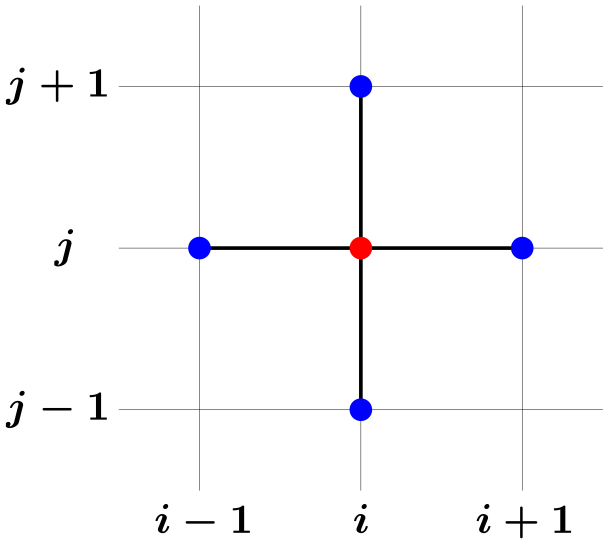
Numerical Methods for Engineers

Lecture Notes for coursera

Jeffrey R. Chasnov



THE HONG KONG UNIVERSITY OF
SCIENCE AND TECHNOLOGY

The Hong Kong University of Science and Technology
Department of Mathematics
Clear Water Bay, Kowloon
Hong Kong

Preface

View the promotional video on YouTube

These are the lecture notes for my upcoming Coursera course , *Numerical Methods for Engineers* (for release in January 2021). Before students take this course, they should have some basic knowledge of single-variable calculus, vector calculus, differential equations and matrix algebra. Students should also be familiar with at least one programming language. In this course, however, I will exclusively use MATLAB. I teach the basics of MATLAB in the first week, but this may be too short an introduction for students of limited programming ability and they may need to supplement their programming lessons elsewhere.

I have divided these notes into chapters called Lectures, with each Lecture corresponding to a video on Coursera. I have also uploaded all my Coursera videos to YouTube, and links are placed at the top of each Lecture.

There are problems at the end of each lecture, some that require analytical solutions and others that require MATLAB programs. Solutions to the analytical questions and Learner Templates for the MATLAB programs can be found in the Appendix.

On the Coursera platform, at the end of each week there is also both an assessed multiple-choice quiz and a MATLAB project. Details of the MATLAB projects and their Learner Templates can also be found in these lecture notes.

JEFFREY R. CHASNOV
Hong Kong
Nov 2020

Contents

I Scientific Computing 1

1 Binary numbers 2

2 Double precision 4

3 MATLAB as a calculator 6

4 Scripts and functions 8

5 Vectors 10

6 Line plots 13

7 Matrices 16

8 Logicals 20

9 Conditionals 22

10 Loops 24

11 Project I: Logistic map (Part A) 26

12 Project I: Logistic map (Part B) 28

II Root Finding 30

13 Bisection method 31

14 Newton's method 33

15 Secant method 35

16 Order of convergence 37

17 Convergence of Newton's method 39

18 Fractals from Newton's method 41

19 Coding the Newton fractal 43

20 Root finding in MATLAB 46

21 Project II: Feigenbaum delta (Part A) 48

22 Project II: Feigenbaum delta (Part B) 50

23 Project II: Feigenbaum delta (Part C) 51

III Matrix Algebra 53

24 Gaussian elimination without pivoting 54

25 Gaussian elimination with partial pivoting 56

26 LU decomposition with partial pivoting 58

27 Operation counts 61

28 Operation counts for Gaussian elimination 63

29 Operation counts for forward and backward substitution 65

30 Eigenvalue power method 67

31 Eigenvalue power method (example) 69

32 Matrix algebra in MATLAB 71

33 Systems of nonlinear equations 74

34 Systems of nonlinear equations (example) 76

35 Project III: Fractals from the Lorenz equations 78

IV Quadrature and Interpolation 80

36 Midpoint rule 81

37 Trapezoidal rule 83

38 Simpson's rule 85

39 Composite quadrature rules 87

40 Gaussian quadrature 89

41 Adaptive quadrature 91

42 Quadrature in MATLAB 93

43 Interpolation 95

44 Cubic spline interpolation (Part A) 97

45 Cubic spline interpolation (Part B) 99

46 Interpolation in MATLAB 102

47 Project IV: Bessel functions and their zeros 104

V Ordinary Differential Equations 106

48 Euler method 107

49 Modified Euler method 109

50 Runge-Kutta methods 111

51 Second-order Runge-Kutta methods 112

52 Higher-order Runge-Kutta methods 114

53 Higher-order odes and systems 116

54 Adaptive Runge-Kutta methods 118

55 Integrating odes in MATLAB (Part A) 120

56 Integrating odes in MATLAB (Part B) 121

57 Shooting method for boundary value problems 124

58 Project V: Two-body problem (Part A) 126

59 Project V: Two-body problem (Part B) 128

VI Partial Differential Equations 130

60 Boundary and initial value problems 131

Practice quiz: Classify partial differential equations 132

61 Central difference approximation 133

62 Discrete Laplace equation 135

63 Natural ordering 137

64 Matrix formulation 139

65 MATLAB solution of the Laplace equation (direct method) 141

66 Jacobi, Gauss-Seidel and SOR methods 144

67 Red-black ordering 146

68 MATLAB solution of the Laplace equation (iterative method) 147

69 Explicit methods for solving the diffusion equation 149

70 Von Neumann stability analysis 151

71 Implicit methods for solving the diffusion equation 153

72 Crank-Nicolson method for the diffusion equation 155

73 MATLAB solution of the diffusion equation 157

74 Project VI: Two-dimensional diffusion equation 160

Problem solutions and MATLAB learner templates 163

Week I

Scientific Computing

In this week's lectures, we learn how to program using MATLAB. *We learn how real numbers are represented in double precision and how to do basic arithmetic with* MATLAB. *We learn how to use scripts and functions, how to represent vectors and matrices, how to draw line plots, how to use logical variables, conditional statements,* `for` *loops and* `while` *loops.*

Your programming project will be to write a MATLAB *code to compute the bifurcation diagram for the logistic map.*

Lecture 1 | Binary numbers

View this lecture on YouTube

We do our arithmetic using decimals, which is a base-ten positional number system. For example, the meaning of the usual decimal notation is illustrated by

$$524.503 = 5 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1} + 0 \times 10^{-2} + 3 \times 10^{-3}.$$

Each position in a decimal number corresponds to a power of 10. A digit in this number system is defined as any of the numerals from 0 to 9, while digit is also the English word meaning a finger or a toe. Decimals probably arose from counting using ten fingers.

Computers count using two states, and this has led to the use of binary numbers, which is a base-two positional number system. Here, instead of digits, we use bits, defined as either a 0 or a 1. The meaning of a binary number is illustrated by

$$101.011 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}.$$

In both decimal and binary, certain fractions can be represented only by infinitely repeated numerals. In decimal, only fully reduced fractions whose denominators are a product of powers of 2 and 5 do not repeat. For example, we have

$$\frac{1}{2} = 0.5, \quad \frac{1}{4} = 0.25, \quad \frac{1}{5} = 0.2, \quad \frac{1}{8} = 0.125;$$

but we also have

$$\frac{1}{3} = 0.\overline{3}, \quad \frac{1}{6} = 0.1\overline{6}, \quad \frac{1}{7} = 0.\overline{142857}, \quad \frac{1}{9} = 0.\overline{1},$$

where we use the bar notation to indicate repeating numerals.

In binary, only fully reduced fractions whose denominators are a product of powers of 2 do not repeat. For example, using our more familiar digits to represent the fraction, the corresponding binary numbers are given by

$$\frac{1}{2} = 0.1, \quad \frac{1}{4} = 0.01, \quad \frac{1}{8} = 0.001,$$

and

$$\frac{1}{3} = 0.\overline{01}, \quad \frac{1}{5} = 0.\overline{0011}, \quad \frac{1}{6} = 0.00\overline{10}, \quad \frac{1}{7} = 0.\overline{001}, \quad \frac{1}{9} = 0.\overline{000111}.$$

Binary numbers with infinite repeats can not be represented exactly using a finite number of bits and are a potential source of round-off errors.

Problems for Lecture 1

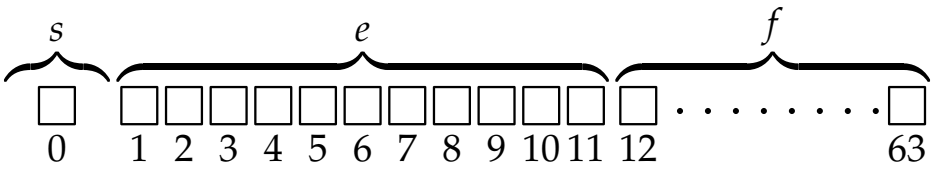
1. Using binary, round the fractions 1/3, 1/5, 1/6, 1/7 and 1/9 to six places after the binary point. Which numbers round down and which numbers round up?

Solutions to the Problems

Lecture 2 | Double precision

View this lecture on YouTube

Eight bits make a byte. Most numerical computation is done in double precision, with numbers stored using eight bytes. The format of a double precision number is



$$\# = (-1)^{\mathrm{s}} \times 2^{\mathrm{e}-1023} \times 1.\mathrm{f}$$

where s is the sign bit, e is the biased exponent, and 1.f is the significand. Here, e is a whole number written in binary, 1023 is in decimal, and f is the fractional part of the binary number following a binary point.

The 64 bits of a double precision number are distributed so that the sign uses one bit, the exponent uses 11 bits (in decimal, $0 \le e \le 2047$), and the significand uses 52 bits. The distribution of bits reconciles two conflicting needs: that the numbers should range from the very large to the very small, and that numbers should be closely spaced.

Both the largest and smallest exponents are reserved. When e is all ones ($e = 2047$ in decimal), f $= 0$ is used to represent infinity (written in MATLAB as `Inf`) and f $\neq 0$ is used to represent 'not a number' (written in MATLAB as `NaN`). `NaN` typically results from a $0/0$, ∞/∞ or $\infty - \infty$ operation. When e is all zeros, the double precision representation changes from $1.f$ to $0.f$, allowing these denormal numbers to gracefully underflow towards zero. The largest positive double precision number is `realmax = 1.7977e+308`, and the smallest positive normal number is `realmin = 2.2251e-308`.

Another important number is called machine epsilon (written in MATLAB as `eps`) and is defined as the distance between one and the next largest machine number. If $0 \le \delta <$ `eps`$/2$, then $1 + \delta = 1$ in computer arithmetic. And since

$$x + y = x(1 + y/x),$$

when $y/x <$ `eps`$/2$, then $x + y = x$. In double precision, machine epsilon is equal to `eps = 2.2204e-16`. Note that the spacing between numbers is uniform between powers of 2, but changes by a factor of two with each additional power of two. For example, the spacing of numbers between one and two is `eps`, between two and four is `2*eps`, between four and eight is `4*eps`, and so on.

We have already learned that not all rational numbers can be represented exactly in double precision. For example, the numbers $1/3$ and $1/5$ are inexact. In most computations this doesn't matter. But one needs to be aware that a calculation using inexact numbers that should result in an integer (such as zero) may not because of roundoff errors.

Problems for Lecture 2

1. Determine the double precision formats of the numbers 1, 1/2 and 1/3.

2. Using the format of a double precision number, determine the largest machine number `realmax`.

3. Using the format of a double precision number, determine the smallest positive normal machine number `realmin`.

4. Using the format of a double precision number, determine the distance between one and the next largest machine number.

Solutions to the Problems

Lecture 3 | MATLAB as a calculator

View this lecture on YouTube

The basic arithmetic operators are the usual `+`, `-`, `*`, `/`, `^`. The single quote `'` transposes a matrix. The order of operations follow the standard mathematical rules, and parenthesis can be used. Common mathematical functions are `sin`, `cos`, `tan`, `exp`, `log`. The base-10 logarithm is `log10`. The square-root is `sqrt` and the absolute value is `abs`. There is a constant `pi`, though one will need to define `e=exp(1)`. Complex numbers use the imaginary unit `i` or `j`. Some examples are

```
>> log(exp(1))
ans =
     1
>> (sqrt(5)+1)/2
ans =
     1.6180
>>
```

MATLAB also has two special numbers called `Inf` (for infinity) and `NaN` (for not-a-number). These special numbers commonly occur when your program either has a bug or some other run-time error. Simple occurrences of `Inf` and `NaN` are

```
>> 1/0
ans =
     Inf
>> 0/0
ans =
     NaN
>>
```

A semicolon at the end of a command suppresses output. Two commands can also be placed on one line, using either a comma or a semicolon.:

```
>> x=0; y=1;
>> x,y
x =
     0
y =
     1
>>
```

Problems for Lecture 3

1. Use MATLAB to compute the following expressions.

a) $\dfrac{\sqrt{5}-1}{2}$

b) $\dfrac{\left(\frac{\sqrt{5}+1}{2}\right)^{10}-\left(\frac{\sqrt{5}-1}{2}\right)^{10}}{\sqrt{5}}$

c) $\dfrac{2^5}{2^5-1}$

d) $\left(1-\dfrac{1}{2^5}\right)^{-1}$

e) e^3

f) $\ln\left(e^3\right)$

g) $\sin\left(\pi/6\right)$

h) $\cos\left(\pi/6\right)$

i) $\sin^2\left(\pi/6\right)+\cos^2\left(\pi/6\right)$

Solutions to the Problems

Lecture 4 | Scripts and functions

View this lecture on YouTube

For a quick solution to a problem, one can type directly into the Command Window. But when you want to type a longer sequence of commands, or write a program that will be run several times, you are better off writing either a script or a function. These saved programs are called m-files because of their .m extension.

A script is essentially a sequence of commands that can be typed directly into the Command Window, but are saved into a file. When working with a file, editing and debugging become easier, and commands can be quickly added or changed.

You can easily start a new script in the HOME tab, and save it with a name and in a directory of your choice. When developing code, many MATLAB programmers like to start a script with the commands

```
clear all; close all; clc;
```

These three commands clear the work space, close all figures, and clear the command window. It is a quick way to reset MATLAB to prevent previously work from interfering with your new script.

It is often useful for a script to print data during execution. To print to the command window or a file, you will need to learn how to use the MATLAB function `fprintf.m`.

A function is different than a script because the variables defined in a function are removed from the Workspace after the function ends. You can also create a new function in the HOME tab. When creating a new function, my Editor helpfully shows me the following:

```
function [outputArg1,outputArg2] = untitled(inputArg1,inputArg2)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
outputArg1 = inputArg1;
outputArg2 = inputArg2;
end
```

A function can optionally have both input arguments and output arguments. The title of the function should match the name of the file it is saved under. Functions may also be written inside of scripts. MATLAB calls these local functions.

In this course, we will mainly write scripts. But we will also make use of many MATLAB provided functions. After this course, you may sometimes want to download functions from the MATLAB File Exchange. Maybe you will even write a function yourself for others to use.

Problems for Lecture 4

1. The first few Fibonacci numbers F_n are given by $1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$, where starting from 2, each number is the sum of the preceding two numbers. Write a function

```
F = Fibonacci(n)
```

that returns the nth Fibonacci number. You should use Binet's formula, given by

$$F_n = \frac{\Phi^n - (-\phi)^n}{\sqrt{5}},$$

where

$$\Phi = \frac{\sqrt{5}+1}{2}, \qquad \phi = \frac{\sqrt{5}-1}{2}.$$

Because of round-off errors, your function should return a number rounded to the nearest integer using the built-in MATLAB function `round.m`.

Solutions to the Problems

Lecture 5 | Vectors

View this lecture on YouTube

MATLAB can work directly with both vectors and matrices. We discuss vectors first. A vector in MATLAB can either be a row vector or a column vector. An example of a row vector is `x=[1 2 3]`, or equivalently, `x=[1,2,3]`. On the command line,

```
>> x=[1,2,3]
x =
     1     2     3
>>
```

An example of a column vector is `x=[1;2;3]`. On the command line,

```
>> x=[1;2;3]
x =
     1
     2
     3
>>
```

Spaces or commas stay in the same row; semicolons move to the next row. Extra spaces are ignored. Row vectors can be changed to column vectors, or vice-a-versa, by the transpose operator `'`, e.g.,

```
>> x=[1;2;3]'
x =
     1     2     3
>>
```

There are some useful functions for constructing vectors. The zero row vector of size n is constructed from `x=zeros(1,n)` (and the column vector from `x=zeros(n,1)` or by using the transpose operator). The row vector of size n consisting of all ones is constructed from `x=ones(1,n)`. For example,

```
>> x=zeros(1,3)
x =
     0     0     0
>> x=ones(1,3)
x =
     1     1     1
>>
```

Another useful function constructs a vector of evenly spaced points. To construct a row vector of six evenly space points between zero and one, we can write `x=linspace(0,1,6)`. If the number of points is not specified, the default is 100. The colon operator also works,

where the spacing is specified rather than the number of points: `x=0:0.2:1`. On the command line,

```
>> x=linspace(0,1,6)
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
>> x=0:0.2:1
x =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
>>
```

If `x` is a vector, then `x*x` results in an error. If you want to perform element-wise operations, then you need to add a period (or dot) before the operator. There is a dot-multiply operator (`.*`), a dot-divide operator (`./`) and a dot-exponentiation operator (`.^`). On the command line,

```
>> x=[1 2 3];
>> x.*x
ans =
     1    4    9
>> x./x
ans =
     1    1    1
>> x.^x
ans =
     1    4    27
>>
```

All the standard built-in mathematical functions such as `cos`, `sin`, `exp`, `log`, etc. also work element-wise on vectors (and matrices, in general). For example,

```
>> x=0:pi/2:2*pi;
>> cos(x)
ans =
     1.0000    0.0000   -1.0000   -0.0000    1.0000
>>
```

Problems for Lecture 5

1. Compute the values of $\cos x$ and $\sin x$ for $x = 0$, $\pi/6$, $\pi/4$, $\pi/3$, and $\pi/2$. Write a script and use `fprintf.m` to print a nice table to the command window.

