IC 150 P Computation for engineers lab
Lab assignment sheet no: 3, Odd semester, 2016

Simple `for`, `while` loops

Prepared by: Maben Rabi

1 Objectives for this lab session

1. To write simple loops using the `for` statement, and the `while` statement.
2. Learn and implement three simple line search methods for function minimization:
   - Left to Right search with only function evaluations,
   - Trisection method with only function evaluations, and,
   - Golden section method with only function evaluations.

2 The `for` and `while` statements

Read Sections 1.2, 1.3, 3.5, 3.6 from Kernighan and Ritchie. A loop is a special segment of a program where a given sequence of instructions is executed many times. The `for` statement is useful for loops where it is possible to know ahead of time the number of iterations of the loop. The `while` statement is useful for loops where it is NOT possible to know ahead of time the number of iterations of the loop. Such a loop will terminate based on a decision that will be made on the go. Here is a simple program that illustrates the use of the `for` statement.

```
#include<stdio.h>
#include<math.h>

void main(){
  int   n;
  float count, sum;

  printf("This program calculates the sum 1 + 1/2^2 + ... +
     1/n^2\n");
  printf("Enter the positive integer value of n: ");
  scanf("%d",&n);

  sum = 0;
  for (count=1 ; count <= n ;  count = count + 1){
    sum = sum + 1/count/count ;
  }
  printf("The sum is is: %.5f\n",sum);
}
```

simpleForLoop.c

Check if the loop is being run the correct number of times.
A delicate point: In the above program, we have declared the variable `count` as a floating point number. Investigate what will happen if we declare it as an integer ?
Next, we present a longer version of the program, and here the `while` statement is also used.

```
#include<stdio.h>
#include<math.h>

void main(){
  int   n, validEntry;
  float count, sum;

  printf("This program calculates the sum 1 + 1/2^2 + ... +
     1/n^2\n");

  validEntry = 0;
  while (validEntry == 0)
  {
  printf("Enter a nonnegative integer value of n: ");
  scanf("%d",&n);
    if(n >= 0)
    {
        validEntry = 1;
    }
  }

  sum = 0;
  for (count=1 ; count <= n ;  count = count + 1){
    sum = sum + 1/count/count ;
  }
  printf("The sum is is: %.5f\n",sum);
}
```

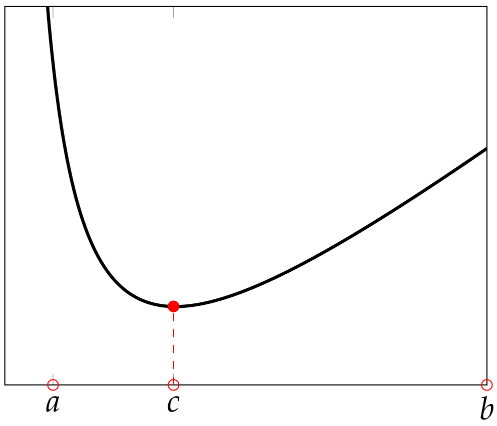
simpleForAndWhileLoops.c

3 Some line searches for function minimization

The material in this section is heavily based on Chapter five of the book: *Optimization: insights and applications,* by Jan Brinkhuis, and Vladimir Tikhomirov.

Definition of an unimodular function. We say a function $f : \mathbb{R} \to \mathbb{R}$ is unimodular on the interval $[a, b]$ if the following two conditions are satisfied:

1. f is continuous, and,
2. there is a point $c \in [a, b]$ such that f is decreasing (or at least non-increasing) on $[a, c]$, AND, that f is increasing (or at least non-decreasing) on $[c, b]$.



Simple examples of unimodal functions are $f(x) = x^2$, and $f(x) = mx + c$, on any interval $[a, b]$.

3.1 Left to Right search

Here, we divide the interval $[a, b]$ into $n+1$ equal intervals by the sequence of points:

$$x_i = a + i \times \frac{(b-a)}{n+1}, \text{ for } 0 \le i \le n+1.$$

We may evaluate the values of function at some or all of the points in this sequence. We evaluate the values of function at some or all of the points in this sequence.

