Computational Physics

Saint Mary's College
Chris Ray
Spring 2017

Contents

1 Computation

This chapter is a tutorial on the use of Matlab for doing simple computations.

§ 1.1 Arithmetic

Launch Matlab and enter at the command prompt `>>` type `1+1`. You should see something like the following on your screen.

```
>> 1+1
ans =
        2
```

This is not terribly exciting, but it is at least correct. We see that the computer can be used like a handheld calculator, it just has a bigger keyboard and screen.

▷ Problem 1.1
Enter the following expressions, and explain the results. Try to predict the results before typing the expression into the computer.
(a) `pi`
(b) `sin(pi/6)`
(c) `4*cos(pi/6)^2`
(d) `cos(2pi/3)`
(e) `2*3+4`
(f) `2+3*4`
(g) `2*(3+4)`
(h) `6/3+2`
(i) `6/(3+2)`
(j) `2*3^4`
(k) `2^3*4`
(l) `2^(3*4)`
(m) `2*asin(1)`
(n) `5.1e3`
(o) `1-(1-2e-16)`
(p) `1-(1-2e-17)`
(q) `exp(0.6931)`
(r) `sqrt(9)`
(s) `sqrt(-9)`
(t) `i^2`
(u) `(1+3i)^2`

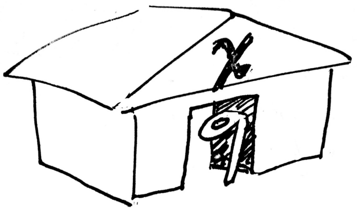
(v) `exp(i*pi)`
(w) `log(100)`
(x) `log10(100)`
(y) `c = 3e8`
(z) `c/3`

§ 1.2 Memory

As we saw in the previous exercise, we were able to create a constant `c` which held the (approximate) value of the speed of light, and that we could use the constant `c` in other expressions just as if we typed the number `300000000`. What is happening here is that the computer is setting aside a location in memory to store the number and allowing you to refer to the location by the symbol `c`. Once this memory location has been allocated it will be available to you under the name `c` unless you specifically tell the computer to forget it. Try the following sequence of expressions at the command line.

```
>> x = 9;
>> disp(x*2)
>> clear x
>> disp(x*2)
```

The best way to think about it is like this. A variable is made of two things, the location of the data and the value kept at that location. Here is the value `9` at home in her house with address `x`



§ 1.3 The symbol "="

In the expression `x = 9` above the equals sign is causing the computer to assign the value 9 to the memory location labeled by `x`. In the way we are used to using the equal sign the expressions $x = 9$ and $9 = x$ are the same thing. This is not the case in programming languages. The second expression does not make sense to the computer and it will complain. Try it . . . now. No, really, try it. I will wait. Ok, so now that you have tried it and gotten an error message we can see what is happening. In normal algebra the equal sign is a statement of truth, a computer does not understand truth it only follows instructions. The computer interprets the equal sign as an instruction to take the value on the right side of the equal and assign that value to the memory location pointed to by the left side. This is why you get an error message when you type `9 = x`, because the left hand side, `9`, is not a memory location, so the value at location `x` cannot be assigned to `9`. Try the following

```
>> x = 5;
>> y = 2;
>> y = x
>> x
>> y
```

So we see that the original value of y gets replaced by the value of x by the statement $y = x$.

Now try the following

```
>> x = 5;
>> y = 2;
>> x = y
>> x
>> y
```

This time the original value of x gets replaced by the value of y. This is what the equal sign is used for in Matlab, it tells the computer to take the value of what ever is on the right hand side and place it in the memory location pointed to by the left hand side. It would be better to use a different symbol such as $\Leftarrow$ instead of $=$ but the use of $=$ is common in many programing languages. To clarify that it is not an equal sign in the normal sense of the word it is sometimes referred to as the *assignment operator*.

Now type the following,

```
>> n = 2
>> n = n + 1
>> n = n + 1
>> n = n + 1
```

The statement $n = n + 1$ is always false in our normal way of thinking, but is a perfectly fine command to the computer.

Every time the computer sees an equal sign it does two things. First it computes the value of the expression to the right of the equal sign. Second it assigns the resultant value to the memory location pointed to by the right hand side.

▷ Problem 1.2

Enter the following commands.

```
>> x = 1
>> x = cos(x)
>> x = cos(x)
>> x = cos(x)
>> x = cos(x)
```

Keep repeating `x = cos(x)` until you get bored. Note that the up-arrow key on the keyboard will bring up previous commands that you have typed. This is very helpful. Explain what is happening.

§ 1.4 Comparison Operators

We will see that there is an operator that is similar to our normal equal sign but first try the following commands.

```
>> 2 < 3
>> 3 < 2
>> 3 > 2
>> 2 > 3
>> 2 < 2
>> 2 <= 2
```

We see that the results of these commands are either `1` or `0`. The statements that are true return a `1` and the statements that are false return a `0`. Now try

```
>> 2 == 2
>> 2 == 3
```

We see that `==` is the closest we have to what we think of as an equal sign, it allows you to make a statement and evaluate if it is either true or false.

§ 1.5 The AND and OR operators

Recall that `pi` is automatically defined in Matlab to be the value of π. The two constants `true` and `false` are also predefined. Try the following.

```
>> true
>> false
```

We see that the 1 means true and 0 means false in Matlab. This is a common convention in programming languages. Now try

```
>> true && true
>> true && false
>> false && true
>> false && false
```

We see that the combination `x && y` is true only if both `x` and `y` are true. The `&&` operator is called the logical AND operator for this reason. Now try

```
>> true || true
>> true || false
>> false || true
>> false || false
```

and convince yourself that the `||` operator is the logical OR operator.

§ 1.6 The NOT operator

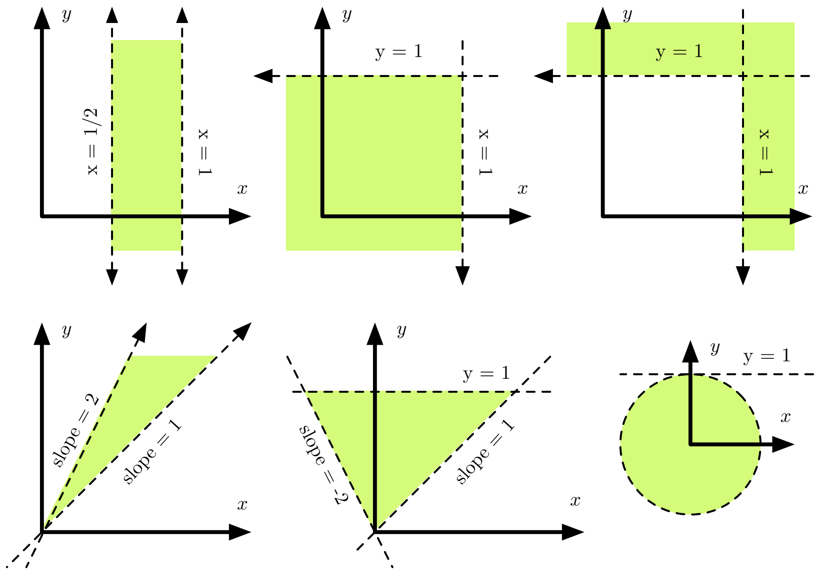
Try the following

```
>> ~true
>> ~false
```

and convince yourself the operator ∼ is the logical NOT operator.

▷ Problem 1.3

The following figure shows four shaded areas in the $x - y$ plane. For each area write a Matlab expression in the variables `x` and `y` that is 1 inside the shaded area and 0 outside that area.



§ 1.7 Arrays

Try the following

```
>> x = [0 0.1 0.2 0.4 1 2 3 4 5 6]
>> y = sqrt(x)
>> plot(x,y)
>> plot(x,y,'o')
>> x(3)
>> x(2)
>> y(6)
```

A number of things can be learned from this example. First we find that a symbol `x` can refer to a list of memory locations. In the above example x holds 10 distinct values. These values can be accessed by indicating which element in the list is wanted, for example `x(3)` refers to the third number in the list `x` so the number 0.2. It is good to think of it as a subscripted variable x_n where n refers to which element in the list we are looking at, so in this case $x_3 = 0.2$ and $x_9 = 5$. A list of numbers is referred to as an *array*.

The command `y = sqrt(x)` does two things. First it computes the square root of each value x_n in the array x. Next it set aside a section of memory as y and then assigns the square root values to this array of memory.

Lastly we see that we can produce a graph if we have two arrays of numbers.

Try the following

```
>> y(9) = 1.1
>> plot(x,y)
```

here we see that we can change a single value in an array.

There is a quick way to create a uniformly spaced array, try this.

```
>> t = -2:0.1:2
>> y = 2*t + t.^2
>> plot(t,y)
```

The unexpected thing here is the period in the expression `t.^2`. What is wanted here is a list of numbers which is the squares of the numbers in the list t. The period tells the computer that the square should be done point by point in the list. For reasons that we will understand later, if the period is left off Matlab is not able to perform the operation and an error occurs.

▷ PROBLEM 1.4

Make the following graphs

(a) The function $f(t) = \frac{1}{1+t^2}$.

(b) Three quarters of a circle or radius 2.

(c) An elipse.

(d) A spiral

(e) A graph of your signature or initials.

▷ PROBLEM 1.5

Plot the trajectory of a rock thrown from the top of a building that is 10 meters tall, if the rock is thrown at an angle of 55 degrees above the horizontal and at a speed of $30\frac{\text{m}}{\text{s}}$. How far from the building will it land? Note that the graph has a zoom function, so that you can inspect more closely the region where the rock lands.

§ 1.8 Simple Numerical Integration

There are a host of predefined functions in Matlab that operate on arrays. For example the sum function will compute the sum of the elements in an array. So if we wanted the value of the sum of the first 5 integers we could do the following.

```
>> x = 1:5;
>> sum(x)
ans =
      15
```

Recall that an integral is defined as the limit of the Riemann sum.

$$\int_a^b f(x)dx = \lim_{\Delta x_i \to 0} \sum_i f(x_i)\Delta x_i$$

Thus we can approximate an integral with the sum function, if we do the sum for a small but finite sized Δx. Let us try doing this to estimate the area of a quarter of a circle of radius 1. For a circle we know that $y^2 + x^2 = 1 \quad \longrightarrow \quad y = \sqrt{1-x^2}$. So the integral

$$\int_0^1 \sqrt{1-x^2}\ dx \approx \sum_i \sqrt{1-x_i^2}\ dx$$

that gives the area is approximated by the sum. This sum can be computed as follows.

```
>> dx = 0.01;
>> x = 0:dx:1;
>> estimate = sum(sqrt(1-x.^2)*dx)
estimate =
           0.7901
>> exact = pi/4
exact =
        0.7854
>> error = 100*(estimate-exact)/exact
error =
        0.5992
```

So we see that we get the integral with an error of about 0.6%, which is not bad for such a simple method. Ponder for a moment the ease with which you can compute the numerical value of any of the integrals that you worked so hard to learn how to solve with paper and pencil.

▷ PROBLEM 1.6

The error can be reduced quite a bit by taking the x values to be the midpoints of the intervals. Show that the midpoint Riemann sum with a step size of $dx = 0.01$ estimates the value of $\int_0^1 \sqrt{1-x^2}dx$ with an error of only 0.01%.

▷ PROBLEM 1.7
Approximate the integral $\int_0^5 e^{-x^2}\,dx$.

▷ PROBLEM 1.8
Suppose we have a net force that depends on time

$$F(t) = (20\text{N})\ln(t/(6\text{s}))$$

that is acting on an object with a mass of 10kg. Estimate the change in velocity of the mass over the time interval from $t = 7$s to $t = 13$s.

§ 1.9 Solving Linear Systems of Equations

Recall that when you write out Kirchhoff's laws for a resistor circuit that we end up with a set of linear equations in the various unknown currents in the circuit. We might end up with something like

$$\begin{aligned} 7 &= 2I_1 + 5I_2 - 3I_3 \\ 11 &= 7I_1 + 2I_2 + 4I_3 \\ 25 &= 9I_1 - 3I_2 - 8I_3 \end{aligned}$$

First we note that this can be written in matrix notation as

$$\begin{bmatrix} 7 \\ 11 \\ 25 \end{bmatrix} = \begin{bmatrix} 2 & 5 & -3 \\ 7 & 2 & 4 \\ 9 & -3 & -8 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

Look at this for a while to be sure that you understand why this is the same as the three equations above.