Solutions to the Problems

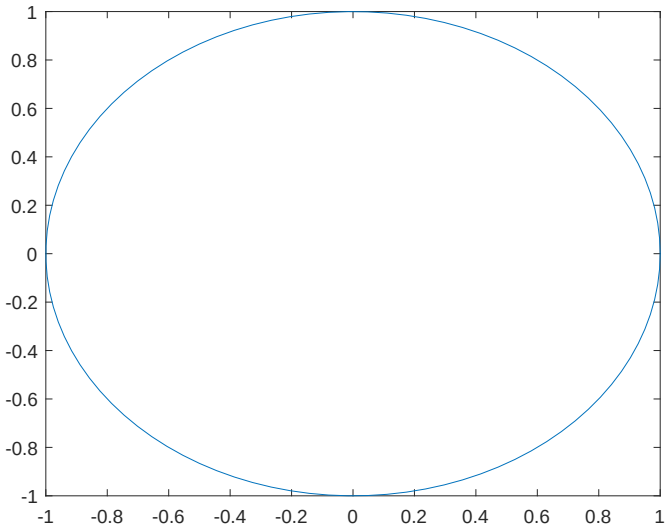
Lecture 6 | Line plots

View this lecture on YouTube

The most commonly used graphics function in MATLAB is the line plot. With `x` and `y` vectors containing the x-values and y-values of a sequence of points, a line plot connecting these points can be created using `plot(x,y)`. For example, we can plot a circle with the following script:

```
theta=linspace(0,2*pi);
x=cos(theta); y=sin(theta);
plot(x,y);
```

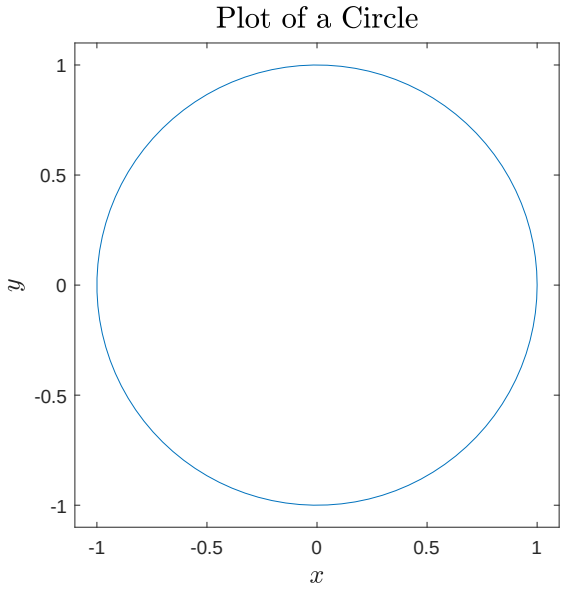
The figure that is plotted looks like this:



Actually, this plot doesn't look exactly like a circle because the unit length of the x-axis is larger than the unit length of the y-axis. We can obtain a plot that looks like a circle by typing the additional command `axis equal`. We can also try to make the plot as pretty as possible. I can show you one possibility:

```
axis equal;
axis([-1.1 1.1 -1.1 1.1]);
ax = gca;
ax.XTick = [-1 -0.5 0 0.5 1];
ax.YTick = [-1 -0.5 0 0.5 1];
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Plot of a Circle', 'Interpreter', 'latex', 'FontSize',16);
```

Here, we use `axis([xmin xmax ymin ymax])` to set the limits on the axis. The command `ax = gca` returns what MATLAB calls the handle to the current axes for the current figure. Its use here allows us to specify the location of the tick marks on the x- and y-axes. To label the axes, I use the `xlabel` and `ylabel` commands, and rather than use the default font (as in `xlabel('x')`), I prefer to write mathematical symbols using latex, and to employ a larger font. The resulting graph now looks like



Matlab can also annotate the graph with text, plot unconnected symbols such as points, open circles, crosses, etc., and use various line types and colors. You will need to frequently consult the plot function in the Help Documentation to construct a graph to your liking. You will also need to use the `hold on` command if you want to add additional points or curves to an already drawn plot.

Problems for Lecture 6

1. Generate the coordinates for a logarithmic spiral, with

$$x = e^{k\theta}\cos\theta, \qquad y = e^{k\theta}\sin\theta.$$

Choose $k = 0.05$ and plot θ values from -10π to 10π using 2000 evenly spaced points.

2. Generate the coordinates of a leminscate, with

$$x = \pm\cos\theta\sqrt{2\cos(2\theta)}, \qquad y = \sin\theta\sqrt{2\cos(2\theta)}.$$

Plot θ values from $-\pi/4$ to $\pi/4$ using 1000 evenly spaced points.

Solutions to the Problems

Lecture 7 | Matrices

View this lecture on YouTube

The name MATLAB stands for Matrix Laboratory, and was created to work with matrices. One can build matrices element-by-element:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>>
```

One can then access the matrix elements of A directly. For example, the element in row m, column n is accessed as `A(m,n)`. Submatrices of A can also be accessed. The submatrix consisting of rows i, j and columns l, m can be accessed as `A([i j], [l m])`. For example,

```
>> A(2,3)
ans =
      6
>> A([1 2], [2 3])
ans =
      2     3
      5     6
>>
```

The colon operator can also be used to good effect. The first row of matrix A can be accessed using `A(1,:)`, and the first column by `A(:,1)`:

```
>> A=[1 2 3;4 5 6;7 8 9];
>> A(1,:)
ans =
      1     2     3
>> A(:,1)
ans =
      1
      4
      7
>>
```

A matrix is actually stored in memory as a single array consisting of concatenated columns of the matrix. To view a matrix as a column vector, we can use `A(:)`.
MATLAB's `end` can also used to denote the last index of a row or column, as in

```
>> A(1,2:end)
ans =
```

```
      2      3
>>
```

One can also directly reassign matrix elements. For example,

```
>> A(1,1)=10
A =
    10      2      3
     4      5      6
     7      8      9
>>
```

MATLAB functions can also be used to build matrices. The m-by-n zero matrix is constructed from `zeros(m,n)` and the m-by-n ones matrix from `ones(m,n)`:

```
>> zeros(3,2)
ans =
      0      0
      0      0
      0      0
>> ones(2,3)
ans =
      1      1      1
      1      1      1
>>
```

The n-by-n identity matrix is constructed from `eye(n)`, and a diagonal matrix is constructed by specifying a vector for the diagonal elements, as in `diag([a b c])`. On the command line,

```
>> eye(2)
ans =
      1      0
      0      1
>> diag([1 2])
ans =
      1      0
      0      2
>>
```

The MATLAB function `diag.m` can also be used to construct banded matrices. For example, the 3-by-3 tridiagonal matrix with 2's on the diagonal and -1's directly above and below the diagonal can be constructed using

```
>> diag([2 2 2]) + diag([-1 -1],-1) + diag([-1 -1],1)
ans =
       2      -1       0
      -1       2      -1
       0      -1       2
>>
```

A more general construction for an n-by-n matrix (here, $n = 4$) can be done using the `ones.m` function:

```
>> n=4;
>> diag(2*ones(1,n)) + diag(-ones(1,n-1),-1) + diag(-ones(1,n-1),1)
ans =
       2      -1       0       0
      -1       2      -1       0
       0      -1       2      -1
       0       0      -1       2
>>
```

Matrix multiplication uses the * symbol. For example, we can multiply two two-by-two matrices:

```
>> A=[1 2;3 4]; B=[-3 -4;4 5]
>> A*B
ans =
      5      6
      7      8
>>
```

We cannot matrix multiply two row vectors, because an m-by-n matrix can only be right multiplied by an n-by-p matrix (inner matrix dimensions must agree):

```
>> x=[1 2 3];
>> x*x
Error using  *
Inner matrix dimensions must agree.
>>
```

We can, however, multiply after taking the transpose of one of the vectors. For example, the scalar product is

```
>> x=[1 2 3];
>> x*x'
ans =
      14
>>
```

Problems for Lecture 7

1. Construct the following matrix in MATLAB. Use the colon operator and the `ones.m` function:

$$A = \begin{pmatrix} 6 & 9 & 12 & 15 & 18 & 21 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 2 & 1 & 0 & -1 & -2 & -3 \\ -6 & -4 & -2 & 0 & 2 & 4 \end{pmatrix}$$

Evaluate the following expressions without using MATLAB. Check your answers with MATLAB.

a) `A([1,3],[2,4])`

b) `A(:,[1,4:6])`

c) `A([2,3],:)`

2. Construct matrices of size n-by-n that have the following characteristics:

a) Matrix A is a diagonal matrix with $1, 2, 3, \ldots, n$ along the diagonal.

b) Matrix B has ones along the superdiagonal (one above the diagonal), ones along the subdiagonal (one below the diagonal) and zeros everywhere else.

c) Matrix C has threes on the diagonal, fours on the superdiagonal, ones on two above the diagonal and zeros everywhere else.

To construct these three matrices, write the MATLAB function specified by

```
function [A, B, C] = banded_matrices(n)
```

Solutions to the Problems

Lecture 8 | Logicals

View this lecture on YouTube

A logical variable takes the value of 1 for true and 0 for false. Below is a table of logical operators used to create logical variables:

Operator	Description
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equal to
~=	not equal to
&	element-wise AND operator
&&	scalar AND operator
|	element-wise OR operator
||	scalar OR operator
~	NOT operator

For example, `p=(0<1)` will create a logical variable `p` and assign it the value of `1` because `0<1` is a true statement. Note that "equal to" is a double equal sign to distinguish it from the assignment operator.

Logical variables (with value 0 or 1) can be used in arithmetic expressions. Logical operators can also be used with vectors and matrices. The dimensions on both sides of a logical operator must agree, and the logical operator returns a vector or matrix of the same size after doing an element-by-element comparison. Here is a simple example:

```
>> x=[0 1 2 0]; y=[0 2 4 0];
>> x==y
ans =
      1      0      0      1
>>
```

Logical operators have lower precedence than arithmetic operators, so for example `p=(0<1)` can be written as `p=0<1`, and `(10/2)==5` can be written `10/2==5`. For clarity, you can always use parentheses. Logicals find their main use in conditional statements, which we discuss next.

Problems for Lecture 8

1. Evaluate the following expressions. Check your answer with MATLAB.

a) `14>15/3`

b) `8/2<5*3+1>9`

c) `8/(2<5)*3+(1>9)`

d) `2+4+3~=60/4-1`

2. Given `u=[4 -2 1]` and `v=[0 2 1]`, evaluate the following logical expressions. Check your answer with MATLAB.

a) `u<=v`

b) `u==v`

c) `u<u+v`

d) `(u<v)+u`

Solutions to the Problems

Lecture 9 | Conditionals

View this lecture on YouTube

MATLAB supports the `if ... end` structure with its variants `if ... else ... end` and `if ... elseif ... else ... end`. An expression immediately follows the `if` or `elseif` keyword. Then follows one or more statements. The statements are executed only when the expression is true. An expression is true when it is nonempty and contains only nonzero elements. An expression can be a logical, integer or real variable or array. Most commonly, the expression contains logical operators and evaluates either as true (1) or false (0).

As an example, let's write a simple MATLAB function to evaluate $\sin x/x$ with the correct limiting value when $x = 0$. Here it is:

```
function y = my_sinc(x)
if x~=0
    y=sin(x)/x;
else
    y=1;
end
```

Instead of writing `if x~=0`, some MATLAB programmers might simply write `if x` because the following statement is executed only when `x~=0`. On the command line,

```
>> sin(0)/0
ans =
      NaN
>> my_sinc(0)
ans =
      1
```

Problems for Lecture 9

1. Write two functions that return the two solutions to the quadratic equation $ax^2 + bx + c = 0$. Use something like

```
function [p, q] = quadratic_formula(a, b, c)
```

For the first function, assign

$$p = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad q = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Make sure the code works for some trial values of a, b, and c. Then try your code for $a = 1$, $b = -10^{12}$, $c = 1$. Note that $q = 0$, which is obviously not a correct root. This is an example of round-off error.

For the second function, assign

$$p = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad q = \frac{2c}{-b + \sqrt{b^2 - 4ac}} \qquad (b < 0),$$

and

$$p = \frac{2c}{-b - \sqrt{b^2 - 4ac}}, \quad q = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \qquad (b \geq 0).$$

Again, make sure the code works for some trial values of a, b, and c. Now try this code for $a = 1$, $b = -10^{12}$, $c = 1$ to obtain more reasonable results.

Solutions to the Problems

Lecture 10 | Loops

View this lecture on YouTube

Most mathematical algorithms require iteration. The workhorse for iterations is the `for` ... `end` loop. The usage is

```
for variable = expression
     statements
end
```

It is common to make use of the colon operator in the expression. For example, suppose that you want to iterate the logistic map $x_{n+1} = rx_n(1 - x_n)$ a fixed number of times. Part of your script may contain the code snippet

```
ntimes=10000;
for n=1:ntimes
     x=r*x*(1-x);
end
```

MATLAB also provides a `while` loop. The usage is

```
while expression
    statements;
end
```

The `while` loop continues as long as the expression is true. A `while` loop can be useful, for instance, when you want to iterate a sequence of commands until some event happens. For example, you may want to iterate a loop until some variable converges to a value. You can then end the loop when the change in the variable's value from the previous iteration falls below some specified threshold. For example,

