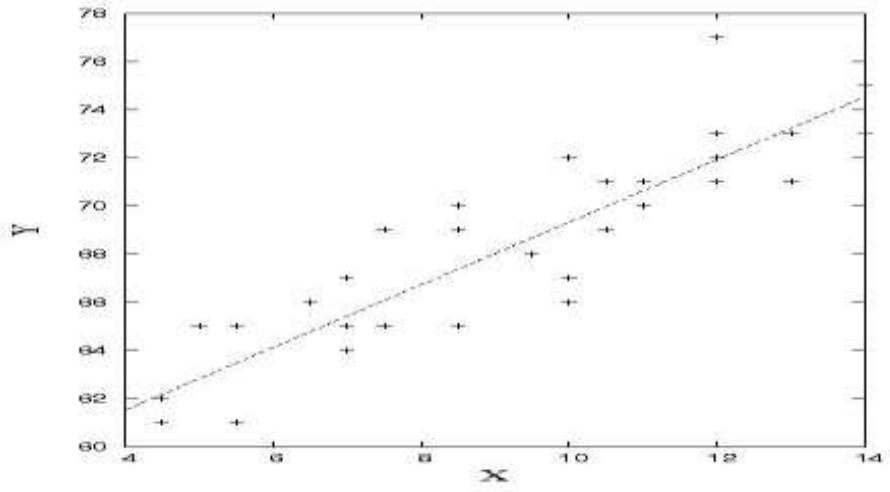


Fig. Example of a linear least square fit

Simpson's 1/3 Rule:

Theory:

Simpson's 1/3 rule is where the integrand is approximated by a second order polynomial.

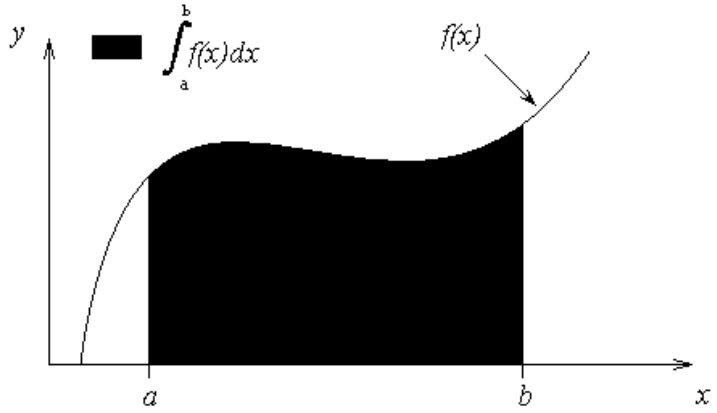


Fig. Integration of a function

$$I = \int_a^b f(x)dx \approx \int_a^b f_2(x)dx$$

where $f_2(x)$ is a second order polynomial given by

$$f_2(x) = a_0 + a_1 x + a_2 x^2$$

.

Choose

$$(a, f(a)), \left(\frac{a+b}{2}, f\left(\frac{a+b}{2}\right)\right), \text{ and } (b, f(b))$$

as the three points of the function to evaluate a_0, a_1 and a_2.

$$f(a) = f_2(a) = a_0 + a_1 a + a_2 a^2$$

$$f\left(\frac{a+b}{2}\right) = f_2\left(\frac{a+b}{2}\right) = a_0 + a_1\left(\frac{a+b}{2}\right) + a_2\left(\frac{a+b}{2}\right)^2$$

$$f(b) = f_2(b) = a_0 + a_1 b + a_2 b^2$$

Solving the above three equations for unknowns, a_0, a_1 and a_2 give

$$a_0 = \frac{a^2 f(b) + abf(b) - 4abf\left(\frac{a+b}{2}\right) + abf(a) + b^2 f(a)}{a^2 - 2ab + b^2}$$

$$a_1 = -\frac{af(a) - 4af\left(\frac{a+b}{2}\right) + 3af(b) + 3bf(a) - 4bf\left(\frac{a+b}{2}\right) + bf(b)}{a^2 - 2ab + b^2}$$

$$a_2 = \frac{2\left(f(a) - 2f\left(\frac{a+b}{2}\right) + f(b)\right)}{a^2 - 2ab + b^2}$$