Now here is the cool part. If we enter the matrix

```
>> M = [2 5 -3; 7 2 4; 9 -3 -8]
```

and the column vector

```
>> V = [7; 11; 25]
```

we can compute the solution $I = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$ to this set of equations as

```
>> I = inv(M)*V
```

try it! This works because `inv(M)` computes the inverse of the matrix `M`.

```
>> Minv = inv(M)
ans =
    -0.0070 0.0861 0.0457
     0.1617 0.0193 -0.0510
    -0.0685 0.0896 -0.0545
```

The inverse is useful because the product of the matrix and its inverse is the identity matrix, the equivalent of 1 in matrix land.

```
>> Minv * M
ans =
    1 0 0
    0 1 0
    0 0 1
```

So if we multiply both sides of our original equation representing our system of equations with the inverse matrix we find the following.

$$\begin{bmatrix} -0.0070 & 0.0861 & 0.0457 \\ 0.1617 & 0.0193 & -0.0510 \\ -0.0685 & 0.0896 & -0.0545 \end{bmatrix} \begin{bmatrix} 7 \\ 11 \\ 25 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0070 & 0.0861 & 0.0457 \\ 0.1617 & 0.0193 & -0.0510 \\ -0.0685 & 0.0896 & -0.0545 \end{bmatrix} \begin{bmatrix} 2 & 5 & -3 \\ 7 & 2 & 4 \\ 9 & -3 & -8 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

and after doing the multiplication (with Matlab) we get

$$\begin{bmatrix} 2.0404 \\ 0.0703 \\ -0.8559 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

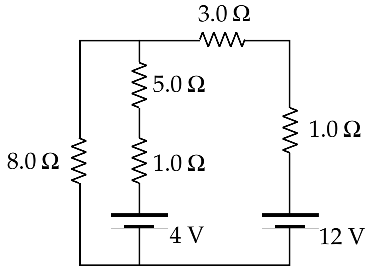
and after doing the remaining multiplication we find

$$\begin{bmatrix} 2.0404 \\ 0.0703 \\ -0.8559 \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

and we have found our three currents. So you never need to solve a system of linear equations again, you can let Matlab do it for you.

▷ Problem 1.9

Use Matlab to find the currents in the following circuit.



2 Programming

§ 2.1 Introduction

This chapter is a brief introduction to programming. Short examples will be provided that will demonstrate the fundamental elements of a program.

§ 2.2 Functions

The basic building blocks of a Matlab program are self contained pieces of code called *functions* which are stored as text files on the computer. Here is some code which defines a function.

```
function y = myFunc(x)
    if x > 0
        y = sqrt(x);
    else
        y = 0;
    end
end
```

Each line of the code is an instruction to the computer to do something. The lines are read and executed by the computer sequentially starting at the top and working down until the end of the code.

The function is used by *calling* the function from either another function of from the *command line*. For example to the right we see the function being called from the command line. The lines starting with `>>` such as `>> myFunc(2)` were typed in from the keyboard. The rest was printed out by the program. We see that the function computes the square root of the number given to it if the number is greater than zero and gives zero if the number is less than zero.

```
>> myFunc(2)

ans = 1.4142

>> myFunc(-4)

ans = 0
```

There is a number of things to learn from this example, but the most important is the format of the code. We call the variable that the function operates on the *argument* or *input variable* of the function. The argument in the example to the right is `x`. The variable that is returned is called the *return value* or *output variable*. In the example to the right is `y`. The return value must be computed someplace in the body of the code. The first line of the code defines the *interface* of the program. It tells us what the output variables are, what the name of the function is and what the input variables are.

The words in blue are *keywords* of the programming language and must be used exactly as typed. But the other words are dummy variables and can be given any name. The following code for `anything()` is identical to the code for `myFunc()`.

```
function cat = anything(dog)
    if dog > 0
        cat = sqrt(dog);
    else
        cat = 0;
    end
end
```

In general a function can have more than one return value and more than one argument. For example the following function computes the surface area A and volume V of a box of of length L, height h and width w.

```
function [A, V] = boxSize(L,h,w)
    % This function computes the
    % area and volume of a box with
    % dimensions L,h and w.
    V = L*h*w;
    sides = (L+L+w+w)*h;
    A = sides + 2*w*L;
end
```

▷ Problem 2.1
Write a function `canSize(r,h)` that computes the surface area and volume of a can that has a radius `r` and height `h`.

▷ Problem 2.2
Write a function that gives the two solutions of the quadratic equation $ax^2 + bx + c = 0$. The function will have three input variables and two output variables. The interface should look like this.

```
function [xp, xm] = quadratic(a,b,c)
```

§ 2.3 The scope of a variable

An important idea in the use of functions is the concept of the *scope* of a variable. The scope of a variable is where the variable has meaning. For example suppose that we have defined the function `boxSize` above, and that at the command line we enter

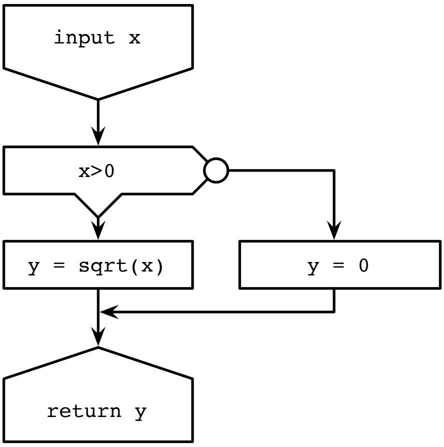
```
>> [area,volume] = boxSize(2,3,4);
>> disp(area)
    52
>> disp(sides)
Undefined function or variable 'sides'.
>> disp(A)
Undefined function or variable 'A'.
```

One might expect that the value of `sides` would be (2+2+4+4)*3 = 36 after the call to the function `boxSize`, but the way that functions work is that variable inside the function cannot be seen outside the function unless they are return values. Thus the scope of `sides` does not reach outside of the function. This limiting of scope is very important in coding, since otherwise it would be very hard to build complex programs. Similarly a variable defined outside of a function is undefined inside the function unless it is passed into the function as an input variable. Notice that the variable `A` that holds the return value is not defined outside the function either, but it's value was assigned to the variable `area` outside the function and in this way the value became available outside the function.

§ 2.4 Branches

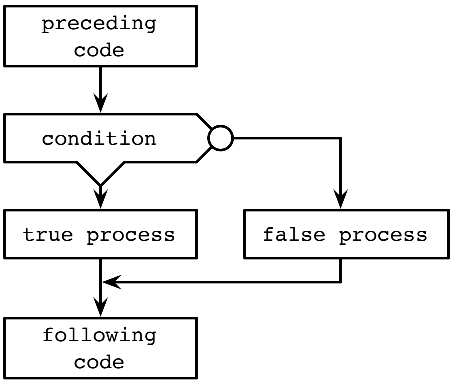
In the previous example `myFunc` we saw the use of an `if-else-end` structure. Such conditional execution of code is the beginning of intelligence in a program, as the computer is making a choice based on the conditions at the time. This structure is an example of a *branch*. It is called a branch because the program takes one of a number of alternatives, so imagine an ant climbing on a tree, when the tree branches the ant must decide which branch to take. The `if-else-end` in `myFunc` can be depicted graphical as a flow diagram.

```
function y = myFunc(x)
    if x > 0
        y = sqrt(x);
    else
        y = 0;
    end
end
```



The generic `if-else-end` can be depicted as follows.

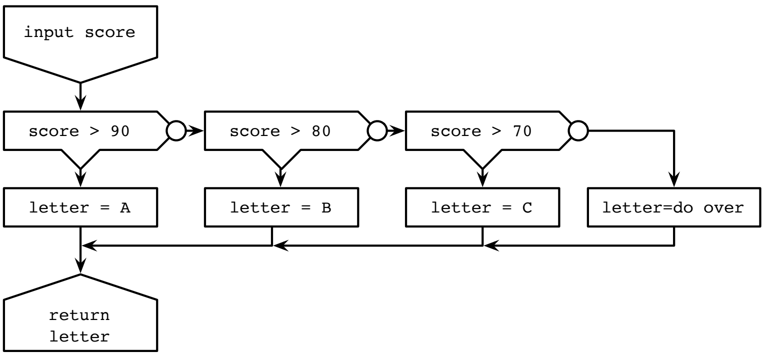
```
code before
if condition
   true code
       ⋮
else
   flase code
       ⋮
end
code after
```



The arrows in a flow diagram show the order in which the steps of the computation proceed.

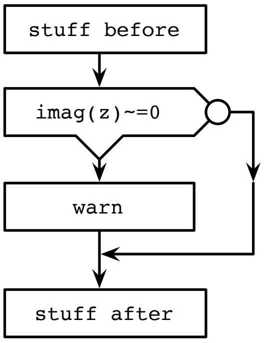
It is possible to have multiple conditions. Below is the code and flow diagram of an `if-elseif-elseif-else-end` structure.

```
function letter = grade(score)
    if score > 90
        letter = 'A';
    elseif score > 80
        letter = 'B';
    elseif score > 70
        letter = 'C';
    else
        letter = 'do over';
    end
end

```



Sometimes there is nothing to be done in the `else` branch of an `if-else-end` structure, in this case you can just leave out the `else`.

```
code before
if imag(z)~=0
  disp('WARNING: z is imaginary.')
end
code after
```

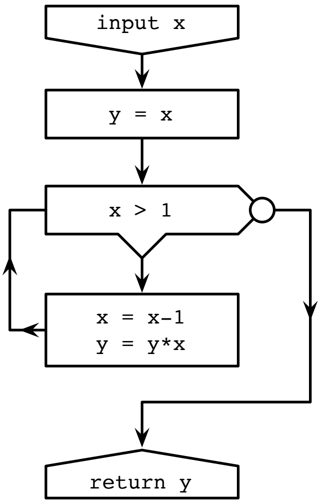


▷ PROBLEM 2.3
Write a function `cosP(x)` that returns $\cos(x)$ if $\cos(x) > 0$ and returns 0 if $\cos(x) \leq 0$.

§ 2.5 Loops

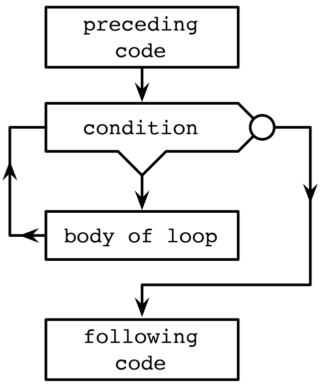
Often in a program one would like to repeat a similar operation many times. For example to compute the factorial of a number we continue multiplying by successive integers. The following code will compute the factorial.

```
function y = fact(x)
    y = x;
    while x>1
        x = x-1;
        y = y*x;
    end
end
```



The `while` in the code instructs the computer to repeat the lines of code that are between the `while` and it's closing `end` as long as the condition that is on the `while` line is true. In the program above the two lines of code `x=x-1;` and `y=y*x;` are executed repeatedly. Each time the execution returns to the beginning, the condition `x>1` is checked. If this condition is true then the code in the body of the loop is executed again. If the condition is false then the the loop is exited and the execution resumes after the end of the loop. The general while-loop structure is as follows.

```
code before
while condition
   body code
       ⋮
end
code after
```



There is a second type of loop, a `for` loop, that is useful if you know when you start the loop how many you will need to go through the loop. For example in computing the factorial 5! we know we need to do four multiplications. The factorial function could be coded using a `for` loop as follows

```
function y = fact2(N)
y = 1;
for n=2:N
    y = y*n;
end
end
```

▷ Problem 2.4
Consider the following function.

```
function y = loopy(x)
    y = 0;
    while y<x
        y = y+1
    end
end

```

Without running the function in Matlab, determine what will happen if the function is called with `loopy(4)`.

▷ PROBLEM 2.5
Write a program `solvCos(threshold)` that uses a while loop to find the solution to $x = \cos x$ that has a precision of `threshold`.

▷ PROBLEM 2.6
Write a program `plotSolvCos(N)` that makes a graph of the first N elements of the sequence defined by $x_{n+1} = \cos x_n$ with $x_1 = 1$. Also make a graph of $|x_n - x_N|$. For the second plot use the Matlab plot `semilogy()` command instead of normal `plot` command.

§ 2.6 Example: Simulating solar heating

A solar oven collects energy from the sun to heat an object. Suppose that we build an insulated box with a total surface area A and that we have a glass window of area A_g that lets the sunlight into the box. The energy radiated by the sun has an intensity (energy flux density) of up to $\phi = 1000 \frac{\mathrm{W}}{\cdot\mathrm{m}^2}$ depending on conditions, that is about 1000 Watts per square meter, or 1000 Joules per second per square meter.