3.1.1 Full Left to Right search

In this version, we compute the function values along this sequence starting at the point x_1 and going till the point x_n. As we do this, we keep track of the minimum of the function values, and the point where this minimum was achieved. Suppose for example that we computed the sequence of values of f and have also found that the minimum value occurs at x_6. Then we declare that the minimizing x lies in the interval $[x_5, x_7]$, and that the minimum value is approximately $f(x_6)$.

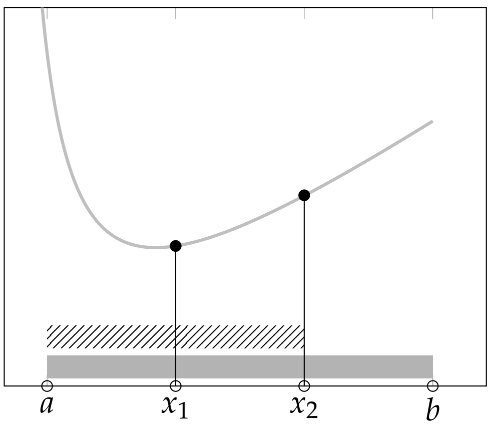
3.1.2 Left to Right search up to bend

In this version also, we divide the interval $[a, b]$ into $n+1$ equal intervals by the above sequence of points. We compute the function values along this sequence starting at the point x_1, but we may not go up to the final point x_{n+1}. We stop at the point x_i if $f(x_i) \ge f(x_{i-1})$. Then we declare that the minimizing x lies in the interval $[x_{i-2}, x_i]$, and that the minimum value is approximately $f(x_{i-1})$. This method gives reliable results only when the function f is unimodular.

3.2 Trisection method with only function evaluations

This method is also guaranteed to work only for unimodular functions. The basic action unit of this method is a scheme to reduce the interval of uncertainty to a smaller interval that has two thirds of the original length.

Trisection step $[a, b]$: Suppose that we are given an unimodular function f on a bounded interval $[a, b]$, and are asked to find out where the function attains its minimum. Then we will find a shorter interval inside $[a, b]$ where the minimum will be contained. We do this by carrying out two function evaluations, and a comparison. The evaluations will be at the points: $x_1 = (2a+b)/3, x_2 = (a+2b)/3$. Since f is unimodular:
if $f(x_1) < f(x_2)$, then the minimizing x is contained within the interval $[a, x_2]$,
if $f(x_1) > f(x_2)$, then the minimizing x is contained within the interval $[x_1, b]$.



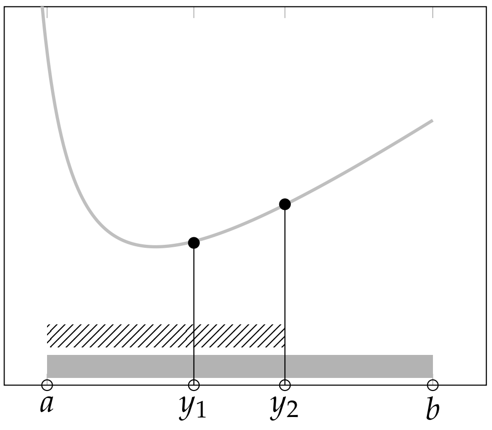
If there is a tie ($f(x_1) = f(x_2)$), then it does not matter which of the two intervals is picked. Either alternative serves our purpose equally well. We now say that the minimizing x is contained in the shorter interval. On the figure, the shaded horizontal bar represents the initial interval of uncertainty, and the hashed horizontal bar represents the shorter interval of uncertainty obtained by applying the Trisection step.

We can then carry out the above Trisection step on this shorter interval, and so on and on.

3.3 Golden section method with only function evaluations

This method is also guaranteed to work only for unimodular functions. Like for the trisection method, the basic action unit of this method is a scheme to reduce the interval of uncertainty to a smaller interval.