```
tol=1.e-08;
error=2*tol;
while error>tol
     xold=x;
     x=r*x*(1-x);
     error=abs(x-xold)
end
```

MATLAB also has the commands `continue` and `break`. The `continue` command skips all remaining statements in the current loop iteration and proceeds to the top of the loop and the next iteration. The `break` command exits the loop entirely.

Loops are also commonly nested. Make sure that a different name for the control variable is used for each nested loop.

Problems for Lecture 10

1. Write a function that returns the nth Fibonacci number F_n for integer input n. Use the recursion relation

$$F_n = F_{n-1} + F_{n-2}; \qquad F_1 = 1, \quad F_2 = 1.$$

Handle zero and negative integers using

$$F_0 = 0, \qquad F_{-n} = (-1)^{n+1} F_n.$$

Solutions to the Problems

Lecture 11 | Project I: Logistic map (Part A)

View this lecture on YouTube

Your project for this week is to compute the bifurcation diagram for the logistic map. First, a little theory. The logistic map is a one-dimensional map, whose general form is given by

$$x_{n+1} = f(x_n). \tag{11.1}$$

A solution of a map proceeds by iteration: starting with an initial value x_0, one generates the sequence x_1, x_2, x_3, and so on.

We say that x_* is a fixed point of the map if $x_* = f(x_*)$. The stability of a fixed point can be determined by perturbation. We write $x_n = x_* + \epsilon_n$, and $x_{n+1} = x_* + \epsilon_{n+1}$, and substitute into (11.1). Using a first-order Taylor series expansion, we obtain

$$\begin{aligned} x_* + \epsilon_{n+1} &= f(x_* + \epsilon_n) = f(x_*) + \epsilon_n f'(x_*) + \dots \\ &= x_* + \epsilon_n f'(x_*) + \dots , \end{aligned}$$

which for a small enough perturbation yields

$$|\epsilon_{n+1}/\epsilon_n| = \left|f'(x_*)\right|.$$

The fixed point x_* is called stable if $|\epsilon_{n+1}| < |\epsilon_n|$ so that the perturbation decays. Therefore,

$$x_* \quad \text{is} \quad \begin{cases} \text{a stable fixed point if} & |f'(x_*)| < 1; \\ \text{an unstable fixed point if} & |f'(x_*)| > 1. \end{cases}$$

The logistic map that you will study is given by

$$x_{n+1} = \mu x_n (1 - x_n),$$

where you will assume that $0 < \mu < 4$ and $0 < x_0 < 1$.

A fixed point x_* of the logistic map satisfies the quadratic equation $x = \mu x(1 - x)$, which has two solutions given by $x_* = 0$ and $x_* = 1 - 1/\mu$. The stability of the fixed points are determined from the derivative $f'(x) = \mu(1 - 2x)$ evaluated at the fixed points, and we find that $x_* = 0$ is stable for $0 < \mu < 1$ and $x_* = 1 - 1/\mu$ is stable for $1 < \mu < 3$. For $\mu > 3$, there are no longer any stable fixed points and you will reveal the behavior of the map numerically.

Problems for Lecture 11

1. We say that x_1 and x_2 are in the orbit of a period-2 cycle of a one-dimensional map $f(x)$ if $x_2 = f(x_1)$ and $x_1 = f(x_2)$, and $x_1 \neq x_2$. Determine the orbit of a period-2 cycle for the logistic map by solving the equation $x = f(f(x))$, with $f(x) = \mu x(1-x)$. You will obtain a fourth-degree polynomial equation. Solve it by factoring out the known roots $x = 0$ and $x = 1 - 1/\mu$.

Solutions to the Problems

Lecture 12 | Project I: Logistic map (Part B)

View this lecture on YouTube

The computational engine of your MATLAB code is the iteration of the logistic map given by

$$x_{n+1} = \mu x_n (1 - x_n).$$

Your goal is to draw a diagram which illustrates the behavior of the iterates of the logistic map as a function of the parameter μ. You should discard the earliest iterations corresponding to transient behavior. The parameter μ should vary over the parameter range $2.4 \leq \mu \leq 4$.

Your main computation can contain one outer and two inner loops. Here is a reasonable outline:

Loop 1 Start at $\mu = 2.4$ and finish at $\mu = 4$.
  Set $x = x_0$.
  Loop 2 Iterate logistic map a fixed number of times (transient).
    Compute x
  Loop 2 (end)
  Loop 3 Iterate logistic map a fixed number of times (data).
    Compute and save x
  Loop 3 (end)
Loop 1 (end)

One will need to set some parameters and these can usually be adjusted after viewing a preliminary plot. Parameters include: (1) the resolution in μ; (2) the starting value x_0; (3) the number of transient iterations; (4) the number data points for each μ. For grading purposes, I will preset these parameters in the assessed code, but feel free to experiment with their values in MATLAB.

There are two possible approaches to graphing the bifurcation diagram. The simplest approach is to plot the iterates of the logistic map as points on a graph. This can serve to illustrate the bifurcation diagram but can not ultimately result in a high resolution image. The plotting of more and more data, rather than resulting in a better image, results in a blackened figure with all fine details obscured.

The best approach is to bin the iteration data in x, and to plot the bifurcation diagram by shading individual pixels of the image directly. To bin the data, you will need an additional parameter (5) the resolution in x. This portion of code is outside the scope of this exercise and will be provided to the students.

Problems for Lecture 12

1. Compute the bifurcation diagram for the logistic map. The logistic map is given by

$$x_{n+1} = \mu x_n (1 - x_n).$$

and the bifurcation diagram illustrates the behavior of the iterates of the map as a function of the parameter μ. You are tasked with writing the computational engine for this code. Here is a reasonable outline:

Loop 1 Start at $\mu = 2.4$ and finish at $\mu = 4$.
  Set $x = x_0$.
  Loop 2 Iterate logistic map a fixed number of times (transient).
    Compute x
  Loop 2 (end)
  Loop 3 Iterate logistic map a fixed number of times (data).
    Compute and save x
  Loop 3 (end)
Loop 1 (end)

Solutions to the Problems

Week II

Root Finding

In this week's lectures, we learn about root finding. We begin by learning basic numerical methods: the bisection method, Newton's method and the secant method. We then learn how fast these methods converge to the root, defining the concept of order of convergence. The order of convergence of Newton's method is determined, and students are asked in the problems to find the order of convergence of the secant method. An interesting computation of fractals from Newton's method is demonstrated using MATLAB*, and we also discuss the* MATLAB *functions that can perform root finding.*

Your programming project will be to write a MATLAB *code using Newton's method to compute the Feigenbaum delta from the bifurcation diagram for the logistic map.*

Lecture 13 | Bisection method

View this lecture on YouTube

The problem of root finding is to solve $f(x) = 0$ for a root $x = r$. The bisection method is conceptually the simplest method and almost always works. It is, however, the slowest method and because of this should usually be avoided.

The bisection method requires finding values x_0 and x_1 such that $x_0 < r < x_1$. We say that x_0 and x_1 bracket the root. With $f(r) = 0$, we want $f(x_0)$ and $f(x_1)$ to be of opposite sign, so that $f(x_0)f(x_1) < 0$. We then assign x_2 to be the midpoint of x_0 and x_1, that is $x_2 = (x_1 + x_0)/2$, which we can write in the form

$$x_2 = x_1 - \frac{x_1 - x_0}{2}.$$

Here, $-(x_1 - x_0)/2$ corrects the previous best value of the root, assumed to be x_1, and can be used as an estimate of the error. For the next iteration step, the sign of $f(x_2)$ needs to be determined, and we proceed to compute x_3 using either x_0 and x_2, or x_1 and x_2, whichever pair brackets the root. The iteration proceeds in this fashion and is typically stopped when the absolute value of the error estimate is smaller than some required precision.

Example: Estimate $\sqrt{2} = 1.41421\ldots$ *using* $x_0 = 1$ *and* $x_1 = 2$.

Now $\sqrt{2}$ is the zero of the function $f(x) = x^2 - 2$, and $f(x_0 = 1) = -1$ and $f(x_1 = 2) = 2$, so that the two initial guesses bracket the root. We iterate the algorithm. We have

$$x_2 = (1 + 2)/2 = 3/2 = 1.5.$$

Now, $f(x_2) = 9/4 - 2 = 1/4 > 0$ so that x_0 and x_2 bracket the root. Therefore,

$$x_3 = (1 + (3/2))/2 = \frac{5}{4} = 1.25.$$

Now, $f(x_3) = 25/16 - 2 = -7/16 < 0$ so that x_2 and x_3 bracket the root. Therefore,

$$x_4 = ((3/2) + (5/4))/2 = \frac{11}{8} = 1.375,$$

and so on.

Problems for Lecture 13

1. Using the bisection method, estimate $\sqrt{3} = 1.73205\ldots$. Use $x_0 = 1$ and $x_1 = 2$ and iterate to the value of x_4.

Solutions to the Problems

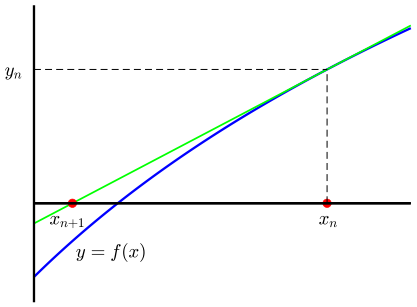
Lecture 14 | Newton's method

View this lecture on YouTube

Newton's method is the fastest method, but requires analytical computation of the derivative of $f(x)$. If the derivative is known and computational speed is required, then this method should be used, although convergence to the desired root is not guaranteed.

Newton's method approximates the function $y = f(x)$ by the tangent line to the curve at the point (x_n, y_n), where $y_n = f(x_n)$. The slope of the tangent line is $f'(x_n)$, and we have

$$y - f(x_n) = f'(x_n)(x - x_n).$$



The next approximation to the root occurs where the line intercepts the x-axis. At this point, $y = 0$ and $x = x_{n+1}$. We have $-f(x_n) = f'(x_n)(x_{n+1} - x_n)$, or

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

Newton's Method requires a guess for x_0, which should be chosen as close as possible to the root $x = r$.

Example: Estimate $\sqrt{2} = 1.41421\ldots$ *using* $x_0 = 1$.

Again, we solve $f(x) = 0$, where $f(x) = x^2 - 2$. To implement Newton's Method, we use $f'(x) = 2x$. Therefore, Newton's Method is the iteration

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{x_n^2 + 2}{2x_n}.$$

Choosing an initial guess $x_0 = 1$, we have

$$x_1 = \frac{(1)^2 + 2}{2 \times 1} = \frac{3}{2} = 1.5, \qquad x_2 = \frac{(\frac{3}{2})^2 + 2}{2 \times \frac{3}{2}} = \frac{17}{12} = 1.416667,$$

$$x_3 = \frac{(\frac{17}{12})^2 + 2}{2 \times (\frac{17}{12})} = \frac{577}{408} = 1.41426,$$

and so on. Notice how much faster Newton's method converges compared to the bisection method.

Problems for Lecture 14

1. Using Newton's method, estimate $\sqrt{3} = 1.73205\ldots$. Use $x_0 = 1$ and iterate to the value of x_3.

Solutions to the Problems

Lecture 15 | Secant method

View this lecture on YouTube

The secant method is second fastest to Newton's method, and is most often used when it is not possible to take an analytical derivative of the function $f(x)$. To derive the secant method, we begin with Newton's method,

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

and substitute an approximate numerical derivative for $f'(x_n)$:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

The Secant method requires an initial guess for both x_0 and x_1. The initial guesses need not bracket the root, but convergence of the method is not guaranteed.

Estimate $\sqrt{2} = 1.41421\ldots$ *using* $x_0 = 1$ *and* $x_1 = 2$.

Again, we solve $f(x) = 0$, where $f(x) = x^2 - 2$. The Secant Method iterates

$$\begin{aligned} x_{n+1} &= x_n - \frac{(x_n^2 - 2)(x_n - x_{n-1})}{x_n^2 - x_{n-1}^2} \\ &= \frac{x_n x_{n-1} + 2}{x_n + x_{n-1}}. \end{aligned}$$

With $x_0 = 1$ and $x_1 = 2$, we have

$$x_2 = \frac{2 \times 1 + 2}{2 + 1} = \frac{4}{3} = 1.33333, \quad x_3 = \frac{\frac{4}{3} \times 2 + 2}{\frac{4}{3} + 2} = \frac{7}{5} = 1.4,$$
$$x_4 = \frac{\frac{7}{5} \times \frac{4}{3} + 2}{\frac{7}{5} + \frac{4}{3}} = \frac{58}{41} = 1.41463,$$

and so on.

Problems for Lecture 15

1. Using the secant method, estimate $\sqrt{3} = 1.73205\ldots$. Use $x_0 = 1$ and $x_1 = 2$ and iterate to x_4.

Solutions to the Problems

Lecture 16 | Order of convergence

View this lecture on YouTube

Consider $f(x) = 0$, and suppose r is a root and x_n is the nth approximation to this root. Define the error in the nth approximation as

$$\epsilon_n = r - x_n.$$

If for large n, we have the approximate relationship

$$|\epsilon_{n+1}| = k|\epsilon_n|^p,$$

with k a positive constant and $p \geq 1$, we say the root-finding numerical method is of order p. Larger values of p correspond to faster convergence to the root. The bisection method is of order one: the error is reduced by approximately a factor of 2 with each iteration so that

$$|\epsilon_{n+1}| = \frac{1}{2}|\epsilon_n|.$$

Newton's method is the fastest converging root-finding method and is of order two. The secant method is somewhat slower and is of order approximately 1.6. In the next lecture, we show how to derive the order of Newton's method using Taylor series expansions.

Problems for Lecture 16

1. Complete the table below. Assume $|\epsilon_{n+1}| = 0.5|\epsilon_n|^p$.

	Error		
Iteration #	$p = 1$	$p = 1.6$	$p = 2$
0	1	1	1
1	0.5	0.5	0.5
2			
3			
4			
5			

Solutions to the Problems

Lecture 17 | Convergence of Newton's method

View this lecture on YouTube

We derive the rate of convergence of Newton's method. Consider $f(x) = 0$ with root r. Newton's method is given by

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

Defining $\epsilon_n = r - x_n$, and subtract both sides of Newton's iteration from r:

$$r - x_{n+1} = r - x_n + \frac{f(x_n)}{f'(x_n)}, \quad \text{which we write as} \quad \epsilon_{n+1} = \epsilon_n + \frac{f(r-\epsilon_n)}{f'(r-\epsilon_n)}.$$

We Taylor series expand the functions $f(r-\epsilon_n)$ and $f'(r-\epsilon_n)$ for small ϵ_n, using $f(r) = 0$. We have

$$f(r-\epsilon_n) = -\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \ldots, \qquad f'(r-\epsilon_n) = f'(r) - \epsilon_n f''(r) + \ldots.$$

The ellipsis $(\ldots)$ represents higher-order terms in ϵ that we will eventually neglect. We will make use of the Taylor series $1/(1-\epsilon) = 1 + \epsilon + \epsilon^2 + \ldots$, which converges for $|\epsilon| < 1$. We expand as follows:

$$\begin{aligned}
\epsilon_{n+1} &= \epsilon_n + \frac{f(r-\epsilon_n)}{f'(r-\epsilon_n)} = \epsilon_n + \frac{-\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \ldots}{f'(r) - \epsilon_n f''(r) + \ldots} \\
&= \epsilon_n + \frac{-\epsilon_n + \frac{1}{2}\epsilon_n^2 \frac{f''(r)}{f'(r)} + \ldots}{1 - \epsilon_n \frac{f''(r)}{f'(r)} + \ldots} = \epsilon_n + \left(-\epsilon_n + \frac{1}{2}\epsilon_n^2 \frac{f''(r)}{f'(r)} + \ldots\right)\left(1 + \epsilon_n \frac{f''(r)}{f'(r)} + \ldots\right) \\
&= \epsilon_n + \left(-\epsilon_n + \epsilon_n^2\left(\frac{1}{2}\frac{f''(r)}{f'(r)} - \frac{f''(r)}{f'(r)}\right) + \ldots\right) = -\frac{1}{2}\frac{f''(r)}{f'(r)}\epsilon_n^2 + \ldots.
\end{aligned}$$

We have shown that to leading order in ϵ, we have

$$|\epsilon_{n+1}| = k|\epsilon_n|^2, \quad \text{with} \quad k = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right|,$$

an expression that is valid as long as $f'(r) \neq 0$ (a so-called simple root). Because the error decreases with each iteration as the square of the previous error, we say that Newton's method is of order two.

Problems for Lecture 17

1. Take the following steps to determine the rate of convergence of the secant method, given by

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})}.$$

a) Let $\epsilon_n = r - x_n$, and subtract both sides of the secant method from r to obtain

$$\epsilon_{n+1} = \epsilon_n + \frac{(\epsilon_{n-1} - \epsilon_n)f(r - \epsilon_n)}{f(r - \epsilon_n) - f(r - \epsilon_{n-1})}.$$

b) Taylor series expand $f(r - \epsilon_n)$ and $f(r - \epsilon_{n-1})$ for small ϵ using $f(r) = 0$. Obtain

$$\epsilon_{n+1} = \epsilon_n + \frac{-\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \dots}{f'(r) - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n)f''(r) + \dots}.$$

c) For small ϵ, use the Taylor series expansion

$$\frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \dots$$

to obtain

$$|\epsilon_{n+1}| = \frac{1}{2}\left|\frac{f''(r)}{f'(r)}\right| |\epsilon_n||\epsilon_{n-1}|.$$

d) Try $|\epsilon_{n+1}| = k|\epsilon_n|^p$ and $|\epsilon_n| = k|\epsilon_{n-1}|^p$ to obtain the equation $p^2 = p + 1$. Determine p.

Solutions to the Problems

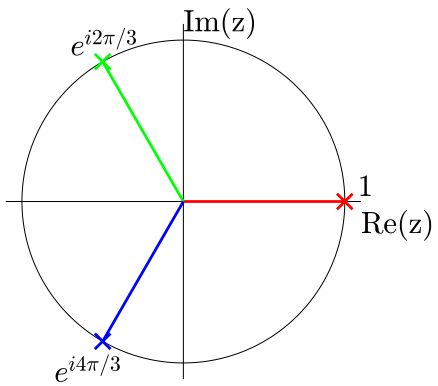
Lecture 18 | Fractals from Newton's method

View this lecture on YouTube

The three cube roots of unity are the solutions of $z^3 = 1$. We can solve this equation using Euler's formula, $\exp(i\theta) = \cos\theta + i\sin\theta$. We write $z^3 = \exp(i2\pi n)$, with n an integer, and take the cube root to find the solutions $z = \exp(i2\pi n/3)$. The three unique complex roots correspond to $n = 0, 1$ and 2, and using the trig values of special angles, we find

$$r_1 = 1, \qquad r_2 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i, \qquad r_3 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i.$$

These roots can be located on the unit circle in the complex plane, and are shown below.



We now define the root-finding problem to be $f(z) = 0$, where $f(z) = z^3 - 1$, with derivative $f'(z) = 3z^2$. Newton's method for determining the three complex roots of unity is given by the iteration

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}.$$

Here, we want to determine which initial values of z_0 in the complex plane converge to which of the three cube roots of unity. So if for a given z_0, the iteration converges to r_1, say, we will color red the point z_0 in the complex plane; if it converges to r_2, we will color z_0 green; and if to r_3, blue.

In the next lecture, we show you how to construct a MATLAB program that grids up a rectangle in the complex plane, and determines the convergence of each grid point z_0. By coloring these grid points, we will compute a beautiful fractal.

Problems for Lecture 18

1. What are the four fourth roots of unity?

Solutions to the Problems

Lecture 19 | Coding the Newton fractal

View this lecture on YouTube

Our code first defines the function $f(z)$ and its derivative to be used in Newton's method, and defines the three cube roots of unity:

```
f = @(z) z.^3-1; fp = @(z) 3*z.^2;
root1 = 1; root2 = -1/2 + 1i*sqrt(3)/2; root3 = -1/2 - 1i*sqrt(3)/2;
```

We let $z = x + iy$. The complex plane is represented as a two-dimensional grid and we define `nx` and `ny` to be the number of grid points in the x- and y-directions, respectively. We further define `xmin` and `xmax` to be the minimum and maximum values of x, and similarly for `ymin` and `ymax`. Appropriate values were determined by numerical experimentation.

```
nx=2000; ny=2000;
xmin=-2; xmax=2; ymin=-2; ymax=2;
```

The grid in x and y are defined using `linspace`.

```
x=linspace(xmin,xmax,nx); y=linspace(ymin,ymax,ny);
```

We then use `meshgrid` to construct the two-dimensional grid. Suppose that the x-y grid is defined by `x=[x1 x2 x3]` and `y=[y1 y2 y3]`. Then `[X, Y]=meshgrid(x,y)` results in the matrices

$$\texttt{X} = \begin{bmatrix} \texttt{x1} & \texttt{x2} & \texttt{x3} \\ \texttt{x1} & \texttt{x2} & \texttt{x3} \\ \texttt{x1} & \texttt{x2} & \texttt{x3} \end{bmatrix}, \qquad \texttt{Y} = \begin{bmatrix} \texttt{y1} & \texttt{y1} & \texttt{y1} \\ \texttt{y2} & \texttt{y2} & \texttt{y2} \\ \texttt{y3} & \texttt{y3} & \texttt{y3} \end{bmatrix},$$

and we can grid the complex plane using

```
[X,Y]=meshgrid(x,y);
Z=X+1i*Y;
```

We now iterate Newton's method `nit` times. These lines constitute the computational engine of the code.

```
nit=40;
for n=1:nit
    Z = Z - f(Z) ./ fp(Z);
end
```

We next test to see which roots have converged. We use the logical variables `Z1`, `Z2` and `Z3` to mark the grid points that converge to one of the three roots. Grid points that have not converged are marked by `Z4`, and our convergence criteria is set by the variable `eps`. The function `abs` returns the modulus of a complex number.