Fact: **Solar energy flux**
The power absorbed by an area A is
$$P_{\text{in}} = \begin{cases} \phi A \ \cos\theta & \text{if } |\theta| < \pi/2 \\ 0 & \text{otherwise} \end{cases} = \phi A \ \texttt{cosP}(\theta)$$
where θ is the angle between the direction of the radiation and the inward normal to the area and `cosP` is the function defined in a previous homework.

At the same time heat will be leaking out of the box. We can model this as conduction through the insulated walls of the box and through the glass.

Fact: **Conductive Heat Loss**
The rate of energy loss due to conduction through one wall of the box is given by
$$P_{\text{out}} = A\frac{k}{d}(T - T_e)$$
where k is the thermal conductivity of the wall, A is the surface area of the wall, T is the temperature inside the box, T_e is the temperature outside the box, and d is the thickness of the wall. The ratio $U = \frac{k}{d}$ for a given surface is called the U-factor. In terms of the U-factor
$$P_{\text{out}} = AU(T - T_e)$$

The net energy gain of the oven is,

$$P = \frac{dE}{dt} = P_{\text{in}} - P_{\text{out}}.$$

Fact: **Heat Capacity**
The temperature of an object changes as it absorbs energy

$$\frac{dT}{dE} = \frac{1}{mc}$$

where m is the mass of the object and c is the specific heat.

Thus we can compute

$$\frac{dT}{dt} = \frac{dT}{dE}\frac{dE}{dt} = \frac{1}{mc}\frac{dE}{dt} = \frac{1}{mc}(P_{\text{in}} - P_{\text{out}})$$

Now consider the net power gain

$$\begin{aligned} P &= P_{\text{in}} - P_{\text{out}} \\ &= P_{\text{in}} - \left[P_{\text{out}}^{\text{walls}} + P_{\text{out}}^{\text{glass}}\right] \\ &= \phi A_g \cos\theta - [A_w U_w (T - T_e) + A_g U_g (T - T_e)] \\ &= \phi A_g \cos\theta - (A_w U_w + A_g U_g)(T - T_e) \end{aligned}$$

We note here that the angle of the sun changes with time

$$\theta = \theta(t) = \frac{2\pi t}{(1\text{day})} + \theta_0,$$

so that the net rate of energy gain is a function of the internal temperature T and time t.

$$P(t,T) = \phi A_g \cos(\theta(t)) - (A_w U_w + A_g U_g)(T - T_e)$$

So we end up with the equation

$$\frac{dT}{dt} = \frac{1}{mc}P(t,T)$$

We see that the rate of change of the temperature with time is a function of the temperature and the time. The solution to this equation is a function $T(t)$ that makes the equation true. This type of equation occurs in many situations and most of the time is is possible to find an approximate solution to the equation with a very simple numerical method. The method is based on the observation that if Δt is small then by the definition of the derivative

$$\frac{dx}{dt} = \frac{x(t+\Delta t) - x(t)}{\Delta t}.$$

This can be rearranged as

$$x(t+\Delta t) = x(t) + \frac{dx}{dt}\Delta t$$

This equation allows us to find the value of x at a later time, $t+\Delta t$, if we know the value of x now, t. But once we have the value at $t+\Delta t$ we can do the same thing again and again to find the value at later and later times.

Theorem: **Euler method: first order ODE**
If $\frac{dx}{dt} = f(t,x)$ then

$$t_{n+1} = t_n + \Delta t$$
$$x_{n+1} = x_n + f(t_n, x_n)\Delta t$$

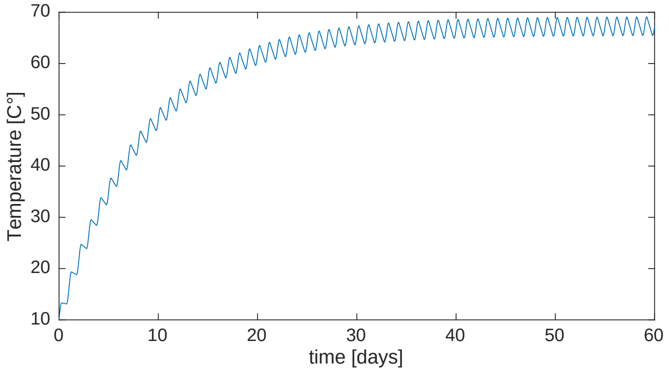
where we use the notation $x_n = x(t_n)$.

In our present case of a solar oven this becomes,

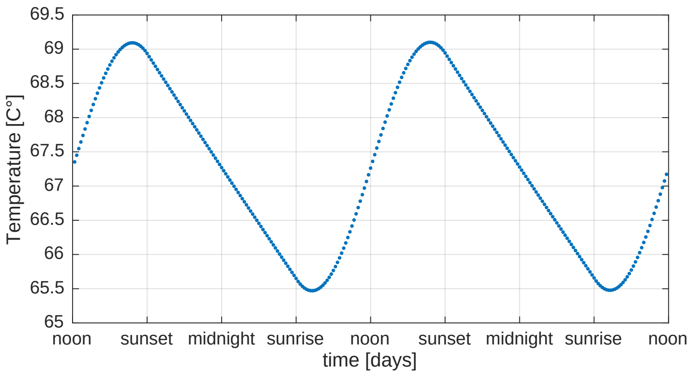
$$t_{n+1} = t_n + \Delta t$$
$$T_{n+1} = T_n + f(t_n, T_n)\Delta t$$

where $f(t,T) = \frac{1}{mc}P(t,T)$.

Below we see a graph of the simulated temperature of a solar water heater over 60 days. The heater is a one meter cube and completely filled with water ($m = 1000\text{kg}, c = 4186\text{J}/℃ \cdot \text{kg}$) at an initial temperature of 10℃, and $T_e = 10℃$, and the walls are 5cm thick styrofoam ($U = 0.66\frac{\text{W}}{\text{m}^2\cdot\text{K}}$) and a “double insulated” window ($U_g = 2.25\frac{\text{W}}{\text{m}^2\cdot\text{K}}$) covering the upward facing side of the box.



The final two days is graphed below with the times of the day marked.



In this graph you can also see that the temperature is only computed at particular times (every ten minutes, $\Delta t = 600\text{s}$).

▷ PROBLEM 2.7
Write a program to reproduce the above results for the solar water heater. Once you are satisfied that the code is working correctly adapt the code to answer the following questions.
(a) If you make the walls of the oven thicker does this make the temperature fluctuation between night and day bigger or smaller?
(b) Now suppose you build the solar oven so that the window covers half of the top of the box and that the remaining half of the top is covered in the same material as the walls. How does this effect the night/day temperature fluctuations? How does it effect the average temperature?
(c) Write a version of program that models an oven with an arbitrary wall thickness d and with a widow that covers an arbitrary fraction α of the top, that is the area of the window is $A_w = \alpha A$, where A is the area of the top.

▷ PROBLEM 2.8
Simulate a solar heated house. Assume that the house is 10 meters on a side and 3 meters tall, with a flat roof. The walls have a U-factor of $U = 0.33\frac{\text{W}}{\text{m}^2\cdot\text{K}}$, and the windows $U_g = 2.25\frac{\text{W}}{\text{m}^2\cdot\text{K}}$. There is an areas A_e of windows facing east, A_u facing upward, and A_w facing west. The heat is stored in the concrete slab that forms the floor of the house, which has a thickness d, and is well insulated below so that practically no heat is lost through the floor to the earth. The concrete has a density $\rho = 2400\frac{\text{kg}}{\text{m}^3}$ and a specific heat of $c = 750\frac{\text{J}}{\text{kg}\cdot\text{K}}$. Once the code is working, use the code to help pick the best sized windows to have a comfortable house.

§ 2.7 Solving Newton's Second Law

Suppose that we have a force acting on an object and that the force changes with time and position and velocity. For example, the force acting on an object that is falling if we included the wind resistance, this force depends on how fast the object is moving. To show that the strength of the force depends on the position, velocity and time we write the force as $F(x, v, t)$. Let the position and velocity be x_0 and v_0 at a time t_0. We can estimate the position x_1 at a time $t_1 = t_0 + \Delta t$ because if Δt is small then we can approximate

$$\frac{x_1 - x_0}{t_1 - t_0} \approx \frac{dx}{dt} = v \approx v_0 \quad \longrightarrow \quad x_1 \approx x_0 + v_0 \ \Delta t$$

and similarly

$$\frac{dv}{dt} = \frac{F}{m} \approx \frac{F(x_0, v_0, t_0)}{m} \quad \longrightarrow \quad v_1 \approx v_0 + \frac{F(x_0, v_0, t_0)}{m} \ \Delta t$$

But once we have estimates of position x_1 and velocity v_1 at the time t_1 we can use these to estimate the position and velocity at time $t_2 = t_1 + \Delta t$ by the same method.

$$\begin{aligned} x_2 &\approx x_1 + v_1 \ \Delta t \\ v_2 &\approx v_1 + \frac{F(x_1, v_1, t_1)}{m} \ \Delta t \end{aligned}$$

Indeed, we can **always** "step forward" in time to find the next estimate from the previous. All that is required is that we know the net force as a function of the position, velocity and time. This method is called Euler's Method.

Theorem: **Euler Method: second order ODE**

$$\begin{aligned} t_{n+1} &= t_n + \Delta t \\ x_{n+1} &= x_n + v_n \ \Delta t \\ v_{n+1} &= v_n + f(x_n, v_n, t_n) \ \Delta t \end{aligned}$$

with $f(x, v, t) = \frac{1}{m} F(x, v, t)$.

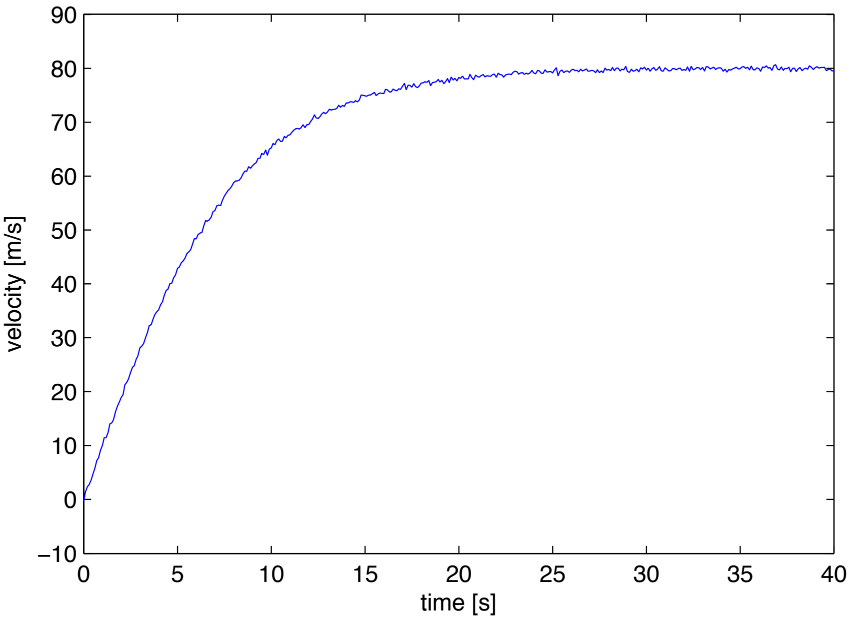
▷ PROBLEM 2.9

Use Euler's Method to find the vertical motion of an object under a constant gravitational force. Since we know the analytical solution to this problem, there is no need to solve this problem numerically, but it allows us to check our method since we already know the answer. Simulate two second of a ball thrown upward with a speed of $5.0 \frac{\text{m}}{\text{s}}$ from an initial height of 10 meters. Graph the height versus time and the theoretical height versus time on the same axis. Graph the error in height versus time. Make the above graphs for time steps of $dt = 0.1$, 0.01, 0.001, 0.0001. What is the error in the height after two seconds in each case?

▷ PROBLEM 2.10
Use Euler's method to simulate the motion of quintic oscillator $F = -kx^5$ for which there is no closed form solutions. Use a time step of 0.001 and simulate 10 seconds. Let the initial velocity be zero, and $\frac{k}{m} = 1$.
(a) If the initial position is 1, what is the period of the oscillation?
(b) If the initial position is 2, what is the period of the oscillation?
(c) For a normal harmonic oscillator the period does not change with amplitude. Is this true for the quintic oscillator?

Falling with air resistance

A rock is dropped and the velocity of the rock is recorded. The velocity versus time is graphed below.