Then

$$I \approx \int_a^b f_2(x)dx$$

$$= \int_a^b \left(a_0 + a_1 x + a_2 x^2\right)dx$$

$$= \left[a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3}\right]_a^b$$

$$= a_0(b-a) + a_1 \frac{b^2 - a^2}{2} + a_2 \frac{b^3 - a^3}{3}$$

Substituting values of a_0, a_1 and a_2 gives

$$\int_a^b f_2(x)dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Since for Simpson 1/3 rule, the interval $[a,b]$ is broken into 2 segments, the segment width

$$h = \frac{b-a}{2}$$

Hence the Simpson's 1/3 rule is given by

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

Since the above form has 1/3 in its formula, it is called **Simpson's 1/3 rule**.

Simpson's 1/3 Rule (modified version)
(Multiple-segment Simpson's 1/3 Rule)

Theory:

For better result, one can subdivide the interval $[a,b]$ into n segments and apply Simpson's 1/3 rule repeatedly over every two segments. Note that n needs to be even. Divide interval $[a,b]$ into n equal segments, so that the segment width is given by

$$h = \frac{b-a}{n}.$$

Now

$$\int_a^b f(x)dx = \int_{x_0}^{x_n} f(x)dx$$

where

$$x_0 = a$$

$$x_n = b$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + ... + \int_{x_{n-4}}^{x_{n-2}} f(x)dx + \int_{x_{n-2}}^{x_n} f(x)dx$$

Apply Simpson's 1/3rd Rule over each interval,

$$\int_a^b f(x)dx \cong (x_2 - x_0)\left[\frac{f(x_0) + 4f(x_1) + f(x_2)}{6} \right] + (x_4 - x_2)\left[\frac{f(x_2) + 4f(x_3) + f(x_4)}{6} \right] + ...$$

$$+ (x_{n-2} - x_{n-4})\left[\frac{f(x_{n-4}) + 4f(x_{n-3}) + f(x_{n-2})}{6} \right] + (x_n - x_{n-2})\left[\frac{f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)}{6} \right]$$

Since

$$x_i - x_{i-2} = 2h$$

$$i = 2,\ 4,\ ...,\ n$$

then

$$\int_a^b f(x)dx \cong 2h\left[\frac{f(x_0)+4f(x_1)+f(x_2)}{6}\right]+2h\left[\frac{f(x_2)+4f(x_3)+f(x_4)}{6}\right]+\ldots$$
$$+2h\left[\frac{f(x_{n-4})+4f(x_{n-3})+f(x_{n-2})}{6}\right]+2h\left[\frac{f(x_{n-2})+4f(x_{n-1})+f(x_n)}{6}\right]$$

$$=\frac{h}{3}\left[f(x_0)+4\left\{f(x_1)+f(x_3)+\ldots+f(x_{n-1})\right\}+2\left\{f(x_2)+f(x_4)+\ldots+f(x_{n-2})\right\}+f(x_n)\right]$$

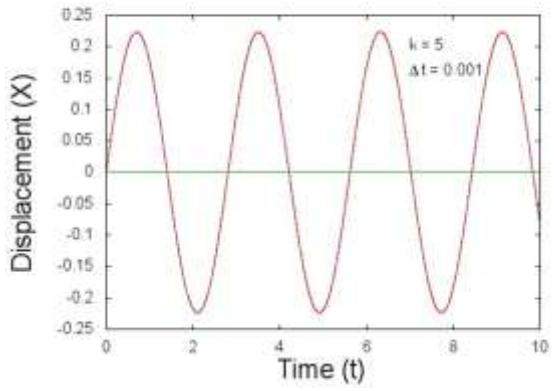
APPENDIX II

Oscillations: a pictorial tour

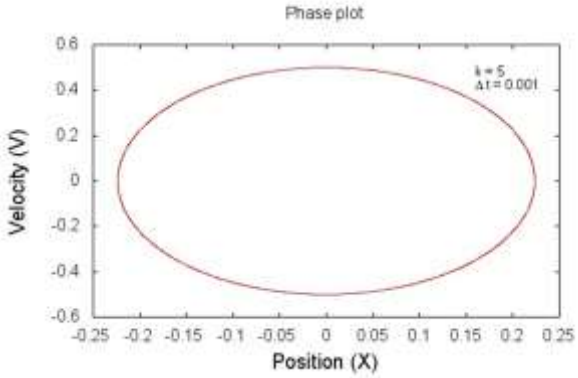
(Results obtained from solving corresponding differential equations in computer)