```
eps=0.001;
Z1 = abs(Z-root1) < eps; Z2 = abs(Z-root2) < eps;
Z3 = abs(Z-root3) < eps; Z4 = ~(Z1+Z2+Z3);
```

Finally, we draw the fractal. The appropriate graphing function to use here is `image`, which can color pixels directly. We first open a figure and set our desired colormap.

Here, our map will be a four-by-three matrix, where row one of the matrix corresponds with the RGB triplet `[1 0 0]` that specifies red; row two of the matrix `[0 1 0]` specifies green; row three of the matrix `[0 0 1]` specifies blue; and row four of the matrix `[0 0 0]` specifies black. The numbers 1-2-3-4 in our image file will then be colored red-green-blue-black.

```
figure;
map = [1 0 0; 0 1 0; 0 0 1; 0 0 0]; colormap(map);
```

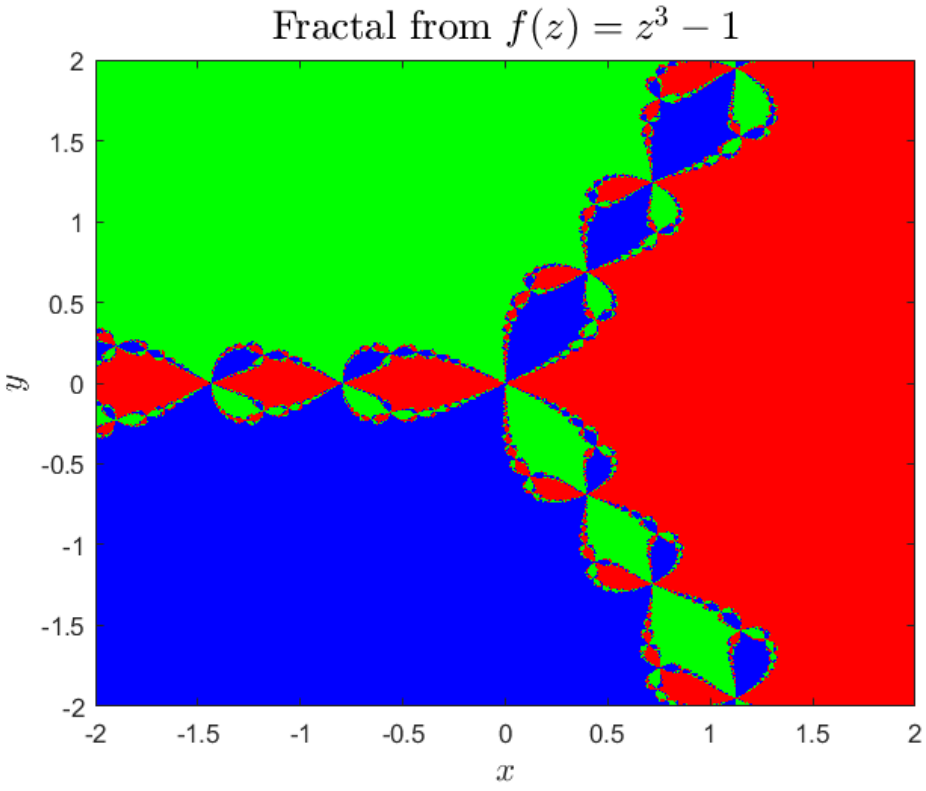
We construct the image file by combining the values of our four `Z` matrices into a single `Z` matrix containing elements 1, 2, 3, or 4.

```
Z=(Z1+2*Z2+3*Z3+4*Z4);
```

The graph is created in our final lines of code. We use the limits of x and y to specify the location of our image in the complex plane. One needs to be aware that the function `image` assumes that the first row of pixels is located at the top of the image. So by default, `image` inverts the y-axis direction by setting the 'YDir' property to 'reverse.' We need to undo this when plotting data from a computation because the usual convention is for the first row of pixels to be at the bottom of the image. We therefore set the 'YDir' property to 'normal.'

```
image([xmin xmax], [ymin ymax], Z); set(gca,'YDir','normal');
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Fractal from $f(z)=z^3-1$', 'Interpreter', 'latex','FontSize', 16)
```

The resulting plot looks like



Problems for Lecture 19

1. Determine the fractal that arises from using Newton's method to compute the four fourth roots of unity.

Solutions to the Problems

Lecture 20 | Root finding in MATLAB

View this lecture on YouTube

MATLAB has two commonly used functions for root finding. The first, `roots.m`, finds the n complex roots of an nth degree polynomial. This function is called with syntax `r = roots(p)`, where `p` is a vector containing $n+1$ polynomial coefficients, starting with the coefficient of x^n. For example, to find the roots of the cubic equation $x^3 - 3x^2 + 4x - 2 = 0$, one types

```
>> p=[1 -3 4 -2];
>> r=roots(p)
r =
    1.0000 + 1.0000i
    1.0000 - 1.0000i
    1.0000 + 0.0000i
>>
```

The function `roots.m` works in a roundabout way by finding the eigenvalues of a matrix using the MATLAB function `eig.m`. You can browse the help file for more details, or type `edit roots.m` to look at the function code directly.

The second MATLAB function, `fzero.m`, finds a real root of a nonlinear function. The function `fzero` is typically called as `r = fzero(f,x0)`, where `fzero` solves $f(x) = 0$, and `x0` is either an initial guess for the root, or a two-dimensional vector whose components bracket the root. For `fzero` to work, the function must change sign at the root. For example, to solve $x = \exp(-a * x)$, where a is a parameter set equal to one-half, one can write the script

```
f = @(x,a) x-exp(-a*x); a=1/2;
x0=1;
r = fzero(@(x) f(x,a), x0);
```

Here, we have defined `f` using an anonymous function. The argument includes x and also parameters. The function `fzero` is passed the defined function as `@(x) f(x,a)`.

Sometimes it may be easier to define f in a subfunction. Then a similar script would look like

```
a=1/2; x0=1;
r = fzero(@(x) f(x,a), x0)
function y = f(x,a)
y=x-exp(-a*x);
end
```

Problems for Lecture 20

1. Consider a planet in an elliptical orbit about the sun. Assume the sun is fixed in space and at the origin of the coordinate system, and the planet is located at position $\mathbf{r}(t) = x(t)\mathbf{i} + y(t)\mathbf{j}$. Let $T = 2\pi/\omega$ be the period of the planet's orbit, let a and b be one-half the major and minor axes of the ellipse, and let $e = \sqrt{1 - b^2/a^2}$ be the eccentricity of the ellipse, with $0 \le e < 1$. Then the coordinates of the planet at time t are given by

$$x(t) = a\,(e - \cos E)\,, \qquad y(t) = b \sin E.$$

The eccentric anomaly, $E = E(t)$, is a solution of Kepler's equation, a transcendental equation given by

$$E = \omega t + e \sin E.$$

By solving Kepler's equation (using `fzero.m`), you will compute and plot several different elliptical orbits of a planet.

The closest approach to the sun occurs when $t = 0$ and $E = 0$. To construct your plots, choose units so that the distance of closest approach is one. At the closest approach, $y = 0$ and $x = a(e - 1)$, so choose the unit of length such that $a(e - 1) = -1$. Solving for a and then b in terms of the dimensionless parameter e, we have

$$a = \frac{1}{1 - e}, \qquad b = \sqrt{\frac{1 + e}{1 - e}}.$$

If we also choose units such that the period of an orbit is one, then $\omega = 2\pi$. On a single plot, show the four orbits corresponding to $e = 0$, $1/4$, $1/2$ and $3/4$.

Solutions to the Problems

Lecture 21 | Project II: Feigenbaum delta (Part A)

View this lecture on YouTube

The bifurcation diagram for the logistic map,

$$x_{i+1} = \mu x_i(1 - x_i),$$

showed a fixed point of the map bifurcating to a period-2 cycle at $\mu = 3$, and to a period-4 cycle near $\mu = 3.45$. A period-8 cycle can be observed soon after, and in fact, period doubling continues indefinitely, eventually accumulating at a value of μ after which the period becomes infinite. This phenomena is called the period-doubling route to chaos, and is observed in many nonlinear systems.

Let μ_n be the value of μ at which the period-2^n cycle bifurcates to a period-2^{n+1} cycle. The Feigenbaum delta is defined as

$$\delta = \lim_{n\to\infty} \frac{\mu_{n-1} - \mu_{n-2}}{\mu_n - \mu_{n-1}},$$

and is a measure of the decreasing width in μ of the period-2^n regions as n increases.

The goal here is to accurately compute the Feigenbaum delta. Let $N = 2^n$, and define the orbit of a period-N cycle to be $x_0, x_1, \ldots, x_{N-1}$. It can be shown that every period-N cycle contains one value of μ with $x = 1/2$ in the orbit. This orbit is superstable: iterations of the logistic map most rapidly converge to the orbit solution. If we define m_n to be the value of μ at which $x_0 = 1/2$, say, is in the orbit of the period-N cycle, then the easiest way to compute the Feigenbaum delta is to replace μ_n by m_n in its definition.

We can determine analytically the first two values of m_n. For the period-1 cycle, we solve $x_0 = \mu x_0(1 - x_0)$; and taking $x_0 = 1/2$ yields $\mu = m_0 = 2$. For the period-2 cycle, we solve

$$x_1 = \mu x_0(1 - x_0), \qquad x_0 = \mu x_1(1 - x_1);$$

and taking $x_0 = 1/2$ reduces to $\mu^3 - 4\mu^2 + 8 = 0$. We can make use of MATLAB's `roots.m`:

```
>> roots([1,-4,0,8])'
ans =
    3.2361      2.0000    -1.2361
>>
```

The root $\mu = 2$ corresponds to the m_0 solution. We could have divided this root out of the cubic equation to obtain $m_1 = 1 + \sqrt{5} \approx 3.2361$. Further values of m_n will be computed numerically.

Problems for Lecture 21

1. Determine the value of m_1 as follows:

a) Show that the period-two fixed-point equations, given by

$$x_1 = \mu x_0(1 - x_0), \qquad x_0 = \mu x_1(1 - x_1),$$

with $x_0 = 1/2$ reduces to $\mu^3 - 4\mu^2 + 8 = 0$.

b) Using long division, determine the quadratic polynomial obtained from

$$\frac{\mu^3 - 4\mu^2 + 8}{\mu - 2}.$$

Show that the positive root of this quadratic is $m_1 = 1 + \sqrt{5}$.

Solutions to the Problems

Lecture 22 | Project II: Feigenbaum delta (Part B)

View this lecture on YouTube

We have defined the Feigenbaum delta as

$$\delta = \lim_{n\to\infty} \frac{m_{n-1} - m_{n-2}}{m_n - m_{n-1}},$$

where m_n is the value of μ at which $x_0 = 1/2$ is in the orbit of the period-N cycle, with $N = 2^n$. Analytically, we found $m_0 = 2$ and $m_1 = 1 + \sqrt{5} \approx 3.2361$.

To determine more values of m_n, we solve a root-finding problem. For each value of n, we determine the value of μ such that the orbit of the logistic map starts with $x_0 = 1/2$ and ends with $x_N = 1/2$. In other words, we solve for $g(\mu) = 0$, where

$$g(\mu) = x_N - 1/2.$$

The roots are $\mu = m_0, m_1, \ldots, m_n$, and the largest root m_n will be the sort-after solution.

We use Newton's method. The derivative of $g(\mu)$ with respect to μ is given by $g'(\mu) = x'_N$. From the logistic map and its derivative, we obtain the coupled equations

$$x_{i+1} = \mu x_i(1 - x_i), \qquad x'_{i+1} = x_i(1 - x_i) + \mu x'_i(1 - 2x_i),$$

with initial values $x_0 = 1/2$ and $x'_0 = 0$. Newton's method then solves for m_n by iterating

$$\mu^{(j+1)} = \mu^{(j)} - \frac{x_N - 1/2}{x'_N},$$

where both x_N and x'_N are obtained by iterating the coupled difference equations N times.

Newton's method requires an initial guess $\mu^{(0)}$ for the root m_n. Using the definition of the Feigenbaum delta, we can approximate m_n by

$$m_n \approx m_{n-1} + \frac{m_{n-1} - m_{n-2}}{\delta_{n-1}},$$

where δ_{n-1} is our current best approximation to the Feigenbaum delta, given by

$$\delta_{n-1} = \frac{m_{n-2} - m_{n-3}}{m_{n-1} - m_{n-2}}.$$

To start, we will take $\delta_1 = 5$, and compute δ_n for $n \geq 2$. The successive values of δ_n should converge to the Feigenbaum delta. In the next lecture, we will outline the construction of a MATLAB code.

Lecture 23 | Project II: Feigenbaum delta (Part C)

View this lecture on YouTube

Your second MATLAB project is to compute the Feigenbaum delta to high accuracy. The first 30 decimal places of the Feigenbaum delta are known to be

$$\delta = 4.669\,201\,609\,102\,990\,671\,853\,203\,821\,578$$

Let's see how close you can get! Your computational engine should contain three nested loops. Here is a reasonable outline:

Loop 1 Start at period-2^n with $n = 2$, and increment n with each iteration
  Compute initial guess for m_n using m_{n-1}, m_{n-2} and δ_{n-1}
  Loop 2 Iterate Newton's method, either a fixed number of times or until convergence
    Initialize logistic map
    Loop 3 Iterate the logistic map 2^n times
      Compute x and x'
    Loop 3 (end)
    One step of Newton's method
  Loop 2 (end)
  Save m_n and compute δ_n
Loop 1 (end)

You will find that your code runs fast, and convergence during Newton's method is quick. Unfortunately, precise calculation of the Feigenbaum delta is limited by round-off error occurring with double precision. Grading will be done on the computed values of δ_n.

For ambitious students:
One method to obtain a higher precision calculation of the Feigenbaum delta using MATLAB is to replace all double precision calculations with variable precision arithmetic. The relevant MATLAB functions that you will need are `vpa` and `digits`, and you can use the MATLAB help pages to develop your code.

Problems for Lecture 23

1. Compute the Feigenbaum delta from the logistic map. The logistic map is given by

$$x_{i+1} = \mu x_i(1 - x_i),$$

and the Feigenbaum delta is defined as

$$\delta = \lim_{n\to\infty} \delta_n, \quad \text{where} \quad \delta_n = \frac{m_{n-1} - m_{n-2}}{m_n - m_{n-1}},$$

and where m_n is the value of μ for which $x_0 = 1/2$ is in the orbit of the period-N cycle with $N = 2^n$. Here is a reasonable outline:

Loop 1 Start at period-2^n with $n = 2$, and increment n with each iteration
  Compute initial guess for m_n using m_{n-1}, m_{n-2} and δ_{n-1}
  Loop 2 Iterate Newton's method, either a fixed number of times or until convergence
    Initialize logistic map
    Loop 3 Iterate the logistic map 2^n times
      Compute x and x'
    Loop 3 (end)
    One step of Newton's method
  Loop 2 (end)
  Save m_n and compute δ_n
Loop 1 (end)

Grading will be done on the computed values of δ_n up to $n = 11$. Set $\delta_1 = 5$.

Solutions to the Problems

Week III

Matrix Algebra

In this week's lectures, we learn about computational matrix algebra. When performing Gaussian elimination, round-off errors can ruin the computation and must be handled using the method of partial pivoting, where row interchanges are performed before each elimination step. The LU decomposition algorithm then includes permutation matrices. We also present the concept of operation counts, and teach the big-Oh notation for predicting the increase in computational time with large problem sizes. We show how to count operations for Gaussian elimination and forward and backward substitution. The power method for computing the largest eigenvalue and associated eigenvector of a matrix is also explained. Finally, we show how to use Gaussian elimination to solve a system of nonlinear differential equations using Newton's method.

Your programming project will be to write a MATLAB *code that applies Newton's method to a system of nonlinear equations.*

Lecture 24 | Gaussian elimination without pivoting

View this lecture on YouTube

Gaussian elimination is the standard algorithm to solve a system of linear equations. However, a straightforward numerical implementation of Gaussian elimination without row or column interchanges (pivoting) can result in large errors because of the computer representation of real numbers. We illustrate here a clear but extreme example of what can go wrong. Be aware, though, that the accumulation of smaller round-off errors in large systems of linear equations can also lead to inaccurate results.

Define $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ to be machine epsilon, that is, the distance between one and the next largest machine number. In computer arithmetic, recall that $2 + \epsilon = 2$, and $4 - \epsilon = 4$.

Now, consider the system of equations given by

$$\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1.$$

Since $\epsilon << 1$, a close approximation to the solution can be found to be $x_2 = 2$ and $x_1 = 3$.

Gaussian elimination on the augmented matrix without row exchanges (called pivoting) follows

$$\begin{pmatrix} \epsilon & 2 & 4 \\ 1 & -1 & 1 \end{pmatrix} \to \begin{pmatrix} \epsilon & 2 & 4 \\ 0 & -\frac{2}{\epsilon} - 1 & -\frac{4}{\epsilon} + 1 \end{pmatrix}.$$

Back substitution, assuming computer arithmetic, then results in

$$x_2 = \frac{-\frac{4}{\epsilon} + 1}{-\frac{2}{\epsilon} - 1} = \frac{4 - \epsilon}{2 + \epsilon} = 2, \qquad x_1 = \frac{4 - 2x_2}{\epsilon} = \frac{4 - 4}{\epsilon} = 0.$$

The result for x_1 is wrong (zero instead of three). We can see where this error comes from if we solve the problem exactly:

$$x_1 = \frac{4 - 2x_2}{\epsilon} = \frac{4 - 2(\frac{4-\epsilon}{2+\epsilon})}{\epsilon} = \frac{6\epsilon}{\epsilon(2+\epsilon)} = \frac{6}{2+\epsilon} \approx 3.$$

In the exact calculation, x_2 is slightly less than two, and a subtraction between two nearly identical numbers leads to the correct answer. But the roundoff errors, here given by $4 - \epsilon = 4$ and $2 + \epsilon = 2$, causes the calculation to go completely awry.

In the next lecture, we will learn how to fix this problem.

Problems for Lecture 24

1. Consider again the system of equations given by

$$\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1.$$

The solution of these equations using Gaussian elimination without pivoting was found to be

$$x_2 = \frac{-\frac{4}{\epsilon} + 1}{-\frac{2}{\epsilon} - 1}, \qquad x_1 = \frac{4 - 2x_2}{\epsilon}.$$

Compute the value of x_2 and x_1 using MATLAB as a calculator. Now, repeat this calculation for the system of equations given by

$$2\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1.$$

Solutions to the Problems

Lecture 25 | Gaussian elimination with partial pivoting

View this lecture on YouTube

Gaussian elimination with partial pivoting is the standard method for solving a linear system of equations on the computer. We consider again the system of equations given by

$$\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1,$$

with a close approximation to the solution given by $x_2 = 2$ and $x_1 = 3$.

The idea of partial pivoting is to perform row interchanges before each elimination step so that the pivot has the largest absolute value among all the rows. The augmented matrix of our system of equations is given by

$$\begin{pmatrix} \epsilon & 2 & 4 \\ 1 & -1 & 1 \end{pmatrix}.$$

To find the best pivot for the first column, we find the row with the largest absolute value in the pivot position. Here we only have two rows, but in general you would compare potential pivots in all n rows. Since the 1 in the second row has larger magnitude than the ϵ in the first row, we interchange rows. We then proceed with the elimination step:

$$\begin{pmatrix} 1 & -1 & 1 \\ \epsilon & 2 & 4 \end{pmatrix} \to \begin{pmatrix} 1 & -1 & 1 \\ 0 & 2+\epsilon & 4-\epsilon \end{pmatrix} \to \begin{pmatrix} 1 & -1 & 1 \\ 0 & 2 & 4 \end{pmatrix},$$

where the last step results from round-off error. Now, back substitution results in

$$x_2 = \frac{4}{2} = 2, \qquad x_1 = 1 + x_2 = 3,$$

a very close approximation to the exact solution.

This method is called *partial pivoting*. *Full pivoting* also implements column exchanges, but this unnecessarily slows down the computation and is usually not implemented.

Problems for Lecture 25

1. Consider again the system of equations given by

$$\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1.$$

The solution of these equations using Gaussian elimination with partial pivoting is found to be

$$x_2 = \frac{4-\epsilon}{2+\epsilon}, \qquad x_1 = 1 + x_2.$$

Compute the value of x_2 and x_1 using MATLAB as a calculator. Now, repeat this calculation for the system of equations given by

$$2\epsilon x_1 + 2x_2 = 4, \qquad x_1 - x_2 = 1.$$

Solutions to the Problems

Lecture 26 | LU decomposition with partial pivoting

View this lecture on YouTube

The LU decomposition can be generalized to include partial pivoting. For example, consider

$$A = \begin{pmatrix} -2 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -8 & 4 \end{pmatrix}.$$

We interchange rows to obtain the largest pivot; that is,

$$A \to \begin{pmatrix} 6 & -6 & 7 \\ -2 & 2 & -1 \\ 3 & -8 & 4 \end{pmatrix} = P_{12}A, \qquad \text{where } P_{12} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is the permutation matrix that interchanges rows one and two. The elimination step is then

$$P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & 0 & 4/3 \\ 0 & -5 & 1/2 \end{pmatrix} = M_2M_1P_{12}A, \qquad \text{where } M_2M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}.$$

The final step requires one more row interchange:

$$M_2M_1P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & -5 & 1/2 \\ 0 & 0 & 4/3 \end{pmatrix} = P_{23}M_2M_1P_{12}A = U.$$

Since the permutation matrices corresponding to a single row exchange are their own inverses, we can write our result as

$$(P_{23}M_2M_1P_{23})P_{23}P_{12}A = U.$$

Multiplication of M_2M_1 on the left by P_{23} interchanges rows two and three, while multiplication on the right by P_{23} interchanges columns two and three. That is,

$$P_{23}\begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 0 & 1 \\ 1/3 & 1 & 0 \end{pmatrix}P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/3 & 0 & 1 \end{pmatrix}.$$

The net result on M_2M_1 is an interchange of the nondiagonal elements $1/3$ and $-1/2$. We can then multiply by the inverse of $(P_{23}M_2M_1P_{23})$ to obtain

$$P_{23}P_{12}A = (P_{23}M_2M_1P_{23})^{-1}U = LU,$$

where

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1/3 & 0 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 6 & -6 & 7 \\ 0 & -5 & 1/2 \\ 0 & 0 & 4/3 \end{pmatrix}$$

Instead of L, MATLAB will write this as

$$A = (P_{12}P_{23}L)U,$$

where the permutation matrices times the lower triangular matrix is called a "psychologically lower triangular matrix," and is given by

$$P_{12}P_{23}L = \begin{pmatrix} -1/3 & 0 & 1 \\ 1 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix}.$$

Problems for Lecture 26

1. Let

$$A = \begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix}.$$

Using Gaussian elimination with partial pivoting, find the (PL)U decomposition of A, where U is an upper triangular matrix and (PL) is a psychologically lower triangular matrix.

Solutions to the Problems

Lecture 27 | Operation counts

View this lecture on YouTube

To estimate the computational time of an algorithm, one counts the number of operations required (multiplications, additions, etc.). Usually, one is interested in how an algorithm scales with the size of the problem. One can easily time a small problem, and then estimate the computational time required for a much larger problem.

For example, suppose one is multiplying two full $n \times n$ matrices. The calculation of each element requires n multiplications and $n-1$ additions. There are n^2 elements to compute so that the total operation counts are n^3 multiplications and $n^2(n-1)$ additions.

For large n, we might want to know how much longer the computation will take if n is doubled. What matters most is the fastest-growing, leading-order term in the operation count. In this matrix multiplication example, the leading-order term grows like n^3, and we say that the algorithm scales like $O(n^3)$, which is read as "big Oh of n cubed." When using the big-Oh notation, we drop both lower-order terms and constant multipliers.

The big-Oh notation tells us how the computational time of an algorithm scales. For example, suppose that the multiplication of two large $n \times n$ matrices took a computational time of T_n seconds. With the known operation count going like $O(n^3)$, we can write $T_n = kn^3$ for some constant k. To determine how much longer the multiplication of two $2n \times 2n$ matrices will take, we write

$$T_{2n} = k(2n)^3 = 8kn^3 = 8T_n,$$

so that doubling the size of the matrix is expected to increase the computational time by a factor of $2^3 = 8$.

Running MATLAB on my computer, the multiplication of two 4096×4096 matrices took about 1.1 sec. The multiplication of two 8192×8192 matrices took about 7.9 sec, which is a bit more than 7 times longer. Timing of code in MATLAB can be found using the built-in stopwatch functions tic and toc.

Problems for Lecture 27

1. A genetic model of recombination is solved using a computational algorithm that scales like $O(3^L)$, where L is the number of loci modeled. If it takes 10 sec to compute recombination when $L = 15$, estimate how long it takes to compute recombination when $L = 16$.

Solutions to the Problems

Lecture 28 | Operation counts for Gaussian elimination

View this lecture on YouTube

When counting the number of operations required for an algorithm, three well-known summation formulas will come in handy. They are

$$\sum_{k=1}^{n} 1 = n, \qquad \sum_{k=1}^{n} k = \frac{1}{2}n(n+1), \qquad \sum_{k=1}^{n} k^2 = \frac{1}{6}n(2n+1)(n+1).$$

To determine the operation counts for Gaussian elimination used to solve a system of n equations and n unknowns, consider an elimination step with the pivot on the diagonal in the ith row and column. There are both $n-i$ rows below the pivot and $n-i$ columns to the right of the pivot. To perform elimination on one row, each matrix element to the right of the pivot must be multiplied by a factor and added to the row underneath. This must be done for all the rows. There are therefore $(n-i)(n-i)$ multiplication-additions to be done for a pivot . I will count a multiplication-addition as one operation.

To find the total operation count, we need to perform elimination using $n-1$ pivots, so that

$$\begin{aligned} \text{op. counts} &= \sum_{i=1}^{n-1} (n-i)^2 = (n-1)^2 + (n-2)^2 + \ldots (1)^2 \\ &= \sum_{i=1}^{n-1} i^2 = \frac{1}{6}(n-1)(2n-1)n. \end{aligned}$$

The leading-order term is $n^3/3$, and we say that Gaussian elimination scales like $O(n^3)$. Since LU decomposition with partial pivoting is essentially Gaussian elimination, it will have the same scaling. Double the number of equations and unknowns, and LU decomposition will take approximately eight times longer.

Problems for Lecture 28

1. Derive the following summation identities:

a)

$$\sum_{k=1}^{n} 1 = n;$$

b)

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1);$$

c)

$$\sum_{k=1}^{n} k^2 = \frac{1}{6}n(2n+1)(n+1).$$

Solutions to the Problems

Lecture 29 | Operation counts for forward and backward substitution

View this lecture on YouTube

Once the LU decomposition of a matrix A is known, the solution of Ax = b can proceed by forward and backward substitution. Here, we compute the operation counts for backward substitution. Forward substitution will be similar. The matrix equation to be solved is of the form

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1(n-1)} & a_{1n} \\ 0 & a_{22} & \cdots & a_{2(n-1)} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{(n-1)(n-1)} & a_{(n-1)n} \\ 0 & 0 & \cdots & 0 & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

The solution for x_i is found after solving for x_j with $j > i$. The explicit solution is given by

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right).$$

The solution for x_i requires $n - i + 1$ multiplications and $n - i$ additions, and we need to solve for n values of x_i. Counting just multiplications, we have

$$\begin{aligned} \text{op. counts} &= \sum_{i=1}^{n} n - i + 1 = n + (n-1) + \cdots + 1 \\ &= \sum_{i=1}^{n} i = \frac{1}{2} n(n+1). \end{aligned}$$

The leading-order term is $n^2/2$, and forward substitution has a similar scaling. The operation counts for a forward and backward substitution, therefore, scales like $O(n^2)$.

For n large, n^2 grows much slower than n^3, and a forward and backward substitution should be substantially faster than Gaussian elimination. If the matrix is fixed and the right-hand side keeps changing — for example, in a time-dependent computation where the matrix is independent of time but the right-hand side changes over time — a fast computation would first find the LU decomposition of the matrix, and then solve the recurring systems of equations by forward and backward substitution.

Problems for Lecture 29

1. Solve the following lower triangular system for x_i in terms of x_j, $j < i$:

$$\begin{pmatrix} a_{11} & 0 & \cdots & 0 & 0 \\ a_{21} & a_{22} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & \cdots & a_{(n-1)(n-1)} & 0 \\ a_{n1} & a_{n2} & \cdots & a_{n(n-1)} & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

Count the total number of multiplication-additions required for a complete solution.

Solutions to the Problems

Lecture 30 | Eigenvalue power method

View this lecture on YouTube

The power method is a fast iterative algorithm to find the largest magnitude eigenvalue of a matrix and its associated eigenvector. Suppose that A is an n-by-n matrix with n distinct real eigenvalues, ordered such that $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, and with corresponding linearly independent eigenvectors $\mathrm{e}_1, \mathrm{e}_2, \ldots \mathrm{e}_n$. Then any vector x_0 can be written as linear combination of the eigenvectors as

$$\mathrm{x}_0 = c_1\mathrm{e}_1 + c_2\mathrm{e}_2 + \ldots c_n\mathrm{e}_n = \sum_{i=1}^{n} c_i\mathrm{e}_i.$$

With $\mathrm{x}_1 = \mathrm{A}\mathrm{x}_0$, we have

$$\mathrm{x}_1 = \mathrm{A}\sum_{i=1}^{n} c_i\mathrm{e}_i = \sum_{i=1}^{n} c_i\mathrm{A}\mathrm{e}_i = \sum_{i=1}^{n} c_i\lambda_i\mathrm{e}_i.$$

Repeated multiplication by A, with $\mathrm{x}_p = \mathrm{A}^p\mathrm{x}_0$, yields

$$\mathrm{x}_p = \sum_{i=1}^{n} c_i\lambda_i^p\mathrm{e}_i = \lambda_1^p\left(c_1\mathrm{e}_1 + \sum_{i=2}^{n} c_i(\lambda_i/\lambda_1)^p\mathrm{e}_i\right).$$

Since for $i \geq 2$,

$$\lim_{p\to\infty}(\lambda_i/\lambda_1)^p = 0,$$

a good approximation for large enough values of p is

$$\mathrm{x}_p \approx c_1\lambda_1^p\mathrm{e}_1.$$

To find the dominant eigenvalue, we write

$$\frac{\mathrm{x}_p^{\mathrm{T}}\mathrm{x}_{p+1}}{\mathrm{x}_p^{\mathrm{T}}\mathrm{x}_p} = \frac{\left(c_1\lambda_1^p\mathrm{e}_1^{\mathrm{T}}\right)\left(c_1\lambda_1^{p+1}\mathrm{e}_1\right)}{\left(c_1\lambda_1^p\mathrm{e}_1^{\mathrm{T}}\right)\left(c_1\lambda_1^p\mathrm{e}_1\right)} = \frac{c_1^2\lambda_1^{2p+1}\mathrm{e}_1^{\mathrm{T}}\mathrm{e}_1}{c_1^2\lambda_1^{2p}\mathrm{e}_1^{\mathrm{T}}\mathrm{e}_1} = \lambda_1.$$

The corresponding eigenvector is found from x_p, which can be normalized as you like.

In practice, after many iterations underflow (when $\lambda_1 < 1$) or overflow (when $\lambda_1 > 1$) can occur, and it is best to normalize the vectors x_p after each iteration. A simple method of normalization is

$$\mathrm{x}_p \to \frac{\mathrm{x}_p}{(\mathrm{x}_p^{\mathrm{T}}\mathrm{x}_p)^{1/2}}.$$

Problems for Lecture 30

1. The two largest (in absolute value) eigenvalues of an n-by-n matrix with real eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 1/2$. Give a rough estimate of how many iterations of the power method is required for convergence to an error of less than 10^{-8}.

Solutions to the Problems

Lecture 31 | Eigenvalue power method (example)

View this lecture on YouTube

Use the power method to determine the dominant eigenvalue and corresponding eigenvector of the matrix

$$A = \begin{pmatrix} 6 & 5 \\ 4 & 5 \end{pmatrix}.$$

For reference, the eigenvalues and eigenvectors of this matrix are given by

$$\lambda_1 = 10, \quad v_1 = \begin{pmatrix} 5/4 \\ 1 \end{pmatrix}; \qquad \lambda_2 = 1, \quad v_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

We apply here the power method without normalizing the vectors after each iteration. A computer solution should normalize vectors to avoid overflow or underflow. The assumed initial vector and first iteration is given by

$$x_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad x_1 = Ax_0 = \begin{pmatrix} 6 \\ 4 \end{pmatrix}.$$

Continuing,

$$x_2 = \begin{pmatrix} 6 & 5 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 56 \\ 44 \end{pmatrix}; \qquad x_3 = \begin{pmatrix} 6 & 5 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 56 \\ 44 \end{pmatrix} = \begin{pmatrix} 556 \\ 444 \end{pmatrix}.$$

Two more iterations give

$$x_4 = \begin{pmatrix} 6 & 5 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 556 \\ 444 \end{pmatrix} = \begin{pmatrix} 5{,}556 \\ 4{,}444 \end{pmatrix}; \qquad x_5 = \begin{pmatrix} 6 & 5 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 5{,}556 \\ 4{,}444 \end{pmatrix} = \begin{pmatrix} 55{,}556 \\ 44{,}444 \end{pmatrix}.$$

The dominant eigenvalue is approximated from

$$\lambda_1 \approx \frac{x_4^T x_5}{x_4^T x_4} = \frac{506{,}178{,}271}{50{,}618{,}272} = 9.99991 \approx 10;$$

and the corresponding eigenvector is approximated by x_5. Dividing by the second component,

$$v_1 = \begin{pmatrix} 55{,}556/44{,}444 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 1.25002 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 5/4 \\ 1 \end{pmatrix}.$$

Problems for Lecture 31

1. Use the power method (without normalizing the vectors after each iteration) to determine the dominant eigenvalue and corresponding eigenvector of the matrix

$$A = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix}.$$

Solutions to the Problems

Lecture 32 | Matrix algebra in MATLAB

View this lecture on YouTube

In MATLAB, the solution to the matrix equation $Ax = b$ is found by typing `x=A\b`. The backslash operator can solve n equations and n unknowns, or a least-squares problem that has more equations than unknowns. For example, to solve the following system of three linear equations and three unknowns:

$$\begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -7 \\ -6 \end{pmatrix},$$

we type

```
>> A=[-3 2 -1; 6 -6 7; 3 -4 4]; b=[-1;-7;-6];
>> x=A\b
x =
      2
      2
     -1
>>
```

To solve the toy least-squares problem of three equations and two unknowns:

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix},$$

we type

```
>> A=[1 1; 1 2; 1 3]; b=[1; 3; 2];
>> x=A\b
x =
      1.0000
      0.5000
>>
```

For a problem with one matrix and many different right-hand-sides, one first finds $A = LU$ by a function call, and then solves for x using `x=U\(L\b)`. For example, to solve again the system of three linear equations and three unknowns, we could type