We see that velocity increases for a while but eventually reaches a constant velocity $v_t \approx 80\frac{\mathrm{m}}{\mathrm{s}}$. We call this speed the terminal velocity. The terminal velocity of an object depends on its shape and relative size and mass. For example a ball of lead will have a higher terminal velocity than a ball of cotton.

When an object moves through the air it feels a resistive force due to the air. This drag force increases as the speed of the object increases. We will suppose that drag force is proportional to a power p of the speed.

$$F_{\text{drag}} = -\beta v^p$$

for some constant β and where we have assumed that the velocity is positive. Thus the net force on the rock can be modeled as

$$F = mg - \beta v^p$$

where we have picked down as the positive direction. So the acceleration is

$$a = \frac{F}{m} = g - \frac{\beta}{m} v^p$$

We see now that the velocity becomes a constant when the velocity gets big enough to make the acceleration zero. thus

$$0 = a = g - \frac{\beta}{m} v_t^p \longrightarrow \frac{\beta}{m} = \frac{g}{v_t^p}$$

and so we can write the acceleration as

$$a = g - \frac{\beta}{m} v^p = g - \frac{g}{v_t^p} v^p = g\left[1 - \left(\frac{v}{v_t}\right)^p\right]$$

▷ PROBLEM 2.11
On the class web site there are the data files for the recorded velocity and time. Use Euler's Method to simulated the falling of the rock with different values of p and by comparing the simulated with recorded velocities determine the actual value of p for the falling rock.

§ 2.8 Binary representation of numbers

The fundamental memory cells in a normal computer can be set to one of two states. We associate these two states with 1 and 0. Because of this numbers that are stored in a computer must be coded into a string of 1's and 0's. This representation of a number is called a binary representation. It is not really very different from the decimal representation of numbers that we are used to.

In general for an arbitrary base b an integer M can be written as

$$M = \sum_{n=0}^{N} a_n b^n.$$

With N a finite number, and the numbers a_n being integers between 0 and $b - 1$. When we write the number 753 we understand it to mean

$$753 = 3 \cdot 10^0 + 5 \cdot 10^1 + 7 \cdot 10^2 = \sum_{n=0}^{2} a_n 10^n$$

with $a_0 = 3$, $a_1 = 5$ and $a_2 = 7$.

For a binary representation the base is 2. So the numbers a_n are either 1 or 0, (and thus can be stored in a computers memory). So for example the number 201 is represented as

$$\begin{aligned} 201 &= 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + 0 \cdot 2^4 + 0 \cdot 2^5 + 1 \cdot 2^6 + 1 \cdot 2^7 \\ &= \sum_{n=0}^{7} a_n 2^n \end{aligned}$$

So

$$\begin{aligned} a_0 &= 1 \\ a_1 &= 0 \\ a_2 &= 0 \\ a_3 &= 1 \\ a_4 &= 0 \\ a_5 &= 0 \\ a_6 &= 1 \\ a_7 &= 1 \end{aligned}$$

So the number 201 can be stored in the computer as `11001001`.

$$201_{10} = 11001001_2$$

▷ Problem 2.12

The following code finds the binary representation of an integer.

```
function a = binary(x)
N = floor(log(x)/log(2));
n = N; % a counter to keep track of where we are
r = x; % the remainder
while n>=0
    p = 2^n;
    nextDigit = floor(r/p);
    r = r - nextDigit*p;
    a(n+1) = nextDigit;
    n = n-1;
end
end
```

(a) First look at the Matlab documentation to learn what the `floor` function does.

(b) Using the `binary` function as a guide, write a function `base(b,x)` that computes the representation of x in an arbitrary base b.

Loops: Representing non-integer numbers

The above representation works for number that are not integers too if we extend the sum to negative exponents.

$$\sum_{n=n_{\min}}^{n_{\max}} a_n b^n$$

where $n_{\min}$ is a negative integer. For example with $n_{\max} = 3$ and $n_{\min} = -3$ we can write

$$\begin{aligned}13.625_{10} &= 1\cdot 8 + 1\cdot 4 + 0\cdot 2 + 1\cdot 1 + 1\cdot\frac{1}{2} + 0\cdot\frac{1}{4} + 1\cdot\frac{1}{8}\\ &= 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3}\\ &= 1101.101_2\end{aligned}$$

▷ Problem 2.13

Based on your function `base(b,x)` write a function `baseFrac(b,x,Nmin)` that computes the expansion of x in powers of b down to the smallest power `Nmin` (which may be negative). That is write a function that computes the numbers a_n such that

$$x = \sum_{n=N_{\min}}^{N_{\max}} a_n b^n$$

▷ Problem 2.14

Recall that the number $\frac{1}{3}$ cannot be represented exactly in a finite length decimal number.

$$\frac{1}{3} = 0.33333\cdots$$

Notice that in base three that

$$\frac{1}{3} = 0.1_3\cdots$$

and the real number $\frac{1}{3}$ is represented exactly by a finite length number of digits. Out of idle curiosity the question arrises, are there numbers that are represented exactly in finite binary number that are not represented exactly by a finite decimal number?

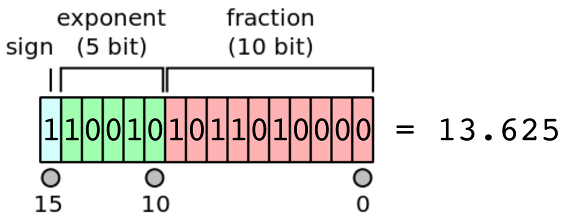
Floating point representation of real numbers

We found that $13.625_{10} = 1101.101_2$. In a floating point representation we move the decimal point over until there is only one digit before the decimal point and fix this change by multiplying by an equal number of 2s. So in the above the decimal point was moved three times so we multiply by three 2s or 2^3.

$$\begin{aligned}13.625_{10} &= 1101.101_2\\ &= 1.101101_2 \times 2^3\\ &= 1.101101_2 \times 2^{18-15}\end{aligned}$$

Similarly

▷ PROBLEM 2.15
The last of these expressions is the way that 16-bit floating point numbers are stored in a computer. The exponent, 3, is stored as $e = 3 + 15 = 18$ (it is offset to allow for negative values) in a five bit binary number $e = 10010_2 = 18_{10}$ and the digits of the number are stored in the 10 bit fractional part 1011010000. The leading 1 is not stored since it is 1 for all numbers.



The number is reconstructed via the following formula.

$$x = (-1)^s \left(1 + \sum_{n=-1}^{-10} a_n 2^n\right) 2^{e-15}$$

Adapt the code given for the binary representation so that it generates the binary floating point representation.

▷ PROBLEM 2.16
For the 16-bit floating point representation above answer the following.
(a) What is the largest number that can be represented?
(b) What is the smallest number that can be represented?
(c) What is the relative precision? The relative precision of a floating point number is the smallest number ϵ such $1 + \epsilon > 1$
(d) Using Matlab estimate the relative precision of numbers in Matlab. From this estimate the number of bits used to store the fractional part of the floating point numbers.

3 Ordinary Differential Equations

§ 3.1 Newton's Second Law: Euler's Method

Newton's Second Law tells us how the motion of an object is determined by the net force that is applied to the object. The Second Law can be written as the following coupled first order ordinary differential equation

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

where $f = \Sigma F/m$ is the ratio of the net force on the object and the mass of the object. These two equations express a relationship between the force and the motion.

A *solution* to this set of equations is two functions of time $x(t)$ and $v(t)$ that make both the equations true. The goal of Euler's Method is to build such a solution, starting from given values of $x_0 = x(t_0)$ and $v_0 = v(t_0)$ at an initial time t_0 and then estimating the values x_1 and v_1 a short time Δt later from these initial values. Once these new values are found at time $t_1 = t_0 + \Delta t$, they can in turn be used to estimate the values x_1 and v_1 at the time $t_2 = t_1 + \Delta t$. This process can be repeated generating a sequence of x_n and v_n at times t_n, which is the approximate solution.

In Euler's Method it is approximated that the rate of change of the position is a constant (the value of the velocity at the beginning of the time interval) and that the rate of change of the velocity is also a constant (the value of $f(x, v, t)$ at the beginning of the time interval). This can be written as

$$\frac{x_{n+1} - x_n}{t_{n+1} - t_n} = \frac{\Delta x}{\Delta t} \approx \left.\frac{dx}{dt}\right|_{t=t_n} = v_n$$
$$\longrightarrow x_{n+1} \approx x_n + v_n \, \Delta t$$

and

$$\frac{v_{n+1} - v_n}{t_{n+1} - t_n} = \frac{\Delta v}{\Delta t} \approx \left.\frac{dv}{dt}\right|_{t=t_n} = f(x_n, v_n, t_n)$$
$$\longrightarrow v_{n+1} \approx v_n + f(x_n, v_n, t_n) \, \Delta t$$

with $\Delta t = t_{n+1} - t_n$

Theorem: **Euler's Approximation**

$$\begin{aligned} t_{n+1} &= t_n + \Delta t \\ x_{n+1} &= x_n + v_n \Delta t \\ v_{n+1} &= v_n + f(x_n, v_n, t_n)\Delta t \end{aligned}$$

The important thing to see in the above equations is that the right hand side of each equation is a function of just the values at t_n, while the left hand side depend only on the values at t_{n+1}. Thus the equation lets us use information about the present t_n to predict the future t_{n+1}.

§ 3.2 Second order Runge-Kutta method

Euler's method is simple to implement but is not very accurate. For example in the case of a simple harmonic oscillator the amplitude of the simulated oscillation is not a constant. The second order and fourth order Runge-Kutta methods can achieve accurate results without making the time step very small.

Theorem: **Second Order Runge-Kutta**
The Euler approximation for the values of x and v at the midpoint of the time interval from t_n to t_{n+1} is

$$\begin{aligned} t_{\text{mid}} &= t_n + \tfrac{1}{2}\Delta t \\ x_{\text{mid}} &= x_n + v_n \tfrac{1}{2}\Delta t \\ v_{\text{mid}} &= v_n + f(x_n, v_n, t_n)\tfrac{1}{2}\Delta t \end{aligned}$$

where $\Delta t = t_{n+1} - t_n$. The second order Runge-Kutta uses the Euler approximation at the midpoint to estimate x and v at the end of the time interval.

$$\begin{aligned} t_{n+1} &= t_n + \Delta t \\ x_{n+1} &= x_n + v_{\text{mid}} \Delta t \\ v_{n+1} &= v_n + f(x_{\text{mid}}, v_{\text{mid}}, t_{\text{mid}})\Delta t \end{aligned}$$

▷ PROBLEM 3.1
Use both the second order Runge-Kutta and the Euler methods to simulate the first ten periods of a harmonic oscillator ($f = -\omega^2 x$) with $\omega = 2\pi$ and an amplitude of 1.
(a) Write a function `rk2HO(Ne,Nr)`. This function should make a graph of both the Euler and Runge-Kutta simulations, the Euler with a `dt` such that there are `Ne` steps per period and the Runge-Kutta with a `dt` such that there are `Nr` steps per period.
(b) If `Nr = 40`, what value of `Ne` is needed to give a similar accuracy?

§ 3.3 Fourth order Runge-Kutta method

Theorem: **Fourth Order Runge-Kutta**
The fourth order Runge-Kutta method uses the value of f at four points to determine the final step.

$$\begin{aligned} dx_a &= v_n \Delta t \\ dv_a &= f(x_n, v_n, t_n)\Delta t \end{aligned}$$

$$\begin{aligned} dx_b &= (v_n + \tfrac{1}{2} dv_a)\Delta t \\ dv_b &= f(x_n + \tfrac{1}{2} dx_a, v_n + \tfrac{1}{2} dv_a, t_n + \tfrac{1}{2}\Delta t)\Delta t \end{aligned}$$

$$\begin{aligned} dx_c &= (v_n + \tfrac{1}{2} dv_b)\Delta t \\ dv_c &= f(x_n + \tfrac{1}{2} dx_b, v_n + \tfrac{1}{2} dv_b, t_n + \tfrac{1}{2}\Delta t)\Delta t \end{aligned}$$

$$\begin{aligned} dx_d &= (v_n + dv_c)\Delta t \\ dv_d &= f(x_n + dx_c, v_n + dv_c, t_n + \Delta t)\Delta t \end{aligned}$$

These four estimates of the change are averaged to find the final values.

$$\begin{aligned} t_{n+1} &= t_n + \Delta t \\ x_{n+1} &= x_n + \frac{dx_a}{6} + \frac{dx_b}{3} + \frac{dx_c}{3} + \frac{dx_d}{6} \\ v_{n+1} &= v_n + \frac{dv_a}{6} + \frac{dv_b}{3} + \frac{dv_c}{3} + \frac{dv_d}{6} \end{aligned}$$

▷ PROBLEM 3.2
Use the fourth order Runge-Kutta, second order Runge-Kutta and the Euler methods to simulate the first ten periods of a harmonic oscillator with $\omega = 2\pi$ and an amplitude of 1.
(a) Write a function `rk4HO(Ne,Nr2,Nr4)`. This function should make a graph of the Euler, second order Runge-Kutta, and fourth order Runge-Kutta simulations, where `Ne,Nr2,Nr4` are the number of steps per period for the three different methods.
(b) If `Nr4 = 20`, what value of `Ne` and `Nr2` are needed to give a similar accuracy?
(c) Which method requires the least computation to achieve this accuracy?