Golden section step $[a, b]$: Suppose that we are given an unimodular function f on a bounded interval $[a, b]$, and are asked to find out where the function attains its minimum. Then we will find a shorter interval inside $[a, b]$ where the minimum will be contained. We do this by carrying out two function evaluations, and a comparison. The evaluations will be at the points: y_1, y_2, which are chosen such that:



- The point y_2 divides the interval $[a, b]$ in the Golden ratio. This means: $y_2 - a = \frac{\sqrt{5}-1}{2}(b - a)$.
- The point y_1 divides the interval $[a, y_2]$ in the Golden ratio. This means: $y_1 - a = \frac{\sqrt{5}-1}{2}(y_2 - a)$.

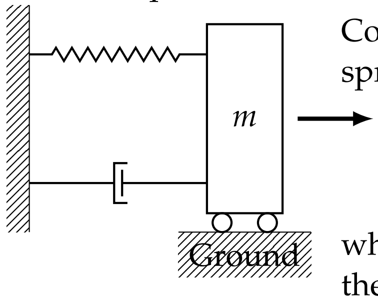
The Golden ratio $\frac{\sqrt{5}-1}{2} = 0.618034$ is chosen for a special reason. Suppose that an interval is divided into two pieces with the longer piece having a length equal to the Golden ratio times the length of the original interval. Then the ratio of lengths of the smaller piece to the bigger piece is also given the Golden ratio. Just like we had in the trisection method, since f is unimodular:
if $f(y_1) < f(y_2)$, then the minimizing x is contained within the interval $[a, y_2]$,
if $f(y_1) > f(y_2)$, then the minimizing x is contained within the interval $[y_1, b]$.
If there is a tie ($f(y_1) = f(y_2)$), then it does not matter which of the two intervals is picked. Either alternative serves our purpose equally well. We now say that the minimizing x is contained in the shorter interval. On the figure, the shaded horizontal bar represents the initial interval of uncertainty, and the hashed horizontal bar represents the shorter interval of uncertainty obtained by applying the Golden section step.

We can then carry out the above Golden section step on this shorter interval, and so on and on. There is one special advantage to this method. In the trisection method, at every run of the Trisection step we need to compute the two points x_1, x_2 that trisect the given interval. However in the Golden ratio method we need to compute two points y_1, y_2 only in the very first run of the Golden section step. The reason is the following. Suppose that we have carried out the Golden section step on the interval $[a, b]$ and obtained the smaller interval $[a, y_2]$. Then notice that the point y_1 already divides the interval $[a, y_2]$ into Golden ratio pieces. Hence, we only need to evaluate the function at one new Golden ratio point. Therefore in the Golden ratio method, we nee two function evaluations at the very first run, and in subsequent runs, we need only on new function evaluation.

4 A unimodular function from physical systems

Note: You may skip this section if you wish. If you do skip, then copy and use the expression for the quantity IAE as a function of ζ.

There is a common way to describe the behavior of electrical circuits made of one inductor, one capacitor and some resistors, as well as mechanical systems made up of one spring, one damper and some masses. All of such systems can be described using second order linear differential equations. You will learn this in IC 260 Signals and Systems.



Consider the differential equation that describes the motion of the spring mass system shown in the figure:

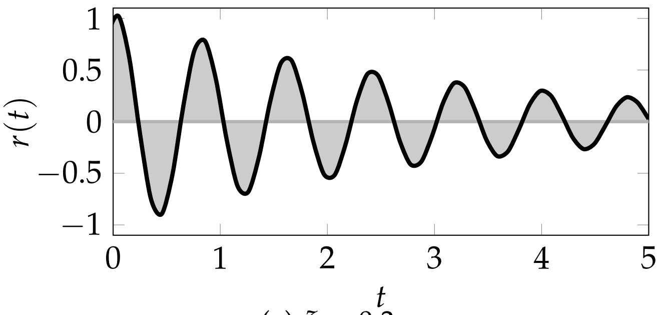
$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0,$$

where m is the mass of the block, c is the damping coefficient, and k is the spring constant.