```
>> A=[-3 2 -1; 6 -6 7; 3 -4 4]; b=[-1; -7; -6];
>> [L U] = lu(A);
>> x = U\(L\b)
x =
```

```
      2
      2
     -1
>>
```

MATLAB can also solve the eigenvalue problem $Ax = \lambda x$. The function `eig.m` can be used to find all the eigenvalues and eigenvectors of A. For example, to find all the eigenvalues of a two-by-two matrix A, we type

```
>> A=[0 1; 1 0];
>> lambda = eig(A)
lambda =
     -1
      1
>>
```

To also find the eigenvectors, we type

```
>> A=[0 1; 1 0];
>> [V, D] = eig(A)
V =
    -0.7071    0.7071
     0.7071    0.7071
D =
     -1     0
      0     1
>>
```

The latter function call finds two matrices V and D that satisfy the matrix equation $AV = VD$.

Problems for Lecture 32

1. Let

$$A = \begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix}.$$

Use MATLAB to find the LU decomposition of A, where U is an upper triangular matrix and L is a psychologically lower triangular matrix.

2. Use MATLAB to find the eigenvalues and eigenvectors of the matrix

$$A = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix}.$$

Normalize the eigenvectors so that their second component is one.

Solutions to the Problems

Lecture 33 | Systems of nonlinear equations

View this lecture on YouTube

A system of nonlinear equations can be solved using Newton's method and matrix algebra. We derive the method for a system of two equations and two unknowns. Suppose that we want to solve

$$f(x,y) = 0, \qquad g(x,y) = 0,$$

for the unknowns x and y. To solve, we construct two parallel sequences $x_0, x_1, x_2, \ldots$ and $y_0, y_1, y_2, \ldots$ that converge to the root. To construct these sequences, we Taylor series expand $f(x_{n+1}, y_{n+1})$ and $g(x_{n+1}, y_{n+1})$ about the point (x_n, y_n). Using the partial derivatives $f_x = \partial f/\partial x$, $f_y = \partial f/\partial y$ and those for g, the two-dimensional Taylor series expansions, displaying up to the linear terms, are given by

$$\begin{aligned} f(x_{n+1}, y_{n+1}) &= f(x_n, y_n) + (x_{n+1} - x_n) f_x(x_n, y_n) + (y_{n+1} - y_n) f_y(x_n, y_n) + \ldots, \\ g(x_{n+1}, y_{n+1}) &= g(x_n, y_n) + (x_{n+1} - x_n) g_x(x_n, y_n) + (y_{n+1} - y_n) g_y(x_n, y_n) + \ldots. \end{aligned}$$

To obtain Newton's method, we set $f(x_{n+1}, y_{n+1}) = g(x_{n+1}, y_{n+1}) = 0$, and drop higher-order terms above linear. It is best to define the unknowns as

$$\Delta x_n = x_{n+1} - x_n, \qquad \Delta y_n = y_{n+1} - y_n,$$

so that the iteration equations will be given by

$$x_{n+1} = x_n + \Delta x_n, \qquad y_{n+1} = y_n + \Delta y_n.$$

The linear equations to be solved for Δx_n and Δy_n, in matrix form, are

$$\begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix} = - \begin{pmatrix} f \\ g \end{pmatrix},$$

where f, g, f_x, f_y, g_x and g_y are all evaluated at the point (x_n, y_n). The two-equation case is easily generalized to n equations. The matrix of partial derivatives is called the Jacobian matrix.

Each step of the iteration requires solving the n-by-n system of linear equations using Gaussian elimination.

Problems for Lecture 33

1. The algorithm for solving the system of two equations and two unknowns,

$$f(x,y) = 0, \qquad g(x,y) = 0,$$

is given by the following two-step process.

1. Solve the linear system for Δx_n and Δy_n given by

$$\begin{pmatrix} f_x & f_y \\ g_x & g_y \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix} = \begin{pmatrix} -f \\ -g \end{pmatrix}.$$

2. Advance the iterative solution, using

$$x_{n+1} = x_n + \Delta x_n, \qquad y_{n+1} = y_n + \Delta y_n.$$

Write down the corresponding algorithm for three equations and three unknowns.

Solutions to the Problems

Lecture 34 | Systems of nonlinear equations (example)

View this lecture on YouTube

The Lorenz equations — a system of nonlinear odes that pioneered the study of chaos — are given by

$$\dot{x} = \sigma(y - x), \qquad \dot{y} = x(r - z) - y, \qquad \dot{z} = xy - \beta z,$$

where σ, β and r are constant parameters. The fixed-point solutions of these equations satisfy $\dot{x} = \dot{y} = \dot{z} = 0$. Although the fixed-point solutions can be found analytically, here we will illustrate Newton's method. We solve

$$\sigma(y - x) = 0, \qquad rx - y - xz = 0, \qquad xy - \beta z = 0.$$

This is a root-finding problem, where we are solving $f(x, y, z) = g(x, y, z) = h(x, y, z) = 0$, and

$$f(x, y, z) = \sigma(y - x), \qquad g(x, y, z) = rx - y - xz, \qquad h(x, y, z) = xy - \beta z.$$

The Jacobian matrix is found from the partial derivatives, computed to be

$$\begin{aligned} f_x &= -\sigma, & f_y &= \sigma, & f_z &= 0, \\ g_x &= r - z, & g_y &= -1, & g_z &= -x, \\ h_x &= y, & h_y &= x, & h_z &= -\beta. \end{aligned}$$

Using the Jacobian matrix, the iteration equation for Newton's method is given by

$$\begin{pmatrix} -\sigma & \sigma & 0 \\ r - z_n & -1 & -x_n \\ y_n & x_n & -\beta \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{pmatrix} = - \begin{pmatrix} \sigma(y_n - x_n) \\ rx_n - y_n - x_n z_n \\ x_n y_n - \beta z_n \end{pmatrix},$$

with

$$x_{n+1} = x_n + \Delta x_n, \qquad y_{n+1} = y_n + \Delta y_n, \qquad z_{n+1} = z_n + \Delta z_n.$$

Problems for Lecture 34

1. The fixed-point solutions of the Lorenz equations satisfy

$$\sigma(y - x) = 0, \qquad rx - y - xz = 0, \qquad xy - \beta z = 0.$$

Find analytically three fixed-point solutions for (x, y, z) as a function of the parameters σ, β and r. What are the numerical values for the fixed points when $r = 28$, $\sigma = 10$ and $\beta = 8/3$?

2. Complete a MATLAB code that uses Newton's method to determine the fixed-point solutions of the Lorenz equations. Solve using the parameters $r = 28$, $\sigma = 10$ and $\beta = 8/3$. Use as your three initial guesses $x = y = z = 1$, $x = y = z = 10$ and $x = y = z = -10$.

Solutions to the Problems

Lecture 35 | Project III: Fractals from the Lorenz equations

View this lecture on YouTube

The fixed points of the Lorenz equations satisfy

$$\sigma(y-x)=0, \qquad rx-y-xz=0, \qquad xy-\beta z=0.$$

Analytical solution of the fixed-point equations yields the three roots,

$$(x,y,z)= \quad (0,0,0), \quad (\pm\sqrt{\beta(r-1)},\pm\sqrt{\beta(r-1)},r-1).$$

Implementation of Newton's method to determine the solution of the fixed-point equations requires iteration of

$$\begin{pmatrix} -\sigma & \sigma & 0 \\ r-z_n & -1 & -x_n \\ y_n & x_n & -\beta \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{pmatrix} = - \begin{pmatrix} \sigma(y_n-x_n) \\ rx_n-y_n-x_nz_n \\ x_ny_n-\beta z_n \end{pmatrix},$$

with

$$x_{n+1}=x_n+\Delta x_n, \qquad y_{n+1}=y_n+\Delta y_n, \qquad z_{n+1}=z_n+\Delta z_n.$$

We may suppose that an initial value given by (x_0,y_0,z_0) will converge to one of the three roots of the fixed-point equations. By mapping a grid of initial values to the converged root, an image of a fractal may result. Your third project is to compute an instance of one of these fractals.

Problems for Lecture 35

1. Determine a fractal that arises from using Newton's method to compute the fixed-point solutions of the Lorenz equations. Use the parameter values $r = 28$, $\sigma = 10$ and $\beta = 8/3$. Initial values (x_0, z_0) are taken on a grid in the x-z plane with always $y_0 = 3\sqrt{2}$. For assessment purposes, the computational grid and the graphics code will be given in the Learner Template. To pass the assessment, every pixel in your figure needs to be colored correctly.

(Hint: Some grid points may require as many as 33 Newton iterations to converge while others may require as few as three. Unfortunately, if you uniformly use 33 Newton iterations at every grid point, the MATLAB Grader may time out. You can accelerate your code by using a `while` loop instead of a `for` loop.)

Solutions to the Problems

Week IV

Quadrature and Interpolation

In this week's lectures, we learn about numerical integration—also called quadrature—and function interpolation. We begin by learning the basics of quadrature, which include the elementary formulas for the trapezoidal rule and Simpson's rule, and how these formulas can be used to develop composite integration rules. We then discuss the more sophisticated method of Gaussian quadrature, and learn how to construct an adaptive integration routine in which the software itself determines the appropriate integration step size. Finally, we learn how to use the MATLAB *function* `integral.m`.

In the second-half of this week we learn about interpolation. Given known points in the plane arising from a smooth function, a good interpolation routine will be able to estimate the values between these points. Linear interpolation is widely used, particularly when plotting data consisting of many points. The more sophisticated method of cubic splines may be a better choice, particularly if the points are more sparse.

Your programming project will be to write a MATLAB *code to compute the zeros of the Bessel functions. This requires combining both quadrature and root-finding routines.*

Lecture 36 | Midpoint rule

View this lecture on YouTube

Elementary integration formulas are derived by integrating $f(x)$ from 0 to h, and these formulas serve as building blocks for integrating over any interval. Here, we derive the midpoint rule.

We define the definite integral I_h to be

$$I_h = \int_0^h f(x)dx.$$

To perform this integral, we Taylor series expand $f(x)$ about the value $x = h/2$:

$$f(x) = f(h/2) + (x - h/2)f'(h/2) + \frac{(x-h/2)^2}{2}f''(h/2) + \frac{(x-h/2)^3}{6}f'''(h/2) + \frac{(x-h/2)^4}{24}f''''(h/2) + \dots.$$

Integrating both sides from zero to h, and using the definition of I_h, we have

$$I_h = \int_0^h \left(f(h/2) + (x - h/2)f'(h/2) + \frac{(x-h/2)^2}{2}f''(h/2) + \frac{(x-h/2)^3}{6}f'''(h/2) + \frac{(x-h/2)^4}{24}f''''(h/2) + \dots \right) dx.$$

To perform the integration, we let $y = x - h/2$ so that $dy = dx$, and simplify according to whether each term in the integrand is even or odd. We have

$$\begin{aligned} I_h &= \int_{-h/2}^{h/2} \left(f(h/2) + yf'(h/2) + \frac{y^2}{2}f''(h/2) + \frac{y^3}{6}f'''(h/2) + \frac{y^4}{24}f''''(h/2) + \dots \right) dy \\ &= 2\int_0^{h/2} \left(f(h/2) + \frac{y^2}{2}f''(h/2) + \frac{y^4}{24}f''''(h/2) + \dots \right) dy. \end{aligned}$$

Using

$$\int_0^{h/2} y^2 dy = h^3/24, \qquad \int_0^{h/2} y^4 dy = h^5/160,$$

we obtain the midpoint rule, given by

$$\begin{aligned} I_h &= hf(h/2) + \frac{h^3}{24}f''(h/2) + \frac{h^5}{1920}f''''(h/2) + \dots \\ &= hf(h/2) + \mathrm{O}(h^3). \end{aligned}$$

Problems for Lecture 36

1. The area of a rectangle with base h and height $f(h/2)$ is given by $hf(h/2)$. Draw a graph illustrating the midpoint rule.

2. Let $f(x) = a + bx + cx^2$, where a, b and c are constants. Prove by explicit calculation that

$$\int_0^h f(x)\,dx = hf(h/2) + \frac{h^3}{24}f''(h/2).$$

Solutions to the Problems

Lecture 37 | Trapezoidal rule

View this lecture on YouTube

The trapezoidal rule is one of the most useful integration formulas. From the Taylor series expansion of $f(x)$ about $x = h/2$, we have

$$f(0) = f(h/2) - \frac{h}{2}f'(h/2) + \frac{h^2}{8}f''(h/2) - \frac{h^3}{48}f'''(h/2) + \frac{h^4}{384}f''''(h/2) + \dots,$$
$$f(h) = f(h/2) + \frac{h}{2}f'(h/2) + \frac{h^2}{8}f''(h/2) + \frac{h^3}{48}f'''(h/2) + \frac{h^4}{384}f''''(h/2) + \dots.$$

Adding and multiplying by $h/2$ we obtain

$$\frac{h}{2}\big(f(0) + f(h)\big) = hf(h/2) + \frac{h^3}{8}f''(h/2) + \frac{h^5}{384}f''''(h/2) + \dots.$$

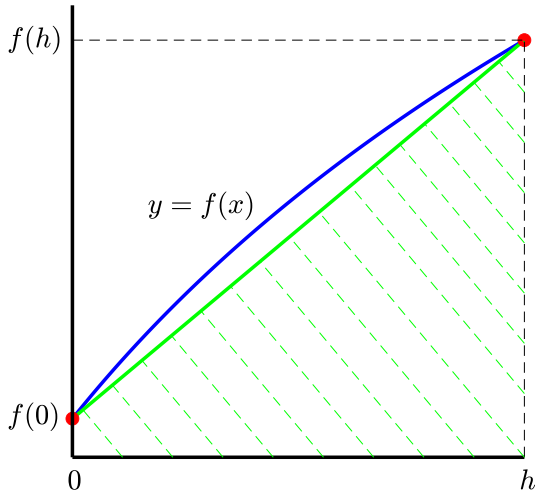
We now substitute for $hf(h/2)$ using the midpoint rule formula:

$$\frac{h}{2}\big(f(0)+f(h)\big) = \left(I_h - \frac{h^3}{24}f''(h/2) - \frac{h^5}{1920}f''''(h/2)\right) + \frac{h^3}{8}f''(h/2) + \frac{h^5}{384}f''''(h/2) + \dots,$$

and solving for I_h, we obtain the trapezoidal rule,

$$\begin{aligned} I_h &= \frac{h}{2}\big(f(0) + f(h)\big) - \frac{h^3}{12}f''(h/2) - \frac{h^5}{480}f''''(h/2) + \dots \\ &= \frac{h}{2}\big(f(0) + f(h)\big) + \mathrm{O}(h^3). \end{aligned}$$

A graphical representation of the trapezoidal rule is shown on the right, where the integral is approximated by the area of the drawn trapezoid.



Problems for Lecture 37

1. Derive the trapezoidal rule by approximating $f(x)$ by the straight line connecting the points $(0, f(0))$ and $(h, f(h))$:

$$f(x) \approx f(0) + \frac{f(h) - f(0)}{h} x.$$

Solutions to the Problems

Lecture 38 | Simpson's rule

View this lecture on YouTube

To obtain Simpson's rule, we combine the midpoint and trapezoidal rule to eliminate the error term proportional to h^3. The midpoint and trapezoidal rules are given by

$$I_h = hf(h/2) + \frac{h^3}{24}f''(h/2) + \frac{h^5}{1920}f''''(h/2) + \dots$$
$$I_h = \frac{h}{2}\big(f(0) + f(h)\big) - \frac{h^3}{12}f''(h/2) - \frac{h^5}{480}f''''(h/2) + \dots.$$

Multiplying the midpoint rule by two and adding it to the trapezoidal rule, we obtain

$$3I_h = h\left(2f(h/2) + \frac{1}{2}\big(f(0) + f(h)\big)\right) + h^5\left(\frac{2}{1920} - \frac{1}{480}\right)f''''(h/2) + \dots,$$

or

$$\begin{aligned} I_h &= \frac{h}{6}\big(f(0) + 4f(h/2) + f(h)\big) - \frac{h^5}{2880}f''''(h/2) + \dots \\ &= \frac{h}{6}\big(f(0) + 4f(h/2) + f(h)\big) + \mathrm{O}(h^5). \end{aligned}$$

Usually, Simpson's rule is written by considering the three consecutive points 0, h and $2h$. Transforming $h \to 2h$, we obtain the more standard result

$$\begin{aligned} I_{2h} &= \frac{h}{3}\big(f(0) + 4f(h) + f(2h)\big) - \frac{h^5}{90}f''''(h) + \dots \\ &= \frac{h}{3}\big(f(0) + 4f(h) + f(2h)\big) + \mathrm{O}(h^5). \end{aligned}$$

Problems for Lecture 38

1. Derive Simpson's rule by approximating $f(x)$ by a quadratic polynomial connecting the points $(0, f(0))$, $(h, f(h))$ and $(2h, f(2h))$.

a) Let $g(x) = a + bx + cx^2$. Determine the values of a, b and c such that $g(x)$ passes through the points $(0, f(0))$, $(h, f(h))$ and $(2h, f(2h))$.

b) Use $f(x) \approx g(x)$ to derive Simpson's rule.

Solutions to the Problems

Lecture 39 | Composite quadrature rules

View this lecture on YouTube

We apply our elementary integration formulas to compute

$$I = \int_a^b f(x)\,dx.$$

Suppose that the function $f(x)$ is known at the points $a = x_0, x_1, \ldots, x_n = b$. Define

$$f_i = f(x_i), \quad h_i = x_{i+1} - x_i.$$

Then the integral I may be decomposed as

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx = \sum_{i=0}^{n-1} \int_0^{h_i} f(x_i + s)ds,$$

where the last equality arises from the change-of-variables $s = x - x_i$. Applying the trapezoidal rule to the integral, we have

$$\int_a^b f(x)dx = \frac{1}{2} \sum_{i=0}^{n-1} h_i \left(f_i + f_{i+1}\right),$$

which is often useful for integrating experimental data.

If the points are evenly spaced, we have $h_i = h = (b-a)/n$, and the composite trapezoidal rule becomes

$$\int_a^b f(x)dx = \frac{h}{2} \left(f_0 + 2f_1 + \cdots + 2f_{n-1} + f_n\right).$$

The first and last terms have a multiple of one, all other terms have a multiple of two, and the entire sum is multiplied by $h/2$.

Similarly, the composite Simpson's rule for evenly space points is found to be

$$\begin{aligned}\int_a^b f(x)dx &= \frac{h}{3}\left(f_0 + 4f_1 + f_2\right) + \frac{h}{3}\left(f_2 + 4f_3 + f_4\right) + \cdots + \frac{h}{3}\left(f_{n-2} + 4f_{n-1} + f_n\right) \\ &= \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 4f_{n-1} + f_n\right).\end{aligned}$$

Note that n must be even for this scheme to work. The first and last terms have a multiple of one, the odd indexed terms have a multiple of 4, the even indexed terms have a multiple of 2, and the entire sum is multiplied by $h/3$.

For composite rules, the error for each elementary interval sums so that the global error increases by a factor of $1/h$. The trapezoidal rule and Simpson's rule have global error h^2 and h^4, respectively.

Problems for Lecture 39

1. Simpson's 3/8 rule has elementary formula given by

$$\int_0^{3h} f(x)\,dx = \frac{3h}{8}\left(f(0) + 3f(h) + 3f(2h) + f(3h)\right).$$

Suppose that $f(x)$ is known at the equally spaced points $a = x_0, x_1, \ldots, x_n = b$, and n is a multiple of three. Let $f_i = f(x_i)$ and $h = x_{i+1} - x_i$. Find the formula for the composite Simpson's 3/8 rule.

Solutions to the Problems

Lecture 40 | Gaussian quadrature

View this lecture on YouTube

A general Gaussian quadrature rule is of the form

$$\int_a^b W(x)f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i),$$

where $W(x)$ is called the weight function, w_i are called the weights, and x_i are called the nodes. The weights and nodes are chosen so that the integral is exact for polynomial functions $f(x)$ of degree less than or equal to $2n-1$. Variations include Chebyshev-Gauss, Laguerre-Gauss, and Hermite-Gauss quadrature. The definite integrals associated with these Gaussian quadratures are given, respectively, by

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)\,dx, \qquad \int_0^{\infty} e^{-x} f(x)\,dx, \qquad \int_{-\infty}^{\infty} e^{-x^2} f(x)\,dx.$$

The most straightforward example is the Legendre-Gauss quadrature rule, with the two-point formula given by

$$\int_{-1}^{1} f(x)\,dx \approx w_1 f(x_1) + w_2 f(x_2).$$

To determine the weights and the nodes, we require the rule to be exact for the polynomial basis functions $f(x) = 1$, x, x^2 and x^3. Substituting in these functions yields the four equations with four unknowns given by

$$2 = w_1 + w_2, \quad 0 = w_1 x_1 + w_2 x_2, \quad 2/3 = w_1 x_1^2 + w_2 x_2^2, \quad 0 = w_1 x_1^3 + w_2 x_2^3.$$

If we argue from symmetry that $x_1 = -x_2$ and $w_1 = w_2$, then a solution is readily found to be $w_1 = w_2 = 1$, and $x_1 = -1/\sqrt{3}$ and $x_2 = 1/\sqrt{3}$.

Problems for Lecture 40

1. Determine the weights and nodes of the three-point Legendre-Gauss quadrature rule. You may assume from symmetry that $x_1 = -x_3$, $w_1 = w_3$, and $x_2 = 0$.

Solutions to the Problems

Lecture 41 | Adaptive quadrature

View this lecture on YouTube

Adaptive quadrature lets the algorithm itself determine the integration step size required to meet a user-specified precision. Even more importantly, the step size need not be constant over the entire region of integration. Using the trapezoidal rule, we sketch how adaptive quadrature is implemented.

We begin integration of $I = \int_a^b f(x)dx$ at what is called Level 1. Let $h = b - a$. Then integration from a to b using the trapezoidal rule is given by

$$I = \frac{h}{2}\left(f(a) + f(b)\right) - \frac{h^3}{12}f''(\xi),$$

where the Taylor remainder theorem states that ξ is a number between a and b. We now add a point c midway between a and b and apply the composite trapezoidal rule to obtain

$$I = \frac{h}{4}\left(f(a) + 2f(c) + f(b)\right) - \frac{(h/2)^3}{12}f''(\xi_l) - \frac{(h/2)^3}{12}f''(\xi_r),$$

where ξ_l is a number between a and c and ξ_r is a number between c and b.

We can define our two approximations to the integral by

$$S_1 = \frac{h}{2}\big(f(a) + f(b)\big), \qquad S_2 = \frac{h}{4}\big(f(a) + 2f(c) + f(b)\big),$$

and our two associated errors by

$$E_1 = -\frac{h^3}{12}f''(\xi), \qquad E_2 = -\frac{h^3}{2^3 \cdot 12}\big(f''(\xi_l) + f''(\xi_r)\big).$$

If we make the approximation that $f''(\xi) \approx f''(\xi_l) \approx f''(\xi_r)$, then $E_1 \approx 4E_2$. Then since $S_1 + E_1 = S_2 + E_2$, we can eliminate E_1 to find an estimate for E_2:

$$|E_2| \approx |S_2 - S_1|/3.$$

If $|E_2|$ is less than some specified tolerance, then we accept S_2 as I. Otherwise, we proceed to Level 2.

The computation at Level 2 starts with the two integration intervals a to c and c to b, and repeats the above procedure independently on both intervals. Integration can be stopped on either interval provided the estimated error on that interval is less than one-half the specified tolerance (since the sum of the errors on both intervals must be less than the tolerance). Otherwise, either interval can proceed to Level 3, and so on, until the global error estimate is less than the tolerance.

Problems for Lecture 41

1. Consider $I = \int_0^h f(x)\,dx$ with $f(x) = x^3$. Using the trapezoidal rule, compute S_1, S_2, E_1 and E_2 and show that $E_1 = 4E_2$.

Solutions to the Problems

Lecture 42 | Quadrature in MATLAB

View this lecture on YouTube

The MATLAB function `integral.m` performs adaptive quadrature by combining a seven-point Legendre-Gauss quadrature rule with a fifteen-point Kronrod rule. We will skip the details here.

If S is the computed value of the integral and I is the (unknown) exact value, then the absolute error is defined as $|S - I|$ and the relative error is defined as $|(S - I)/I|$. The user can specify both an absolute error tolerance and a relative error tolerance, with default values given by 10^{-10} and 10^{-6}, respectively. MATLAB will refine the integration intervals until either the absolute or relative error tolerance is lower than the default (or specified) values.

In practice, one begins a calculation with the default error tolerances and if the accuracy of the solution is unsatisfactory, then the error tolerances can be lowered. Consult the MATLAB help page to learn how to do this. A good rule of thumb is to lower each error tolerance by a factor of ten until you are satisfied. The absolute error tolerance usually comes into play when the integral is close to zero.

The default call is `I=integral(f,xmin,xmax)`, where `f` is the integrand and `xmin` and `xmax` are the limits of integration. The passing of `f` to `integral.m` is similar to what we have done before.

Compute the following integral:

$$I = \int_0^\infty e^{-ax^2} (\ln x)^2 \, dx, \qquad a = \pi.$$

The following MATLAB commands compute the integral:

```
>> f = @(x,a) exp(-a*x.^2).*log(x).^2;
>> a = pi; I = integral(@(x) f(x,a), 0, Inf)
I =
    1.8245
>>
```

Problems for Lecture 42

1. Consider the Fresnel integrals, defined by

$$C(t) = \int_0^t \cos\left(\frac{1}{2}\pi x^2\right) dx, \qquad S(t) = \int_0^t \sin\left(\frac{1}{2}\pi x^2\right) dx.$$

Write a script using `integral.m` to plot a Cornu spiral, which is a smooth curve of $C(t)$ versus $S(t)$. Plot your solution over the range $-8 \leq t \leq 8$.

Solutions to the Problems

Lecture 43 | Interpolation

View this lecture on YouTube

Given the y-values of a function sampled at an ordered sequence of x-values, i.e., (x_0, y_0), (x_1, y_1), $\ldots$, (x_n, y_n), we want to estimate the y-value for some other value of x. When $x_0 \leq x \leq x_n$, the problem is called interpolation; when $x < x_0$ or $x > x_n$, the problem is called extrapolation and is much more perilous. The problem of interpolation is basically one of drawing a smooth curve that passes through the given $n+1$ points.

It is possible to interpolate $n+1$ known points by a unique polynomial of degree n. With only two points, the polynomial is a linear function; with only three points, the polynomial is a quadratic function, and so on. Although low-order polynomials are sometimes helpful when interpolating only a few points, high-order polynomials tend to over-oscillate and not be useful.

In this course, we will learn about the more widely-used piecewise polynomial interpolation. The two most popular are piecewise linear interpolation and cubic spline interpolation. The first makes use of linear polynomials, and the second cubic polynomials. Piecewise linear interpolation is the default interpolation typically used when plotting data, and we start here.

Suppose the interpolating function is $y = g(x)$, and there are $n+1$ points to interpolate. We construct the function $g(x)$ out of n local linear polynomials. We write

$$g(x) = g_i(x), \qquad \text{for } x_i \leq x \leq x_{i+1},$$

where

$$g_i(x) = a_i(x - x_i) + b_i, \qquad \text{for } i = 0, 1, \ldots, n-1.$$

We now require $y = g_i(x)$ to pass through the endpoints (x_i, y_i) and (x_{i+1}, y_{i+1}). We have

$$y_i = b_i, \qquad y_{i+1} = a_i(x_{i+1} - x_i) + b_i.$$

The solution for the coefficients of $g_i(x)$ is easily determined to be

$$a_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \qquad b_i = y_i.$$

Although piecewise linear interpolation is widely used, particularly in plotting routines, it suffers from discontinuities in the derivative. This results in a function which doesn't look smooth if the points are too widely spaced. One may increase the number of function evaluations, but sometimes the computational cost to do so is too high. In the next two lectures we will learn about cubic spline interpolation that uses piecewise cubic polynomials.

Problems for Lecture 43

1. Consider the points $(0,0)$, $(1,1)$ and $(2,1)$.

a) Find the quadratic polynomial that interpolates these points. What are the interpolated y-values at $x = 1/2$ and $x = 3/2$?

b) Find the two piecewise linear polynomials that interpolate these points. What are the interpolated y-values at $x = 1/2$ and $x = 3/2$?

c) Use MATLAB to plot the three points and the two interpolating functions.

Solutions to the Problems

Lecture 44 | Cubic spline interpolation (Part A)

View this lecture on YouTube

For cubic spline interpolation, we define piecewise cubic polynomials by

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \qquad \text{for } i = 0 \text{ to } n-1 \text{ and } x_i \le x \le x_{i+1}.$$

Requiring the global interpolating function $y = g(x)$ to go through all $n+1$ points and be continuous results in the two constraints

$$g_i(x_i) = y_i, \qquad g_i(x_{i+1}) = y_{i+1}, \qquad \text{for } i = 0 \text{ to } n-1.$$

To achieve a smooth interpolation, we further require the first and second derivatives of $g(x)$ to be continuous. This results in the additional constraints

$$g_i'(x_{i+1}) = g_{i+1}'(x_{i+1}), \qquad g_i''(x_{i+1}) = g_{i+1}''(x_{i+1}), \qquad \text{for } i = 0 \text{ to } n-2.$$

There are n piecewise cubic polynomials $g_i(x)$ and each cubic polynomial has four free coefficients, for a total of $4n$ coefficients. The number of constraining equations is $2n + 2(n-1) = 4n - 2$ so that we are missing two equations to obtain a unique solution. These equations usually derive from some extra conditions imposed on $g_0(x)$ and $g_{n-1}(x)$. We will discuss this in the next lecture.

We define now for convenience

$$h_i = x_{i+1} - x_i, \qquad \eta_i = y_{i+1} - y_i, \qquad \text{for } i = 0 \text{ to } n-1.$$

Calculating the derivatives and substituting in the end points, our constraints lead directly to the equations

$$\begin{aligned} d_i &= y_i, & a_i h_i^3 + b_i h_i^2 + c_i h_i &= \eta_i, & &\text{for } i = 0 \text{ to } n-1; \\ 3a_i h_i^2 + 2b_i h_i + c_i &= c_{i+1}, & 6a_i h_i + 2b_i &= 2b_{i+1}, & &\text{for } i = 0 \text{ to } n-2. \end{aligned}$$

It will be useful later for us to include a definition of the coefficient b_n, which is now missing. We simply extend the fourth of the above equations to $i = n-1$ (and thus add one more equation and one more coefficient). How to solve these equations will be the focus of our next lecture.

Problems for Lecture 44

1. Let $y = f(x)$ have known values (x_0, y_0), (x_1, y_1), ..., (x_n, y_n), and define piecewise cubic polynomials by

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \qquad \text{for } i = 0 \text{ to } n-1 \text{ and } x_i \le x \le x_{i+1}.$$

Suppose that the endpoint slopes $f'(x_0) = y'_0$ and $f'(x_n) = y'_n$ are known. From these two extra conditions, determine two extra constraints on the a, b and c coefficients.

Solutions to the Problems

Lecture 45 | Cubic spline interpolation (Part B)

View this lecture on YouTube

We now solve for the a, b, c, and d coefficients. The equations are

$$d_i = y_i, \qquad a_i h_i^3 + b_i h_i^2 + c_i h_i = \eta_i, \qquad 3a_i h_i + b_i = b_{i+1}, \qquad \text{for } i = 0 \text{ to } n-1;$$

$$3a_i h_i^2 + 2b_i h_i + c_i = c_{i+1}, \qquad \text{for } i = 0 \text{ to } n-2.$$

We can use the third equation to eliminate a_i:

$$a_i = \frac{1}{3h_i}\left(b_{i+1} - b_i\right), \quad \text{for } i = 0 \text{ to } n-1.$$

Then the second equation becomes

$$c_i = \frac{1}{h_i}\left(\eta_i - a_i h_i^3 - b_i h_i^2\right) = \frac{\eta_i}{h_i} - \frac{1}{3}h_i\left(b_{i+1} + 2b_i\right), \qquad \text{for } i = 0 \text{ to } n-1.$$

Finally, eliminating the a and c coefficients from the last equation results, after some algebra, in

$$\frac{1}{3}h_i b_i + \frac{2}{3}(h_i + h_{i+1})b_{i+1} + \frac{1}{3}h_{i+1}b_{i+2} = \frac{\eta_{i+1}}{h_{i+1}} - \frac{\eta_i}{h_i}, \qquad \text{for } i = 0 \text{ to } n-2,$$

which is a system of $n-1$ equations for the $n+1$ unknowns b_0, b_1, $\ldots$, b_n. We can represent these equations in matrix form as

$$\begin{pmatrix} \ldots & \ldots & \ldots & \ldots & \text{missing} & \ldots & \ldots \\ \frac{1}{3}h_0 & \frac{2}{3}(h_0+h_1) & \frac{1}{3}h_1 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & \frac{1}{3}h_{n-2} & \frac{2}{3}(h_{n-2}+h_{n-1}) & \frac{1}{3}h_{n-1} \\ \ldots & \ldots & \ldots & \ldots & \text{missing} & \ldots & \ldots \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

$$= \begin{pmatrix} \text{missing} \\ \frac{\eta_1}{h_1} - \frac{\eta_0}{h_0} \\ \vdots \\ \frac{\eta_{n-1}}{h_{n-1}} - \frac{\eta_{n-2}}{h_{n-2}} \\ \text{missing} \end{pmatrix},$$

where the first and last equations are still to be determined. Once we solve for the b-coefficients, all the other coefficients can be found and the interpolating polynomial is determined.

The default way to specify the first and last equations is called the not-a-knot condition, which assumes that

$$g_0(x) = g_1(x), \qquad g_{n-2}(x) = g_{n-1}(x).$$

Consider the first constraint. Now, two cubic polynomials are equal if at some fixed x, the polynomials and their first three derivatives are equal. The cubic spline continuity conditions already ensure that $g_0(x_1) = g_1(x_1)$, $g_0'(x_1) = g_1'(x_1)$ and $g_0''(x_1) = g_1''(x_1)$. Therefore, the two polynomials are equal provided $g_0'''(x_1) = g_1'''(x_1)$, or $a_0 = a_1$. In terms of the b coefficients, this condition is given by

$$h_1 b_0 - (h_0 + h_1) b_1 + h_0 b_2 = 0,$$

which we can use as our missing first equation. A similar argument made for the second constraint yields for the missing last equation

$$h_{n-1} b_{n-2} - (h_{n-2} + h_{n-1}) b_{n-1} + h_{n-2} b_n = 0.$$

Problems for Lecture 45

1. Consider the points $(0,0)$, $(1,1)$, $(2,1)$ and $(3,2)$. Using the not-a-knot condition, determine the four-by-four matrix equation for the b coefficients. Solve for the b's as well as the a's, c's and d's, and thus find the cubic spline interpolant. Plot your result. You may use MATLAB to assist in your algebra.

Solutions to the Problems

Lecture 46 | Interpolation in MATLAB

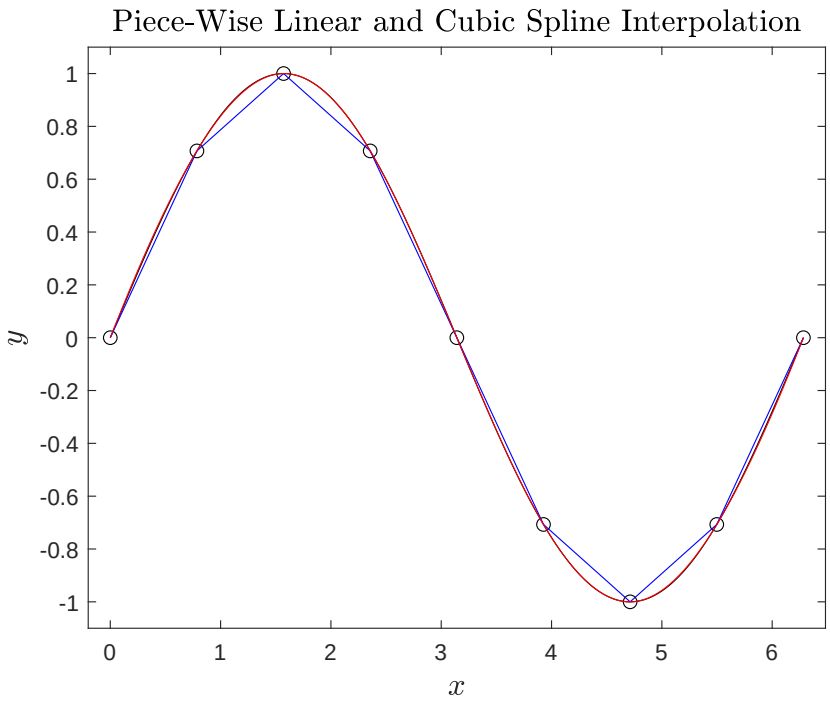
View this lecture on YouTube

The MATLAB function `interp1.m` performs interpolation of a tabulated function. It is mainly called using `yy=interp1(x, y, xx, method);` where `x` and `y` are vectors containing the tabulated x-y function values and `xx` is a vector on whose values you want the function to be interpolated, with `yy` the returned interpolated values.

Several interpolation methods are possible, but the two we have discussed in this week assign `method` the values `'linear'` or `'spline'`, where cubic spline interpolation uses the not-a-knot condition. There is also another MATLAB function `spline.m` that can use other conditions.

As an example, we show how to use `interp1.m` to interpolate the points on a sine curve using either piece-wise linear or cubic spline interpolation.