▷ Problem 3.3
Consider the system with $f(x, v, t) = x$ with the initial conditions $x_0 = 1$ and $v_0 = 1$. Determine the time step needed for the Euler method and each of the two Runge-Kutta methods to achieve an accuracy of 0.002 at time $t = 1$.

§ 3.4 Physical pendulum and scaling

Consider an object that is free to rotate about a horizontal axis. Let the distance from the center of gravity of the object and the axis be L and let the moment of inertia of the object about the axis of rotation be I.

We can then write that the torque cause by gravity is $-mgL\sin\theta$ where θ is the angular position of the object, and $\theta = 0$ corresponds to the equilibrium position (with the center of gravity directly below the axis of rotation). Assuming that there are no other significant torques we may write.

$$\frac{d^2\theta}{dt^2} = \frac{-mgL\sin\theta}{I} = -\omega_0^2\sin\theta$$

where $\omega_0 = \sqrt{\frac{mgL}{I}}$.

If we use the unit-less time parameter $s = \omega_0 t$ then we can write this as.

$$\frac{d^2\theta}{ds^2} = -\sin\theta$$

In this way we can "scale out" the effect of the dimensions of the object, and get a result that will apply to all objects by rescaling with the factor ω_0.

▷ Problem 3.4
Suppose that your system has $\omega_0 = 5.0\ \mathrm{s}^{-1}$ and that you have determined that your period in the unitless parameter s is 2.7. What is the actual period in seconds?

▷ Problem 3.5
The period of the physical pendulum depends on the amplitude of the oscillation. Using the forth order Runge-Kutta, determine the period for amplitudes θ_0 from 0.01 radians to π radians in steps of size $\Delta\theta = 0.01$ radians and then graph the period as a function of the amplitude.

▷ Problem 3.6
Make a graph of the period versus amplitude for the following force per mass

$$\frac{F}{m} = -x(1 + 100e^{-5x^2})$$

4 Boundary Value Problems

§ 4.1 Introduction

The problem of solving Newton's second law is called an *initial value problem* since given the initial value of the position and velocity it is possible to compute the future values. There is another class of differential equations which are called *boundary value problems* since the value is given at the boundaries. One example is the stationary states of quantum particles trapped in a potential well. The boundary condition for this system is that the value of the wavefunction is zero at the boundaries. Between the boundaries the wave function obeys the Time Independent Schrödinger Equation. This chapter will be an investigation of a few boundary value problems.

§ 4.2 Estimating derivatives from a series

Suppose that we have a function $f(x)$ for which we have a sequence of values $f_n = f(x_n)$ where $x_n = n\Delta x$. We wish to estimate the derivatives of f from just the series of values f_n and x_n. To make a second order (in Δx) approximation to the derivatives at x_n we need only f_{n-1}, f_n and f_{n+1}. We can do this by examining the power series expansion of f about the point x_n.

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n) + \tfrac{1}{2}f''(x_n)(x - x_n)^2$$

Thus

$$f(x_{n+1}) \approx f(x_n) + f'(x_n)(x_{n+1} - x_n) + \tfrac{1}{2}f''(x_n)(x_{n+1} - x_n)^2$$

$$\longrightarrow \quad f_{n+1} \approx f_n + f'_n\Delta x + \tfrac{1}{2}f''_n\Delta x^2$$

Similarly

$$f_{n-1} \approx f_n - f'_n\Delta x + \tfrac{1}{2}f''_n\Delta x^2$$

Taking the difference of these we find that

$$f_{n+1} - f_{n-1} = 2f'_n\Delta x \;\longrightarrow\; f'_n \approx \frac{f_{n+1} - f_{n-1}}{2\Delta x}$$

Taking the sum of the same two equations we find that

$$f_{n+1} + f_{n-1} \approx 2f_n + f''_n\Delta x^2 \;\longrightarrow\; f''_n \approx \frac{f_{n+1} - 2f_n + f_{n-1}}{\Delta x^2}$$

Theorem: **Second Order Approximation of Derivatives**

$$f'_n \approx \frac{f_{n+1} - f_{n-1}}{2\Delta x}$$

$$f''_n \approx \frac{f_{n+1} - 2f_n + f_{n-1}}{\Delta x^2}$$

This gives us a way of computing the derivatives of f from the values f_n on the grid.

▷ PROBLEM 4.1
Use the above approximation to compute the first and second derivatives of $f(x) = \sin x$. Graph $f(x)$, $f'(x)$ and $f''(x)$ on the same axis. Is the approximation accurate?

§ 4.3 Boundary Value Problem

Consider a stationary states of quantum particles trapped in a potential well. The boundary condition for this system is that the value of the wavefunction is zero at the boundaries. Between the boundaries the wave function obeys the Time Independent Schrödinger Equation. In one dimension this equation can be written as follows.

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + U(x)\psi = \mathcal{E}\psi$$

where $\psi(x)$ is the wavefunction that describes the state of the particle, $U(x)$ is the potential energy of the particle, and $\mathcal{E}$ is the energy of the particle.

The above differential equation can be written as

$$\frac{d^2\psi}{dx^2} = (V(x) - E)\ \psi$$

with $V(x) = \frac{2m}{\hbar^2}U(x)$ and $E = \frac{2m}{\hbar^2}\mathcal{E}$. This form of a differential equation can be solved by the following technique. We form a grid ($x_n = n\Delta x$ and $\psi_n = \psi(x_n)$) and use the second order approximation to the derivative to write the differential equation as

$$\frac{\psi_{n+1} - 2\psi_n + \psi_{n-1}}{\Delta x^2} = (V(x_n) - E)\ \psi_n.$$

Solving this equation for ψ_{n+1} we find the right-going *recursion relation*

$$\psi_{n+1} = 2\psi_n - \psi_{n-1} + (V(x_n) - E)\ \psi_n\ \Delta x^2.$$

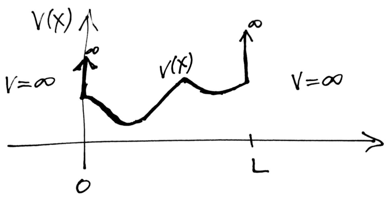
while solving for ψ_{n-1} we find the left-going recursion relation

$$\psi_{n-1} = 2\psi_n - \psi_{n+1} + (V(x_n) - E)\ \psi_n\ \Delta x^2.$$

This expression allows us to compute the value of ψ_{n+1} from the value of ψ_n and ψ_{n-1}. So if we can determine the value of two consecutive ψ_n we can compute the rest. How these two consecutive values are determined depends on the type of potential. We will deal with specific potentials below. But in all cases the idea is that the boundary condition is enforced on one boundary to get the starting values and then the recursion relation is used (assuming a particular value for the energy E) to compute all the values up to the other boundary. The computed value at the other boundary will either satisfy the boundary condition or it will not. If it does not then the energy E that was assumed is not an allowed energy. One searches for the energies E_i that do satisfy the boundary condition. The values of E_i that satisfy the boundary condition are the allowed energy levels. This is the origin of quantization.

Because the Schroödinger equation is linear in the wavefunction, if $f(x)$ is a solution to the equation then so is $g(x) = af(x)$ for any constant a. Thus the overall scale of the wavefunction is not important, and we are able to choose this scale arbitrarily.

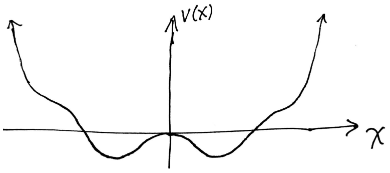
Finite Region



Sometimes the particle is confined to a certain region, for definiteness let us say that it is from $x = 0$ to $x = L$. This is equivalent to saying that the potential is infinite outside the region. Because the wavefunction is continuous we know that the wavefunction must be zero at the edges of the region, since it is zero outside the region. Thus $\psi(0) = 0$ and $\psi(L) = 0$. Thus we can take $\psi_0 = 0$. We know that $\psi_1 \neq 0$ but the question is what is the value. Well since the over all scale is unimportant we can choose it to be anything we want. For definitness sake we can let $\psi_1 = \delta x$. Now we can use the recursion relation to find the other values. Let $x_N = L$ then we are seeking values of E that lead the $\psi_N = 0$.

▷ Problem 4.2
Use the above method to find the two lowest energy stationary states of a particle confined to the region from $x = 0$ to $x = 3\pi$ with a potential in the region of $V(x) = 1 - \sin(x)$.

Symmetric Potential



In this subsection we will treat the special case that the potential is an even function $V(-x) = V(x)$. In this case the wave function ψ will be either even or odd. We shall deal with these two cases separately. In either cases we need only find $\psi(x)$ for $x \geq 0$ since we can use the symmetry of the solution to find $\psi(x)$ for $x < 0$.

Solution with odd ψ: Since the wavefunction must be continuous if it is odd then we know that $\psi_0 = 0$.

Next since the overall scale of the function is arbitrary we can pick the value of ψ_1 to be anything so we pick $\psi_1 = \Delta x$. Now we have the values of ψ_0 and ψ_1 and we can use the recursion relation to compute the rest.

Solution with even ψ: In the case that the solution is even we know that the slope at the origin is zero: $\psi_0' = 0$. Thus we know that

$$0 = \psi_0' = \frac{\psi_1 - \psi_{-1}}{2\Delta x} \longrightarrow \psi_{-1} = \psi_1.$$

Using this in the recursion relation with $n = 0$ we see that

$$\begin{aligned}
\psi_1 &= 2\psi_0 - \psi_{-1} + (V(0) - E)\ \psi_0\ \Delta x^2 \\
\longrightarrow\ \psi_1 &= 2\psi_0 - \psi_1 + (V(0) - E)\ \psi_0\ \Delta x^2 \\
\longrightarrow\ 2\psi_1 &= 2\psi_0 + (V(0) - E)\ \psi_0\ \Delta x^2 \\
\longrightarrow\ \psi_1 &= \psi_0 + \tfrac{1}{2}(V(0) - E)\ \psi_0\ \Delta x^2
\end{aligned}$$

Again we are able to pick one point arbitrarily so we set $\psi_0 = 1$ and then we can use the above result to get ψ_1, and with these two values we can then use the recursion relation to compute the rest.

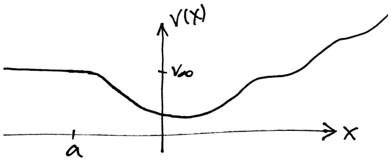
▷ PROBLEM 4.3

Use the above method to find the two lowest energy stationary states of a particle in the following potentials which are even functions of position.

(a) Harmonic oscillator potential $V(x) = x^2$.

(b) Double square well $V(x) = \begin{cases} 0 & \text{if } \left||x| - \frac{3}{2}\right| < 1 \\ 1 & \text{otherwise} \end{cases}$

Constant limiting value



In some cases the value of the potential past some point is a constant, let us call this value V_∞. In order for the particle to be in a bound state to total energy must be less than the potential at infinity: $E < V_\infty$. If the potential is a constant then we can solve the differential equation.

$$\frac{d^2\psi}{dx^2} = (V_\infty - E)\psi = \kappa^2\psi$$

where $\kappa^2 = V_\infty - E$ is a positive constant. which has the solution

$$\psi = Ae^{\kappa x} + Be^{-\kappa x}$$

Let us assume that for $x < a$ that $V(x) \approx V_\infty$. Then since we need the wave function to go to zero as x goes to $-\infty$ it must be that that the constant B is zero and so

$$\psi = Ae^{\kappa x} \qquad \text{for} \qquad x < a.$$

If we pick n such that $x_n < a$ and $x_{n+1} < a$ then

$$\psi_n = Ae^{\kappa x_n} = Ae^{\kappa n \Delta x}$$

$$\psi_{n+1} = Ae^{\kappa x_{n+1}} = Ae^{\kappa(n+1)\Delta x} = e^{\kappa \Delta x}Ae^{\kappa n \Delta x} = e^{\kappa\Delta x}\psi_n$$

Because of the arbitrariness of scale we can pick the value of ψ_n freely and then compute the value of ψ_{n+1} from $e^{\kappa\Delta x}\psi_n$.