Simple Harmonic Motion

(No damping)

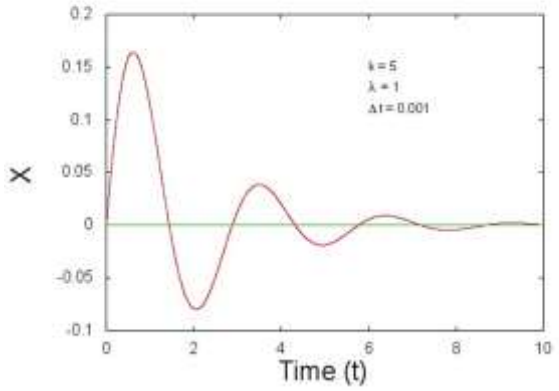


$$\frac{d^2X}{dt^2}+k.X=0$$

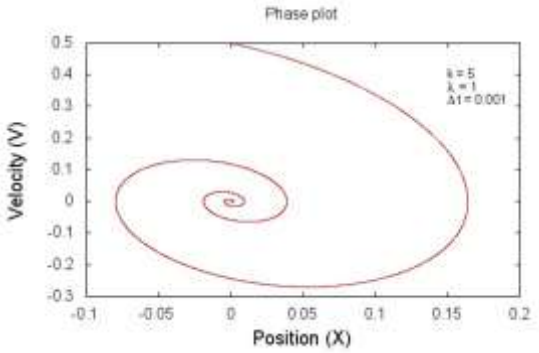


Damped Harmonic Motion

(Critical Damping)

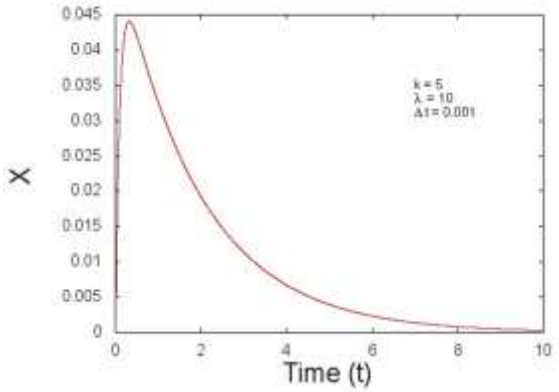


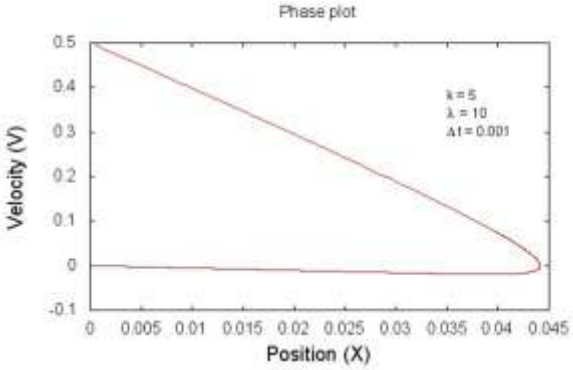
$$\frac{d^2X}{dt^2} + \lambda.\frac{dX}{dt} + k.X = 0$$



Damped Harmonic Motion

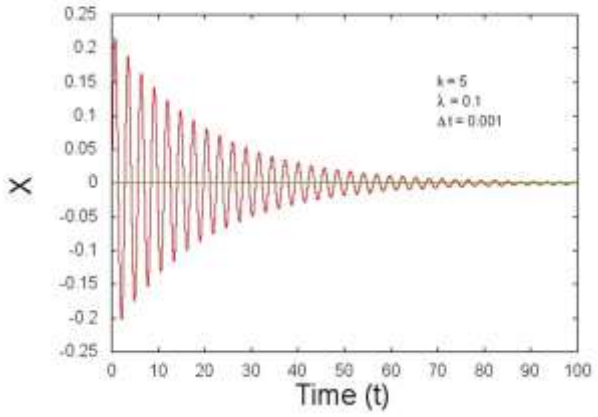
(Large damping)