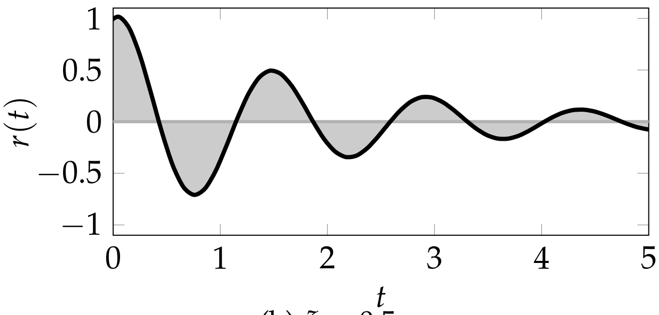
This can be rewritten in a standard form as

$$\frac{d^2x}{dt^2} + 2\zeta\omega_n\frac{dx}{dt} + \omega_n^2 x = 0,$$

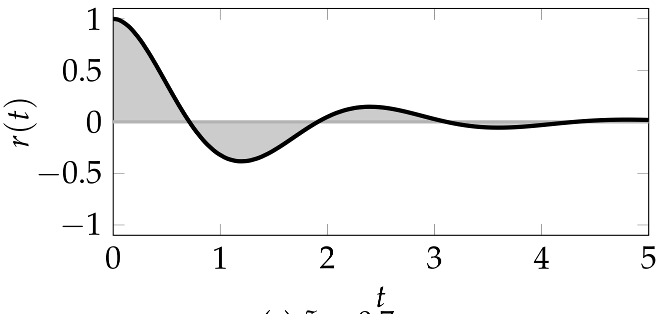
where $\omega_n = \sqrt{k/m}$, and $\zeta = \frac{c}{2m\omega_n}$. The parameter ω_n is called the natural angular frequency, and the parameter ζ is called the damping ratio. The design problem is to choose the damping coefficient c so that the resulting system has a good response to vibrations and shocks. This is equivalent to choosing ζ for the same desired effect. Suppose that a sudden jerk has moved the block from its equilibrium position to a displacement of say 1cm, then the whole system will be set in motion, but will eventually return to its equilibrium position. We want the trajectory of the block to be as close as possible to the equilibrium without fluctuating too wildly.



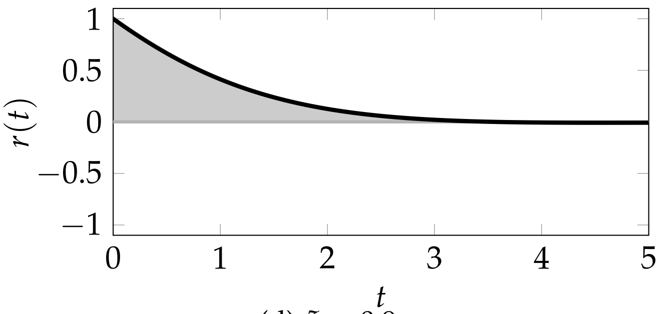
(a) $\zeta = 0.3$



(b) $\zeta = 0.5$



(c) $\zeta = 0.7$



(d) $\zeta = 0.9$

If the damping ratio ζ is between 0 and 1, then the trajectory of the block will take the form:

$$r(t) = e^{-\zeta\omega_n t}\cos\omega_n\sqrt{1-\zeta^2}\;t + \frac{\zeta}{\sqrt{1-\zeta^2}}e^{-\zeta\omega_n t}\sin\omega_n\sqrt{1-\zeta^2}\;t.$$

The figures above show the trajectories for a fixed value of ω_n, and four different values of ζ. Which of these four trajectory is preferable to the others ?