▷ PROBLEM 4.4
Find the lowest two energy levels of the following potential.

$$V(x) = \begin{cases} \frac{1}{2}x & \text{for } x > 0 \\ 10 & \text{otherwise} \end{cases}$$

§ 4.4 Wave equation in one dimension

The equation that describes a wave traveling in a medium with a speed of propagation c is

$$\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi.$$

where $\psi(t,\vec{r})$ is the displacement of the wave at time t and position $\vec{r}$. If there is only one spacial dimension this can be written simply as

$$\frac{\partial^2\psi}{\partial t^2} = c^2\frac{\partial^2\psi}{\partial x^2}$$

where $\psi(t, x)$ is the displacement of the wave at time t and position x. This could for example describe the motion of a wave on a string, an electrical signal traveling down a wire, or a sound wave traveling down a tube. In order to solve this problem numerically we first put the solution on a grid by defining the matrix

$$\psi_n^k = \psi(t_k, x_n)$$

which gives the displacement ψ_n^k of the wave at time t_k and position x_n, with $x_n = n\Delta x$ and $t_k = k\Delta t$. Then using the second order approximation to the derivatives we can write

$$\frac{\psi_n^{k-1} - 2\psi_n^k + \psi_n^{k+1}}{\Delta t^2} = c^2 \frac{\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k}{\Delta x^2}$$

Solving this for ψ_n^{k+1} we find that

$$\psi_n^{k+1} = 2\psi_n^k - \psi_n^{k-1} + \alpha \left[\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right]$$

with $\alpha = \frac{c^2 \Delta t^2}{\Delta x^2}$. This gives us a recursion relation. This allows us to find the displacement at a given time step from the displacement at the previous two time steps.

Initial Conditions

The above equation is fine if we already have the displacement of the previous two time steps, but how do we get started in the first place. In order to have a well defined problem one needs to know not only the initial position but also the initial velocity of the surface. This is not so surprising, it is the same for projectile motion. One must know both the initial position and velocity in order to compute the position at a later time. The only thing different now is that we need to know the displacement and velocity at every position at the initial time $t = 0$. So we must assume that we know two functions over the positions, $f(x)$ and $g(x)$ or at least their discretized versions the matrixes f_n and g_n. These two functions give the initial displacement and velocities of the surface.

$$\psi_n^0 = f_n$$

$$\left[\frac{\partial \psi}{\partial t}\right]_n^0 = g_n$$

Clearly we can use the first equation to set the initial value of ψ. But we can also use the first along with the second to estimate the position at $t = t_1$, since

$$\begin{aligned} \left[\frac{\partial\psi}{\partial t}\right]_n^0 &= g_n \\ \longrightarrow\ \frac{\psi_n^1 - \psi_n^{-1}}{2\Delta t} &= g_n \\ \longrightarrow\ -\psi_n^{-1} &= -\psi_n^1 + 2g_n\Delta t \end{aligned}$$

Putting this into the recursion relation for $k = 0$ we find

$$\begin{aligned} \psi_n^1 &= 2\psi_n^0 - \psi_n^{-1} + \alpha\left[\psi_{n-1}^0 - 2\psi_n^0 + \psi_{n+1}^0\right] \\ &= 2\psi_n^0 - \psi_n^1 + 2g_n\Delta t + \alpha\left[\psi_{n-1}^0 - 2\psi_n^0 + \psi_{n+1}^0\right] \\ \longrightarrow\ \psi_n^1 &= \psi_n^0 + g_n\Delta t + \tfrac{1}{2}\alpha\left[\psi_{n-1}^0 - 2\psi_n^0 + \psi_{n+1}^0\right] \\ &= f_n + g_n\Delta t + \tfrac{1}{2}\alpha\left[f_{n-1} - 2f_n + f_{n+1}\right] \end{aligned}$$

Theorem: **Discrete wave equation in one dimension**

$$\psi_n^{k+1} = 2\psi_n^k - \psi_n^{k-1} + \alpha\left[\psi_{n-1}^k - 2\psi_n^k + \psi_{n+1}^k\right]$$

with $\alpha = \frac{c^2\Delta t^2}{\Delta x^2}$. The method is stable if $c\Delta t \leq \Delta x$. The first two steps can be computed as

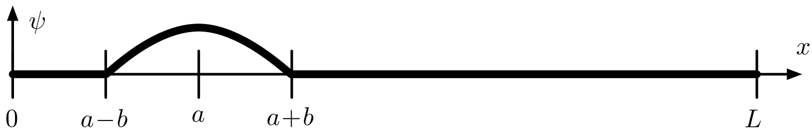
$$\begin{aligned} \psi_n^0 &= f_n, \\ \psi_n^1 &= f_n + g_n\Delta t + \tfrac{1}{2}\alpha\left[f_{n-1} - 2f_n + f_{n+1}\right]. \end{aligned}$$

where f_n is the initial displacement and g_n is the initial velocity.

▷ Problem 4.5

Find the motion of a string of length L that is plucked at a point a from one end, so that the initial shape is as pictured below, and the initial velocity is zero. Describe the motion of the string.

$$\psi(0,x) = \begin{cases} 1 - \frac{1}{b^2}(x-a)^2 & \text{if } |x-a| < b, \\ 0 & \text{otherwise.} \end{cases}$$



▷ Problem 4.6

Find the motion of a string of length L that is struck at a point a from one end, so that the initial velocity is,

$$\left.\frac{\partial\psi}{\partial t}\right|_{(0,x)} = \begin{cases} 1 - \frac{1}{b^2}(x-a)^2 & \text{if } |x-a| < b, \\ 0 & \text{otherwise.} \end{cases}$$

and the initial displacement is zero.

▷ PROBLEM 4.7

(a) Find the motion of a string of length L that has the initial displacement

$$\psi(0,x) = f(x) = \begin{cases} 1+\cos(\frac{\pi}{b}(x-a)) & \text{if } |x-a|<b, \\ 0 & \text{otherwise.} \end{cases}$$

and initial velocity

$$\left.\frac{\partial\psi}{\partial t}\right|_{(0,x)} = g(x) = -c\frac{df}{dx} = \begin{cases} c\frac{\pi}{b}\sin(\frac{\pi}{b}(x-a)) & \text{if } |x-a|<b, \\ 0 & \text{otherwise.} \end{cases}$$

(b) Repeat with the same initial displacement but with the opposite initial velocity.

$$\left.\frac{\partial\psi}{\partial t}\right|_{(0,x)} = g(x) = c\frac{df}{dx} = \begin{cases} -c\frac{\pi}{b}\sin(\frac{\pi}{b}(x-a)) & \text{if } |x-a|<b, \\ 0 & \text{otherwise.} \end{cases}$$

▷ PROBLEM 4.8

Damping is modeled by adding a velocity dependent term to the wave equation.

$$\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi - 2\beta\frac{\partial\psi}{\partial t}$$

(a) Determine the recursion relation.

(b) Add damping to your code simulating the wave equation.

§ 4.5 Wave Equation in two dimensions

The equation that describes a wave traveling on a surface with a speed of propagation c is

$$\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi = c^2\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}\right)$$

where $\psi(t,x,y)$ a function of three variables is the displacement of the surface at the position (x,y) on the surface at time t. The wave equation says that the acceleration of a point on the surface is proportional to the curvature of the surface at that point. This makes sense as we expect points that are cupped upward (positive curvature) to accelerate upward.

We wish to discretize the wave equation so that wave propagation can be simulated on a computer. First we put the solution to the wave equation on a grid by defining the matrix

$$\psi^k_{n,m} = \psi(t_k, x_n, y_m)$$

with $x_n = n\Delta x$, $y_m = m\Delta y$ and $t_k = k\Delta t$. Then using the second order approximation of the derivative we can discretize the laplacian

$\nabla^2\psi$ as

$$[\nabla^2\psi]^k_{n,m} = \left[\frac{\partial^2\psi}{\partial x^2}\right]^k_{n,m} + \left[\frac{\partial^2\psi}{\partial y^2}\right]^k_{n,m}$$
$$= \frac{\psi^k_{n+1,m} - 2\psi^k_{n,m} + \psi^k_{n-1,m}}{\Delta x^2} + \frac{\psi^k_{n,m+1} - 2\psi^k_{n,m} + \psi^k_{n,m-1}}{\Delta y^2}$$

For the sake of simplicity we will assume from here on that $\Delta x = \Delta y = \delta$, in which case we can write

$$[\nabla^2\psi]^k_{n,m} = \frac{\psi^k_{n+1,m} + \psi^k_{n-1,m} + \psi^k_{n,m+1} + \psi^k_{n,m-1} - 4\psi^k_{n,m}}{\delta^2}$$
$$= \frac{[\mathcal{L}\psi^k]_{n,m}}{\delta^2}$$

with the operator $\mathcal{L}$ defined for a matrix as follows.

Definition: $\mathcal{L}$ **operator**
For a two dimensional matrix f

$$[\mathcal{L}f]_{n,m} = f_{n+1,m} + f_{n-1,m} + f_{n,m+1} + f_{n,m-1} - 4f_{n,m}$$

Writing the discrete version of the second order time derivative also, the wave equation ($\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi$) becomes

$$\frac{\psi^{k+1}_{n,m} - 2\psi^k_{n,m} + \psi^{k-1}_{n,m}}{\Delta t^2} = \frac{c^2}{\delta^2}[\mathcal{L}\psi^k]_{n,m}$$

Solving for the $k+1$ term we arrive to the recursion relation.

$$\psi^{k+1}_{n,m} = 2\psi^k_{n,m} - \psi^{k-1}_{n,m} + \frac{c^2\Delta t^2}{\delta^2}[\mathcal{L}\psi^k]_{n,m}$$

Theorem: **Discrete Wave Equation in 2D**
A wave on a surface $\psi(t,x,y)$ can be modeled as $\psi^k_{n,m} = \psi(k\Delta t, n\Delta x, m\Delta y)$ and with

$$\psi^{k+1}_{n,m} = 2\psi^k_{n,m} - \psi^{k-1}_{n,m} + \alpha[\mathcal{L}\psi^k]_{n,m}$$

with $\alpha = \frac{c^2\Delta t^2}{\delta^2}$. The method is stable if $\alpha < 1$.

$$\psi^0_{n,m} = f_{n,m}$$
$$\psi^1_{n,m} = f_{n,m} + g_{n,m}\Delta t + \tfrac{1}{2}\alpha[\mathcal{L}f]_{n,m}$$

Were $f_{n,m} = \psi^0_{n,m}$ is the initial displacement and $g_{n,m} = [\frac{\partial\psi}{\partial t}]^0_{n,m}$ is the initial velocity.

Boundary Conditions

Notice that the discrete formula for the Laplacian fails at the edge of our matrix since it needs to look outside the matrix to the neighbor which is not there. There are different ways to deal with this.

The simplest is to assume that the surface is fixed at the edges so that the displacement there is constant. Imagine something like a rubber sheet stretched over an opening. Waves could exist on the sheet but at the edge the rubber does not move. In this case we do not need to use the recursion relation to update the displacement at the edge since we already know the value, and thus we don't need the laplacian at the edge.

The second most common assumption is that the normal component of the slope is zero at the boundary. This allows one to say that the value of the neighbor off of the edge is the same as the value of the neighbor just inside the edge.

It is possible to implement absorbing boundaries, but this is more complicated.

▷ PROBLEM 4.9
Find the motion of a square drum head (length L) that is hit in the center with a hammer with a spherical hammer head of radius R at time t = 0. The initial position is $f(x, y) = 0$ and the initial velocity is

$$g(x,y) = \begin{cases} \frac{R}{\sqrt{2}} - \sqrt{R^2 - r^2} & \text{if } r < \frac{R}{\sqrt{2}} \\ 0 & \text{otherwise} \end{cases}$$

where $r^2 = (x - L/2)^2 + (y - L/2)^2$. Write a program that simulates this problem with a grid that is N by N and for N_t iterations in time and a hammer with a head of radius R.