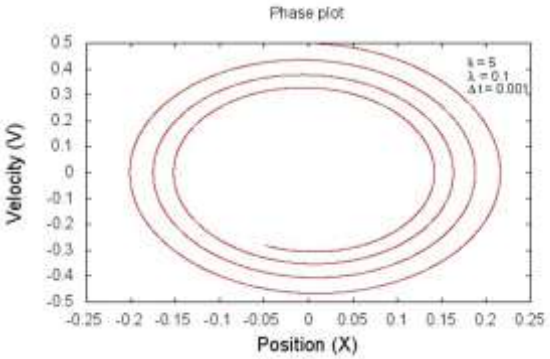




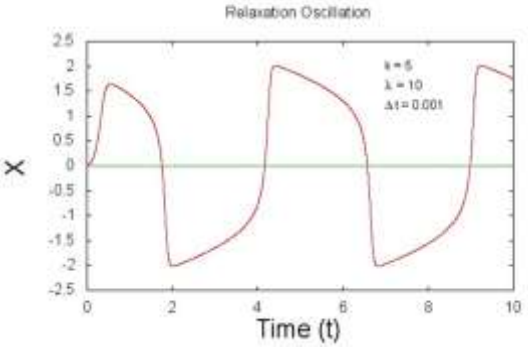
Damped Harmonic Motion

(Small damping)



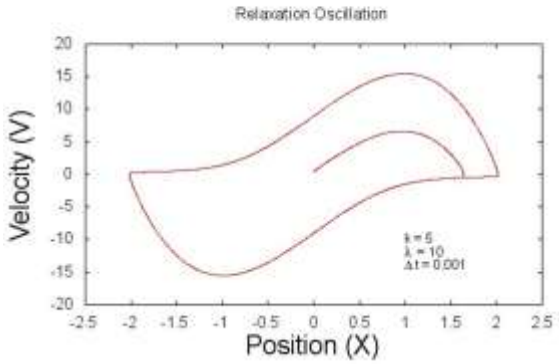


Relaxation Oscillation



Van der Pol Oscillator:

$$\frac{d^2X}{dt^2} + \lambda(1 - X^2).\frac{dX}{dt} + k.X = 0$$



Exercises:

1. Consider the following 2×3 matrix, $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$. Write a computer program to obtain the corresponding transpose matrix B. Also find the product $C = AB$. Display the result.
2. Write a computer program to obtain the first 100 numbers in the Fibonacci sequence and find the golden ratio. [Golden ratio is the ratio of successive terms.] Is the Golden ratio tending towards a fixed and finite number?
3. Write a program using the Newton-Raphson method to determine the roots of the equation: $f(x) = x^3 - x^2 - 2x + 1$.
4. Write a program to determine the roots of the equation: $x^4 - 1.99x^3 - 1.76x^2 + 5.22x - 2.23 = 0$ that are close to $x = 1.5$.
5. Write a program to calculate the correlation coefficient for the 5 pairs of experimental observations corresponding to some measurement: (2,5), (4,9), (5,11), (6,10), (8,12). Use the following formula: $\boldsymbol{r} = \frac{n\sum xy - \sum x \sum y}{\sqrt{n\sum x^2 - (\sum x)^2}.\sqrt{n\sum y^2 - (\sum y)^2}}$
6. Write a computer program to convert a binary number 10101 into its decimal equivalent and then add the digits of the decimal number.
7. Compute numerically the following integral to verify the answer or obtain a value close to that: $\int_0^{\pi} \frac{x}{x^2+1} \cos(10x^2) = 0.0003156$
8. You are given a set of height-weight data as follows: (170, 65), (172, 66), (181, 69), (157, 55), (150, 51). Write a program to find correlation coefficient, $r = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$, where ${\sigma_x}^2 = \frac{1}{n}\sum(x_i - \bar{x})^2$, ${\sigma_y}^2 = \frac{1}{n}\sum(y_i - \bar{y})^2$ and $\sigma_{xy} = \frac{1}{n}\sum(x_i - \bar{x})(y_i - \bar{y})$.
9. Write a program to convert the binary number 1000,000,000 into a decimal number.
10. 123_8=73_{10}. Is that correct? Verify this through a computer program.
11. Write a program to find the transpose of the following matrix: $A = \begin{pmatrix} 3 & 4 & 8 \\ 5 & 9 & 2 \\ 1 & 6 & 0 \end{pmatrix}$. Also verify, $Tr(A) = Tr(A^T)$.
12. Using Simpson's modified 1/3 rd rule, write a program to calculate: $\int_0^1 (x^2 + 1)dx$.
13. Consider the number: 787. Is the number a prime? Test this through computer.
14. Compute this: $\int_{1.8}^{3.4} f(x)dx$, where we have

x	1.8	2.0	2.2	2.4	2.6	2.8	3.0	3.2	3.4
$f(x)$	6.050	7.389	9.025	11.023	13.464	16.445	20.086	24.533	29.964