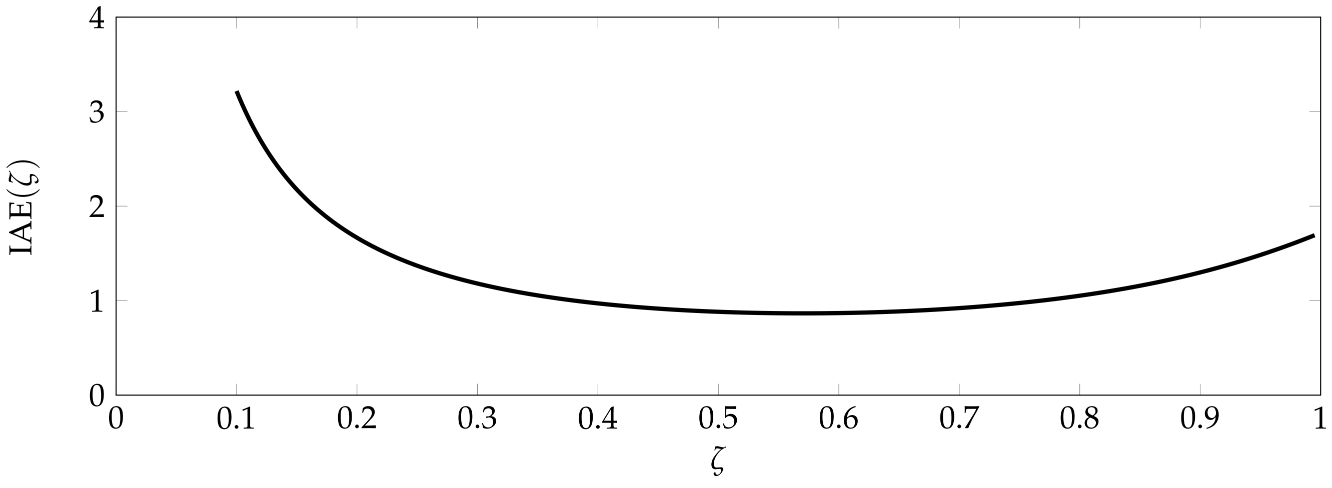
One criterion to choose the damping ratio ζ is to choose it in such a way that the following number, called the *Integral absolute error*

$$\text{IAE} = \int_0^\infty |r(t)|\,dt$$

is as low as possible. Graphically this IAE is the total unsigned area betweeen the graph of the trajectory $r(t)$ and the positive time axis. Using techniques that you will learn in IC 260, it is posible for calculate the function IAE (ζ). That calculation produces:

$$\mathrm{IAE}(\zeta) = \text{constant} \times \left(\frac{\left(1 + e^{-\pi\zeta/\sqrt{1-\zeta^2}}\right)^2 \times e^{\left[\arcsin\left(\sqrt{1-\zeta^2}\right)\right] \times \zeta/\sqrt{1-\zeta^2}}}{\left(1 - e^{-2\pi\zeta/\sqrt{1-\zeta^2}}\right) \times (2-\zeta^2)} - \frac{1}{2-\zeta^2}. \right) \tag{1}$$

It can be seen from plotting it that it is a unimodular function on the interval $[0, 1]$. This is the function you will use for all the tasks below. The interval of interest is of course $[0, 1]$.



5 Task One

Part A: Write a C program that will compute an approximate minimum and minimizing ζ by using the Full Left to Right search.
Part B: Expand on the earlier C program, and make it to also compute an approximate minimum and minimizing ζ by using the Left to Right search up to a bend. Print out the number of function evaluations needed by the two methods.

6 Task Two

Write a C program that will compute an approximate minimum and minimizing ζ by using the Trisection method. Will you use a `for` loop based series of runs, or a `while` loop based series of runs ?

7 Task Three

Expand on the C program from task two. Make it to also compute an approximate minimum and minimizing ζ by using the Golden section method. Will you use a `for` loop based series of runs, or a `while` loop based series of runs ? Print out the number of function evaluations needed by the two different sectioning methods.