▷ PROBLEM 4.10
Do the previous problem but with a circular drum instead of a square one.

To create the circular drum head use the same square drum head setup but after each time step set the displacement of the drum head to zero at the circular edge of the drum.

▷ PROBLEM 4.11
Damping is modeled by adding a velocity dependent term to the wave equation.

$$\frac{\partial^2\psi}{\partial t^2} = c^2\nabla^2\psi - 2\beta\frac{\partial\psi}{\partial t}$$

(a) Show that with damping that

$$\psi^{k+1}_{n,m} = A\psi^k_{n,m} - B\psi^{k-1}_{n,m} + C[\mathcal{L}\psi^k]_{n,m}$$

with

$$A = \frac{2}{1+\beta\Delta t}, \qquad B = \frac{1-\beta\Delta t}{1+\beta\Delta t}, \qquad C = \frac{\alpha}{1+\beta\Delta t}.$$

(b) Add damping to your code simulating the wave equation.

▷ PROBLEM 4.12

It is possible to model a localized "wave packet" with wavelength λ and size b by using the initial condition

$$f(x,y) = e^{-\frac{r^2}{2b^2}} \cos(\vec{k}\cdot\vec{r})$$

$$g(x,y) = k\ c\ e^{-\frac{r^2}{2b^2}} \sin(\vec{k}\cdot\vec{r})$$

and $k = \frac{2\pi}{\lambda}$ and $\vec{k} = k(\cos\theta\hat{\imath} + \sin\theta\hat{\jmath})$, $\vec{r} = (x-x_0)\hat{\imath} + (y-y_0)\hat{\jmath}$ where θ is the direction the wave is traveling. The wave begins at the position (x_0, y_0).

(a) Model a plane wave hitting a double slit.
(b) Model a plane wave hitting a spherical mirror.
(c) Model a plane wave hitting a rough surface.

▷ PROBLEM 4.13

Model refraction by having the speed c be a function of position, that is $c = c_0/n$ where n is the index of refraction.

(a) Verify Snell's law using a simulation of a plane wave hitting a boundary between two different index of refractions.
(b) Verify the thins lens equation using a simulation of a plane wave hitting a "*lens shaped*" region with a higher index of refraction.

5 Signal Processing

§ 5.1 Data streams

There are many digital measurement systems that make repeated measurements at a regular time interval. So for example you could have the position x of some object recorded every hundredth of a second. Measurements are recorded at times

$$t_n = n\Delta t$$

where Δt is the time between measurements. The quantity

$$f_s = \frac{1}{\Delta t}$$

is called the *sample rate* or alternatively the *sample frequency*. We use the notation $x_n = x(t_n)$ for the value measured at a time t_n.

▷ PROBLEM 5.1
You are measuring a system that has a signal $x(t) = \sin(2\pi t)$.

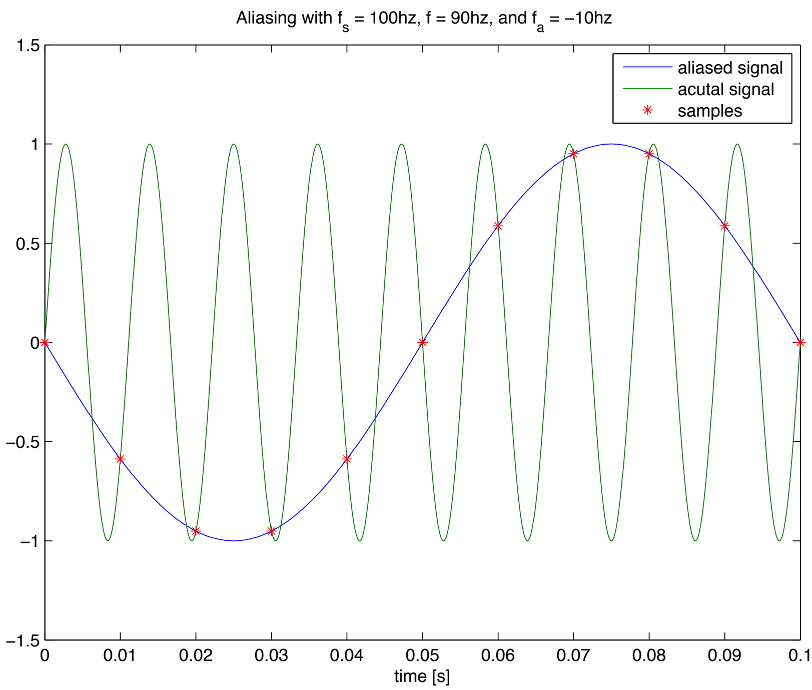
(a) Compute and plot 100 samples with a sample rate of $f_s = 20.00$Hz.

(b) Compute and plot 100 samples with a sample rate of $f_s = 2.00$Hz.

(c) Compute and plot 100 samples with a sample rate of $f_s = 0.99$Hz.

§ 5.2 Aliasing

Suppose that you sample a $f = 90$Hz sine wave at a sample rate of $f_s = 100$Hz.



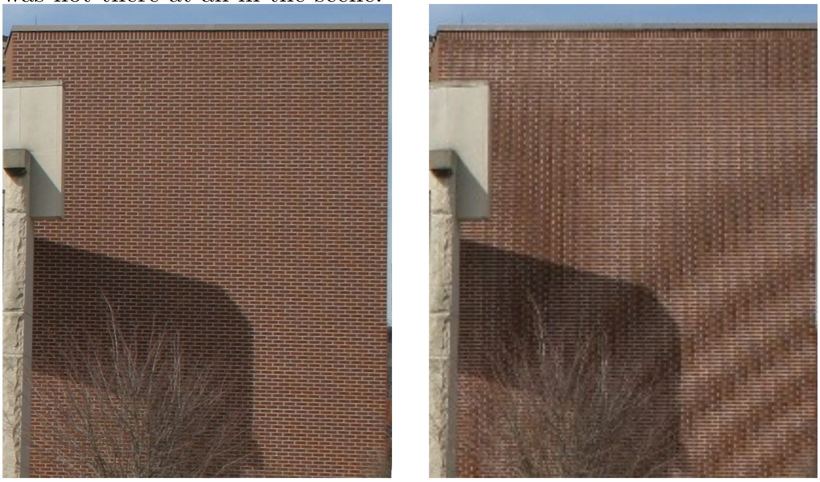
We can see in the graph that the sampled values could just as easily have come from a sine way with a frequency of $f_a = -10\text{Hz}$. We can see what is happening by considering the following. Since $t_k = k\Delta t = k\frac{1}{fs}$ we can see that $f_s t_k = k$ and thus that

$$\begin{aligned}\sin(2\pi(f \pm f_s)t_k) &= \sin(2\pi f t_k \pm 2\pi f_s t_k)\\ &= \sin(2\pi f t_k \pm 2\pi k) = \sin(2\pi f t_k)\end{aligned}$$

So we see that at the sample times t_k the sine waves with frequencies f and $f \pm f_s$ have exactly the same values. Thus the higher frequency signal appears to be lower frequency signal when it is sampled. This effect is called *aliasing*, and the lower frequency signal is said to be the *alias* of the higher frequency one.

This is a fundamental limitation of a discretely sampled signal: any frequency that is more than half the sample frequency will be aliased. In order to avoid aliasing in digital signals the original analog signal is sent through a low pass filter that removes all frequencies above half the sample frequency before the signal is sent to the analog to digital converter (ADC) to be sampled. Half the sample frequency is of enough importance that it has a name, so if someone says something about the the Nyquist frequency, he is referring to half the sample frequency. Harry Nyquist made foundational contributions to the theory of sampling in his work to understand the maximum information that could be transmitted over a telegraph line.

A good example of aliasing can be seen in the image below taken by Colin M. L. Burnett. In the high sample rate image on the left the spacial frequency of the bricks can be clearly seen. While in the low sample rate image on the right the frequency of the brick pattern is aliases to a lower frequency, making a very interesting pattern, that was not there at all in the scene.



The same effect is at play when the wheels of a wagon appear to be spinning backwards in a film.

▷ Problem 5.2

The following signals are sampled at 200Hz. What is the aliased signal in each case.

(a) $x = \sin(2\pi(130\text{Hz})t)$.
(b) $x = \sin(2\pi(237\text{Hz})t)$.
(c) $x = \sin(2\pi(320\text{Hz})t)$.
(d) $x = \sin(2\pi(450\text{Hz})t)$.
(e) $x = \sin(2\pi(-140\text{Hz})t)$.
(f) $x = \cos(2\pi(237\text{Hz})t)$.

▷ Problem 5.3

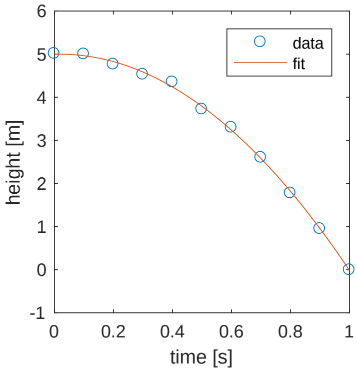
Show that $e^{i2\pi(f\pm f_s)t_k} = e^{i2\pi f t_k}$.

§ 5.3 Data fitting

Suppose that we measure the vertical position of a ball as we drop it. To the extent that gravity is the dominant force on the ball we expect the position to be

$$y = y_0 + v_0 t - \tfrac{1}{2} g t^2$$

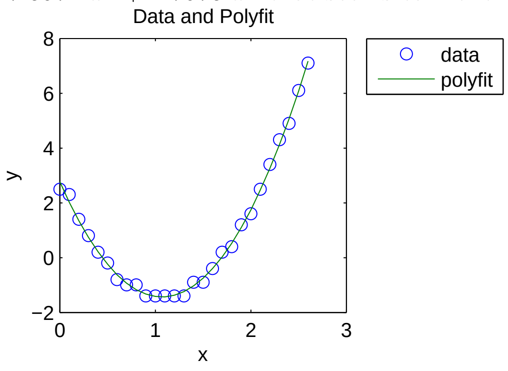
If we have a stream of vertical position data y_n measured at times t_n it will not follow this equation exactly because of measurement error. Even so if we make a graph of the data we could try plotting different parabolas $y = a_2 t^2 + a_1 t + a_0$ until we find one that seems to fit the data ok.



Then we could estimate that $y_0 = a_0$, $v_0 = a_1$ and $g = -2a_2$, and in this way ‘measure’ these properties of the motion. This sort of ‘fitting’ is a common need with experimental data, we often have a theoretical model that predict the experimental observations, and the parameters of the model are what we are really trying to determine. In the example above we could be trying to find the value of g.

The `polyfit()` function

In the graph below we see a set of data points (x_n, y_n), each data point graphed as a circle. Also in the graph has been plotted a parabola $3.6729\ x^2 - 7.8571\ x^1 + 2.7678\ x^0$ that seems to ‘fit’ the data.



The coefficients 3.6729, -7.8571, and 2.7678 that made the parabola fit the data were found by the Matlab function `polyfit()`. With the x_n data in the vector `x` and the y_n data in the vector `y` the function is called as follows, `p = polyfit(x,y,2)`. After this the vector `p` contains the coefficients. The number 2 tells Matlab to fit a polynomial of order two.

When using the output of the `polyfit()` function it is important to keep in mind that the coefficient are organized in the vector p with the coefficient of the highest power first. Thus if we write the polynomial as

$$y(t) = \sum_{n=0}^{N} a_n x^n$$

with N the highest power, then

$$a_n = p_{N-n+1} \qquad \text{OR} \qquad p_\ell = a_{N-\ell+1}$$

▷ Problem 5.4
The date file `carPos.m` contains a stream of data which is the measured position of a rocket powered car. The sample rate is 10Hz. Fit the data with a quadratic function and from this fitting estimate the acceleration of the car.

▷ Problem 5.5
Show that
(a) $a_n = p_{N+1-n}$.
(b) Show that $p_\ell = a_{N+1-\ell}$ is equivalent to $a_n = p_{N+1-n}$.

§ 5.4 Least squares fitting

In the previous section we did not discuss how we know if we have found the 'best fit'. There is no single answer for this, but the most commonly used criteria for the best fit is the function that minimizes the squares of the difference between the fit function and the data. To state this mathematically, suppose you have a set of data points (x_n, y_n), and that you wish to fit the data with a function $y = f(x, \vec{p})$ where $\vec{p}$ is a list of parameters for the function. We define the *mean squared error* as,

$$\epsilon(\vec{p}) = \frac{1}{N} \sum_{n=1}^{N} \left[f(x_n, \vec{p}) - y_n\right]^2 .$$

The *least squares fit* is the $\vec{p}$ that minimizes the mean squared error.