15. Write a program in computer to find out the $\sin(x)$ through infinite series. Calculate up to an accuracy level of 10^{-6}. Find the value of $\sin 45^0$.

16. Find the value of π from the infinite series: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} + \cdots$ up to 6th decimal of accuracy.
17. Use Newton-Raphson method to find a non-zero solution of $x = 2\sin x$.
18. Write a computer program for a $m \times n$ matrix A where the elements are $(-1)^{m+n}mn$ and another $m \times n$ matrix B whose elements are $(-1)^{m+n}m/n$ and then compute the matrix, $C = A.B + B.A$.
19. For a cart sliding down an inclined plane, the relation between the distance (d) and time (t) is given by, $d = \frac{1}{2}g\sin\theta t^2$. A set of measurements are done as following.

t (sec)	0.7	1.3	1.9	2.5	3.1	3.7	4.1	4.9	5.6	6.1	6.7	7.5	7.9	8.5	9.1
d (cm)	1	4	9	16	25	36	49	64	81	100	121	144	169	196	225

Write a program to fit this data set. If you know the angle of inclination, $\theta = 30^0$, how do you find the acceleration of gravity, g from this measurements and your fitting exercise?

20. Given a number, 8956, write a program to compute the sum of the digits of this number.
21. Given two matrices, $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, write a computer program to compute $C = 5A - 3B$.
22. Given two matrices, $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, write a computer program to compute, $C = AB + BA$.
23. Write a program to verify approximately, $\ln 100! \approx 100\ln 100 - 100$. Calculate also the percentage of error you incurred.
24. Write a suitable program to approximately verify, $\int_0^\infty e^{-x^2}dx = \frac{1}{2}\sqrt{\pi}$. Sate the accuracy level that you are dealt with.
25. Given two matrices, $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, write a computer program to confirm that $AB^T = B^T A^T$.
26. Write a program where the function $f(x) = (x-1)(x-2)(x-3)(x-4)(x-5)$ is defined in a subroutine and then obtain$f(6)$.
27. In a Nuclear detector, the counting rates vs. time are measured in the following table. If you would draw a mean straight line through them, what would be the slope of that?

Time (sec)	0	15	30	45	60	75	90	105	120	135
Counts per sec	106	80	98	75	74	73	49	38	37	22

28. Convert $(752.612)_{10}$ to equivalent binary.
29. A set of 20 numbers are given: 1, 0.1, 5, 3, 10, -1, 4, 20, 1000, -9, 2, 14, 4.5, 0.9, 30, 9.8, 11, 22, 48, -10. Write a program to count how many numbers are between 0 to 10 and how many are between 50 to 1000.
30. While solving a cubic equation, we found a recursion relation: $x_{n+1} = x_n - \frac{({x_n}^3 - x_n - 1)}{3{x_n}^2 - 1}$. We are told that a root exists between 1 and 2. Starting from an initial guess, you have to find out the root. Write a computer program for that.
31. Solve $\cos x = 2x$ to 5 decimal places. You can start with an initial guess $x_0 = 0.5$.
32. $\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$ Verify this approximately up to an accuracy of order 10^{-05} by writing a computer program.
33. Write a program to show: $10001011_2 = 213_8$.

This set of programs is intended to be distributed among physics students (U.G. and P.G.) for their foundational coursework in Computation.

NOTE: In this basic course, I have avoided using formatted input and output. Format statements can be used in the 'read' and 'write' statements…..

FORTRAN Programs for Physics Students

(With Theory and Algorithms)

Abhijit Kar Gupta

Department of Physics, Panskura Banamali College
Panskura R.S., East Midnapore, West Bengal, India, Pin Code: 721152
e-mail: kg.abhi@gmail.com