Using `fminsearch` to find the least squares fit

There is a Matlab function called `fminsearch` that can be used to find the minimum of a function. As an example consider the cosine function If we type

```
>> fminsearch(@cos,1)
ans =
        3.1416
```

we see that the `fminsearch` function successfully found that 3.1416 is a local minimum of the function. While typing

```
>> fminsearch(@cos,-1)
ans =
      -3.1416
>> fminsearch(@cos,7)
ans =
        9.4248
```

find different local minima. The first argument of `fminsearch` is the name of the function you wish to minimize and the second argument is an initial guess for the location of the minimum.

We can use `fminsearch` to find the set of parameters that minimize the mean squared error. The code below will fit a quadratic function to a set of data.

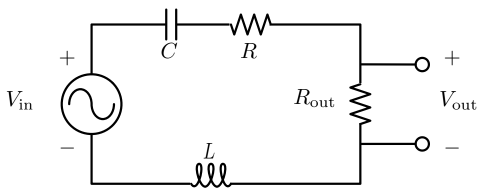
```
function p_fit = fitQuadratic(x,y)
% Fits the data to a function y = a*x^2 + b*x + c.
% p(1) = a
% p(2) = b
% p(3) = c
p_init = [1,1,0]; % Initial guess for the parameters.
p_fit = fminsearch(@compMSE,p_init);

% Here we define the mean squared error function.
% Notice that it is defined inside of fitQuadratic.
    function mse = compMSE(p)
        yfit = p(1)*x.^2 + p(2)*x + p(3);
        mse = mean((y - yfit).^2);
    end
end

```

▷ Problem 5.6

The drag force on an object caused by wind is measured at different velocities of the wind. The file `drag.mat` is an array with the velocity as the first column of the array and the drag force as the second column. Write a least squares program that fits a function $F(v) = av^b$ for the two fitting parameters a and b. Use this program to determine the parameters a and b that lead to the best fit to the data.

▷ Problem 5.7

Consider the following circuit.



The gain G of the circuit is the ratio $\frac{v_{\text{out}}}{v_{\text{in}}}$ and can be rewritten as

$$G = \left|\frac{v_{\text{out}}}{v_{\text{in}}}\right| = \frac{1}{\sqrt{\alpha^2 + \beta^2\left(\frac{f}{f_0} - \frac{f_0}{f}\right)^2}}$$

where $\alpha = 1 + \frac{R}{R_{\text{out}}}$ and $\beta = \frac{1}{R_{\text{out}}}\sqrt{\frac{L}{C}}$ and $f_0 = \frac{1}{2\pi}\sqrt{\frac{1}{LC}}$. We can see that the maximum of G is $G_{\text{max}} = \frac{1}{\alpha}$ and occurs when $f = f_0$.
(a) Write a least squares fitting function that takes a set of data (f_n, G_n) and estimates the value of α, β and f_0 and then using these values computes the values of R, L and C. In order to make the initial estimate of the parameters find the peak in G and use the peak value and location to estimate f_0 and α. Assume that the value of R_{out} is given.
(b) The data file `RLC.mat` contains an array. The first column of the array is frequency and the second column is gain. The value of $R_{\text{out}} = 2.0\Omega$ for this data set. Use your code to determine the value of R, L and C.

§ 5.5 Discrete Fourier Transform

Theorem: **Discrete Fourier Transform**
Let h_k be a sequence of N data points with k going from 0 to $N-1$. Then we can write h_k as follows.

$$h_k = \frac{1}{N}\sum_{n=0}^{N-1} H_n e^{i2\pi nk/N}$$

where the values of the constants H_n are computed as

$$H_n = \sum_{k=0}^{N-1} h_k e^{-i2\pi nk/N}$$

The H_n are said to be the *discrete Fourier transform* (DFT) of the h_k, while the h_k are said to be the *inverse discrete Fourier transform* (iDFT) of the H_n. There is an algorithm to compute the DFT and iDFT that is fast and so called the Fast Fourier Transform or FFT. So usually the DFT is referred to as the FFT and the iDFT is referred to as the iFFT. We will do so too.

We can also write the transform as

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{i2\pi f_n t_k}$$

where the values of the constants H_n are computed as

$$H_n = \sum_{k=0}^{N-1} h_k e^{-i2\pi f_n t_k}$$

where $t_k = k\frac{1}{f_s} = k\Delta t$, and $f_n = n\frac{f_s}{N} = n\Delta f$, where the *frequency resolution* is defined to be $\Delta f = \frac{f_s}{N}$.

▷ PROBLEM 5.8

Using a sample rate of 1024 samples per second, you record 128 samples of the following signals. Use the matlab function fft() to compute the transform H_n of the signal h_k. Does the resultant transform make sense, explain in terms of Euler's formula $e^{i\theta} = \cos\theta + i\sin\theta$.

(a) 200 Hz sine wave.

(b) 200 Hz cosine wave.

(c) 160 Hz sine wave + 200 Hz sine wave with twice the amplitude.

▷ PROBLEM 5.9

Using a sample rate of 1024 samples per second, you record 128 samples of two signals $h(t) = e^{i2\pi f_a t}$ and $h(t) = e^{i2\pi f_b t}$ with $f_a = 200$Hz and $f_b = 204$Hz. Use the matlab function `fft()` to compute the transforms H_n of the two signals. Compare the two transforms.

▷ PROBLEM 5.10

You have a constant stream of audio signal sampled at a rate $f_s = 44.2$kHz. You are making a display to monitor the frequency components of the audio signal. To do so you break the data stream into chunks of N samples. For each chunk of N samples you plot the magnitude of the FFT of the chunk of data $|H_n|$ versus f_n.

(a) What is the update period of the graph, that is how long does it take to receive enough new data to make a chunk?

(b) If you want an update period of at most 0.1 sec what is the limit of N?

(c) If you want your graph to have a frequency resolution at most 2Hz what is the limit of N?

(d) Can you configure your system so that you have both and update period of 0.1 sec and a frequency resolution of 2Hz? If not would changing the sample rate help?

▷ PROBLEM 5.11

Suppose that you know the signal is a sinusoid of frequency f_m, that is $h(t) = A\cos(2\pi f_m t + \phi)$, but you do not know the amplitude A or the phase ϕ. How could you use the `fft()` function to find the amplitude and phase?

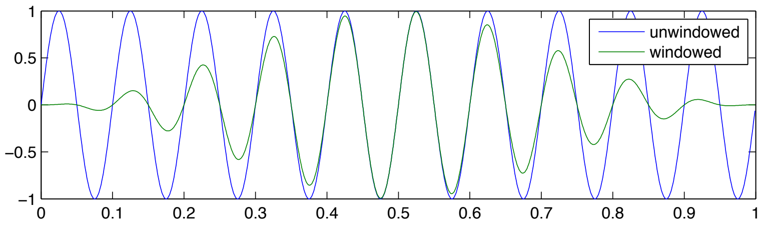
§ 5.6 Windowing

In the previous problem we saw that the transform of a 204 hz signal was dispersed over many frequencies while the transform of the 200 Hz signal was all at one frequency. This makes the transform a bit uneven in it's response. Some frequencies are treated differently from others. This problem can be corrected by multiplying the signal by a windowing function before the transform is done. There are many possible windowing function with different strengths and weaknesses. Here we will use the commonly employed Hann function which is predefined in matlab: `w=hann(N)`.

$$w_k = \sin^2\left(\frac{k\pi}{N}\right) \qquad \text{for} \quad k = 0 \text{ to } (N-1)$$

If we multiply our signal h_k by this function we get a new signal g_k that goes to zero at the ends.

$$g_k = w_k h_k$$



▷ Problem 5.12
Using a sample rate of 1024 samples per second, you record 128 samples of two signals $h(t) = e^{i2\pi f_a t}$ and $h(t) = e^{i2\pi f_b t}$ with $f_a = 200\text{Hz}$ and $f_b = 204\text{Hz}$.
(a) Window the data with the Hann function and then compute the two transforms.
(b) Compute the two transforms without windowing.
(c) Compare and contrasty the four transforms. What is gained by windowing? What is lost by windowing?

▷ Problem 5.13
Using the `audiorecorder` facility of Matlab, write a program that record a sound and then graphs the signal versus time and the coefficients H_n versus f_n. Use a windowed version of the data to compute the FFT. Record various sounds, thumping a book, knocking on a desk, crinkling paper, clapping, a voice, humming, whistling, etc. Can you see a difference between sounds that have a tone and those that do not?

A Hints

1.3 The region above a line defined by $y = mx + b$ is defined by $y > mx + b$.

1.4
(a) You will need to use a `./` and a `.^2`.
(b) Using polar coordinates $x = r\cos\theta$ and $y = r\sin\theta$. So if we let `theta = 0:0.1:3*pi/2` we can compute the x and y coordinates for all of the angles from zero to $3\pi/2$.
(c) An ellipse is a circle with a different radius in the x and y directions: $x = r_x \cos\theta$ and $y = r_y \sin\theta$.
(d) A spiral is a circle with an increasing radius. $x = \theta\cos\theta$ and $y = \theta\sin\theta$.
(e) Write you signature in large letters on graph paper. Pick important points along the signature and find the coordinates of these points, enter these coordinates into `x` and `y` arrays in Matlab. Plot `y` versus `x`.

1.5 $x(t) = 30\cos(55°)t$ and $y(t) = 10 + 30\sin(55°)t - \frac{1}{2}(9.8)t^2$.

1.6 What are the midpoint values? How would you generate a list of midpoint values? What would `x = (dx/2):dx:1` do?

1.8 $\Delta v = \int_{t_i}^{t_f} a\ dt = \int_{5\mathrm{s}}^{10\mathrm{s}} \frac{F}{m} dt \approx 6\frac{\mathrm{m}}{\mathrm{s}}$

1.9 Review Kirchhoff's laws.

2.1 Follow the `boxSize` example worked out in the notes.

2.2 Derive or find the quadratic formula.

2.3 Use the `myFunc()` code as a model.

2.4 Imagine that you are the computer executing the commands, go through step by step.

2.5 Recall the previous exercise where we found the solution to this problem. You will need to keep track of how much the estimate changes each time through the loop in order to know when to stop.

2.6 Use a `for` loop. Use an array to keep track of successive estimates.

2.8 You need to consider that the sun is at a different angle to the windows on different sides. Be clear about what time of day $t = 0$ is, then figure out at what time of day the angle to the window is zero. This should help.

2.9 The force is a constant in this case, thus it does not depend on the position, velocity or time.

2.11 In this case you do not need to keep track of the position of the falling rock, just the velocity.

2.12 A number in base b can be written as $\sum_{n=0}^{n_{\max}} a_n b^n$. The binary representation is base 2.

2.13 To Be Done

2.14 $2 = \frac{10}{2}$.

2.15 To Be Done

2.16 To Be Done

3.1 The exact solution is $x = \cos \omega$. In order to evaluate the accuracy graph the difference between the simulated value and the exact.

3.2 In order to estimate the computational cost assume that the second order Runge-Kutta takes about twice as long as the Euler for a single step, and likewise the fourth order Runge-Kutta takes four times as long as the Euler.

3.3 The exact solution is $x = e^t$. Assume that the second order Runge-Kutta takes about twice as long as the Euler for a single step, and likewise the fourth order Runge-Kutta takes four times as long as the Euler.

3.4 Use $s = \omega_0 t$.

3.5 First write a function that determines the period for a given amplitude θ_0.

3.6 To Be Done

4.1 To Be Done

4.3 Write the code that generates the even and odd solutions as outlined. The solution will depend on the energy E. Only certain energies will have the property that

$$\lim_{x\to\infty} \psi(x) = 0.$$

The allowed energies are these special energies.

4.4 To Be Done

4.8 To Be Done

4.10

4.11 Use the second order approximation to the first derivative is $\frac{df}{dt} = \frac{f(t+\Delta t) - f(t-\Delta t)}{2\Delta t}$.

5.1 $x_n = x(t_n)$ and $t_n = n\Delta t = n/f_s$.

5.2 It really is as simple as you think it is.

5.5 Write out a few example. Look for the pattern.

5.7 Show that $R = R_{\text{out}}(\alpha - 1)$ and $L = \frac{R_{\text{out}}\beta}{2\pi f_0}$ and $C = \frac{1}{R_{\text{out}}\beta 2\pi f_0}$.

5.11 To Be Done

5.12 To Be Done

5.13 To Be Done