```
x=0:pi/4:2*pi; y=sin(x);
plot(x,y,'ok'); hold on;
xx=linspace(0,2*pi,1000);
plot(xx,sin(xx),'k');
yy=interp1(x,y,xx,'linear');
plot(xx,yy,'b');
yy=interp1(x,y,xx,'spline');
plot(xx,yy, 'r');
```



Problems for Lecture 46

1. Load the following MATLAB MAT-files:

```
data1.mat
data2.mat
```

The file `data1.mat` contains the variables `x1` and `y1`. The file `data2.mat` contains the variables `x2` and `y2`. Use cubic spines to interpolate both `y1` and `y2` to a uniform x-grid. Plot `y=y1+y2` versus `x`. Your result should look like $y = \sin x$.

Solutions to the Problems

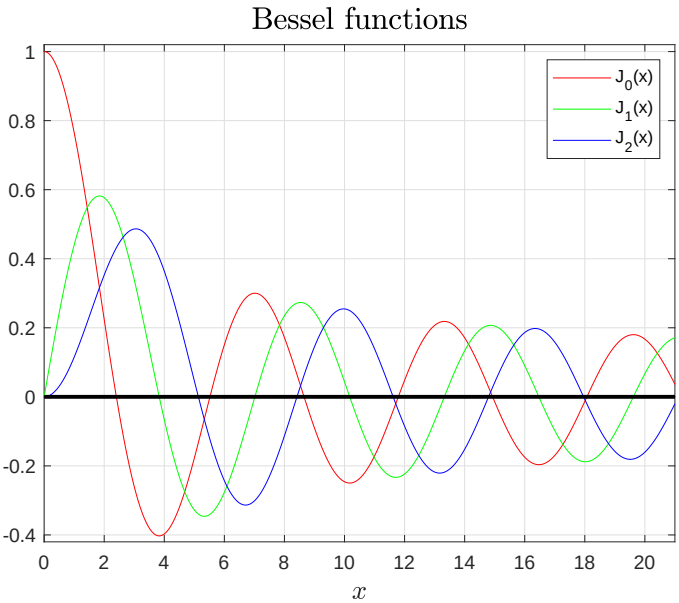
Lecture 47 | Project IV: Bessel functions and their zeros

View this lecture on YouTube

The Bessel function of order n can be defined by the definite integral

$$J_n(x) = \frac{1}{\pi} \int_0^{\pi} \cos\left(x \sin\theta - n\theta\right) d\theta, \qquad \text{for } n = 0, 1, 2, \ldots.$$

The zeros of the Bessel function play an important role in the solution of partial differential equations in cylindrical coordinates. The goal of this project is to compute the first five positive roots, $j_{n,k}$ $(k = 1, 2, \ldots, 5)$ of the first six Bessel functions $J_n(x)$ $(n = 0, 1, \ldots, 5)$ using quadrature and root-finding.



Before you start computing the roots, I would recommend plotting the Bessel functions $J_n(x)$ versus x as I have done above. From the plot, you can easily visualize the behavior of the roots. To plot the Bessel functions, you can make use of the MATLAB function `integrate.m`.

To find the roots of the Bessel functions, you can also make use of the MATLAB function `fzero.m`. Initial guesses for the roots can be obtained from your Bessel function plots, or from Wolfram MathWorld (input by hand initial guesses accurate to no more than one decimal place). Generate your roots using two `for` loops, and output the roots in tabular form similar to that of Wolfram MathWorld.

Problems for Lecture 47

1. The Bessel function of order n, for $n = 0, 1, 2, \ldots$, can be defined by the definite integral

$$J_n(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin\theta - n\theta)\, d\theta.$$

Compute the first five positive roots $j_{n,k}$, $(k = 1, 2, \ldots, 5)$, of the first six Bessel functions $J_n(x)$, $(n = 0, 1, \ldots, 5)$.

Solutions to the Problems

Week V

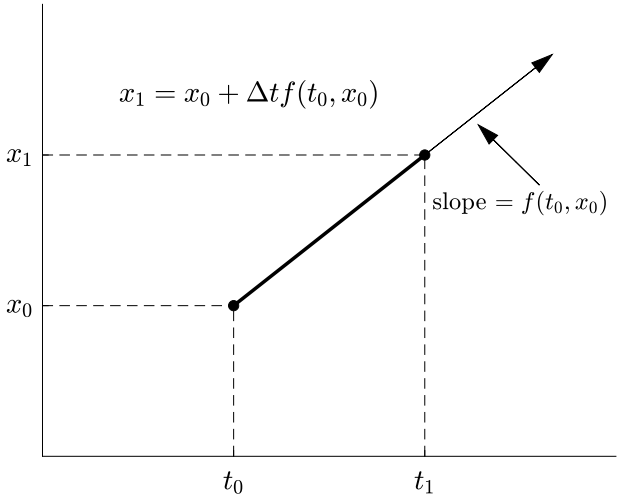
Ordinary Differential Equations

In this week's lectures, we learn about the numerical integration of odes. The most basic method is called the Euler method, and it is a single-step, first-order method. The Runge-Kutta methods extend the Euler method to multiple steps and higher order, with the advantage that larger time-steps can be made. We show how to construct a family of second-order Runge-Kutta methods, and introduce you to the widely-used fourth-order Runge-Kutta method. These methods are easily adopted for solving systems of odes. We will show you how to use the MATLAB *function* `ode45`*, and how to solve a two-point boundary value ode using the shooting method.*

Your programming project will be to write a MATLAB *code to compute the numerical simulation of the gravitational two-body problem.*

Lecture 48 | Euler method

View this lecture on YouTube



The Euler method is the most straightforward method to integrate a differential equation. The first-order differential equation $\dot{x} = f(t, x)$, with initial condition $x(t_0) = x_0$, provides the slope $f(t_0, x_0)$ of the tangent line to the solution curve $x = x(t)$ at the point (t_0, x_0). With a small step size $\Delta t = t_1 - t_0$, the initial condition (t_0, x_0) can be marched forward to (t_1, x_1) along the tangent line (see figure) using $x_1 = x_0 + \Delta t f(t_0, x_0)$. The point (t_1, x_1) then becomes the new initial condition and is marched forward to (t_2, x_2) along a newly determined tangent line with slope given by $f(t_1, x_1)$. This iterative method is usually written as

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

For small enough Δt, the numerical solution should converge to the exact solution of the ode, when such a solution exists.

The Euler Method has a local error, that is, the error incurred over a single time step, of $O(\Delta t^2)$. The global error, however, comes from integrating out to a time T. If this integration takes N time steps, then the global error is the sum of N local errors. Since $N = T/\Delta t$, the global error is given by $O(\Delta t)$, and it is customary to call the Euler Method a first-order method. We next consider some common second-order methods.

Problems for Lecture 48

1. Let $\dot{x} = b$, with initial condition $x(0) = x_0$ and b a constant. With $t = n\Delta t$, show that the Euler method results in the exact solution

$$x(t) = x_0 + bt.$$

Solutions to the Problems

Lecture 49 | Modified Euler method

View this lecture on YouTube

The modified Euler method, also called Heun's method or the predictor-corrector method, is a second-order method. The idea is to average the value of $\dot{x}$ at the beginning and end of each time step. Hopefully, we could write

$$x_{n+1} = x_n + \frac{1}{2}\Delta t\big(f(t_n, x_n) + f(t_n + \Delta t, x_{n+1})\big).$$

The obvious problem with this formula is that the unknown value x_{n+1} appears on the right-hand-side. We can, however, estimate this value, in what is called the predictor step. For the predictor step, we use the Euler method to find

$$x^p_{n+1} = x_n + \Delta t f(t_n, x_n).$$

Then the corrector step becomes

$$x_{n+1} = x_n + \frac{1}{2}\Delta t\big(f(t_n, x_n) + f(t_n + \Delta t, x^p_{n+1})\big).$$

The Modified Euler method is usually coded as

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f(t_n + \Delta t, x_n + k_1),$$
$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2).$$

The modified Euler method is one of a family of methods called second-order Runge-Kutta methods, which we derive in the next lecture.

Problems for Lecture 49

1. Let $\dot{x} = bt$, with initial condition $x(0) = x_0$ and b a constant. With $t = n\Delta t$, show that the Modified Euler method results in the exact solution

$$x(t) = x_0 + \frac{1}{2}bt^2.$$

Solutions to the Problems

Lecture 50 | Runge-Kutta methods

View this lecture on YouTube

To illustrate the derivation of Runge-Kutta methods, we derive here the complete family of second-order methods. We march the solution of $\dot{x} = f(t, x)$ forward by writing

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f(t_n + \alpha\Delta t, x_n + \beta k_1),$$
$$x_{n+1} = x_n + ak_1 + bk_2,$$

where α, β, a and b are constants that define particular second-order methods. We will constrain the values of these constants by computing the Taylor series of x_{n+1} in two ways.

First, we compute the Taylor series for x_{n+1} directly:

$$x_{n+1} = x(t_n + \Delta t) = x(t_n) + \Delta t\dot{x}(t_n) + \frac{1}{2}(\Delta t)^2\ddot{x}(t_n) + \mathrm{O}(\Delta t^3).$$

Now, $\dot{x}(t_n) = f(t_n, x_n)$. The second derivative is more tricky and requires partial derivatives. We have

$$\ddot{x}(t_n) = \frac{d}{dt}f(t, x(t))\bigg|_{t=t_n} = f_t(t_n, x_n) + \dot{x}(t_n)f_x(t_n, x_n) = f_t(t_n, x_n) + f(t_n, x_n)f_x(t_n, x_n).$$

Putting all the terms together, we obtain

$$x_{n+1} = x_n + \Delta t f(t_n, x_n) + \frac{1}{2}(\Delta t)^2\left(f_t(t_n, x_n) + f(t_n, x_n)f_x(t_n, x_n)\right) + \mathrm{O}(\Delta t^3). \tag{50.1}$$

Second, we compute the Taylor series for x_{n+1} from the Runge-Kutta formula. We start with

$$x_{n+1} = x_n + a\Delta t f(t_n, x_n) + b\Delta t f\big(t_n + \alpha\Delta t, x_n + \beta\Delta t f(t_n, x_n)\big) + \mathrm{O}(\Delta t^3);$$

and the Taylor series that we need is

$$f\big(t_n + \alpha\Delta t, x_n + \beta\Delta t f(t_n, x_n)\big) = f(t_n, x_n) + \alpha\Delta t f_t(t_n, x_n) + \beta\Delta t f(t_n, x_n)f_x(t_n, x_n) + \mathrm{O}(\Delta t^2).$$

The Taylor-series for x_{n+1} from the Runge-Kutta method is therefore given by

$$x_{n+1} = x_n + (a + b)\Delta t f(t_n, x_n) + (\Delta t)^2\big(\alpha b f_t(t_n, x_n) + \beta b f(t_n, x_n)f_x(t_n, x_n)\big) + \mathrm{O}(\Delta t^3). \tag{50.2}$$

Comparing (50.1) and (50.2), we find three constraints for the four constants:

$$a + b = 1, \quad \alpha b = 1/2, \quad \beta b = 1/2.$$

Lecture 51 | Second-order Runge-Kutta methods

View this lecture on YouTube

The family of second-order Runge-Kutta methods that solve $\dot{x} = f(t, x)$ is given by

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f(t_n + \alpha \Delta t, x_n + \beta k_1),$$
$$x_{n+1} = x_n + a k_1 + b k_2,$$

where we have derived three constraints for the four constants α, β, a and b:

$$a + b = 1, \quad \alpha b = 1/2, \quad \beta b = 1/2.$$

The modified Euler method corresponds to $\alpha = \beta = 1$ and $a = b = 1/2$. The function $f(t, x)$ is evaluated at the times $t = t_n$ and $t = t_n + \Delta t$, and we have

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f(t_n + \Delta t, x_n + k_1),$$
$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2).$$

The midpoint method corresponds to $\alpha = \beta = 1/2$, $a = 0$ and $b = 1$. In this method, the function $f(t, x)$ is evaluated at the times $t = t_n$ and $t = t_n + \Delta t/2$ and we have

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right),$$
$$x_{n+1} = x_n + k_2.$$

Problems for Lecture 51

1. Construct Ralston's method, which is a second-order Runge-Kutta method corresponding to $\alpha = \beta = 3/4$, $a = 1/3$ and $b = 2/3$.

2. Consider the ode given by

$$\frac{dy}{dx} = f(x),$$

with $y(0)$ as the initial value. Use the second-order Runge-Kutta methods given by the midpoint rule and the modified Euler method to derive two elementary quadrature formulas.

Solutions to the Problems

Lecture 52 | Higher-order Runge-Kutta methods

View this lecture on YouTube

Higher-order Runge-Kutta methods can also be derived, but require substantially more algebra. For example, the general form of the third-order method is given by

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f(t_n + \alpha\Delta t, x_n + \beta k_1), \qquad k_3 = \Delta t f(t_n + \gamma\Delta t, x_n + \delta k_1 + \epsilon k_2),$$
$$x_{n+1} = x_n + ak_1 + bk_2 + ck_3,$$

with constraints on the constants α, β, γ, δ, ϵ, a, b and c. The fourth-order method has stages k_1, k_2, k_3 and k_4. The fifth-order method requires at least six stages. The table below gives the order of the method and the minimum number of stages required.

order	2	3	4	5	6	7	8
minimum # stages	2	3	4	6	7	9	11

Because the fifth-order method requires two more stages than the fourth-order method, the fourth-order method has found some popularity. The general fourth-order method with four stages has 13 constants and 11 constraints. A particularly simple fourth-order method that has been widely used in the past by physicists is given by

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_1\right),$$
$$k_3 = \Delta t f\left(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2}k_2\right), \qquad k_4 = \Delta t f\left(t_n + \Delta t, x_n + k_3\right);$$

$$x_{n+1} = x_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

Problems for Lecture 52

1. Consider the ode given by

$$\frac{dy}{dx} = f(x),$$

with $y(0)$ as the initial value. Use the standard fourth-order Runge-Kutta method to derive Simpson's rule.

Solutions to the Problems

Lecture 53 | Higher-order odes and systems

View this lecture on YouTube

Our numerical methods can be easily adapted to solve higher-order odes, which are equivalent to a system of first-order odes. As an example, consider the second-order ode given by

$$\ddot{x} = f(t, x, \dot{x}).$$

This second-order ode can be rewritten as a system of two first-order odes by defining $u = \dot{x}$. We then obtain

$$\dot{x} = u, \qquad \dot{u} = f(t, x, u).$$

This trick also works for higher-order odes. For example, the third-order ode,

$$\dddot{x} = f(t, x, \dot{x}, \ddot{x}),$$

becomes

$$\dot{x} = u, \qquad \dot{u} = v, \qquad \dot{v} = f(t, x, u, v).$$

We can easily generalize the Runge-Kutta method to solve a system of first-order odes. As an example, we solve the following system of two first-order odes,

$$\dot{x} = f(t, x, y), \quad \dot{y} = g(t, x, y).$$

The generalization of the modified Euler method, say, would be

$$\begin{aligned} k_1 &= \Delta t f(t_n, x_n, y_n), & l_1 &= \Delta t g(t_n, x_n, y_n), \\ k_2 &= \Delta t f(t_n + \Delta t, x_n + k_1, y_n + l_1), & l_2 &= \Delta t g(t_n + \Delta t, x_n + k_1, y_n + l_1); \end{aligned}$$

$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2), \qquad y_{n+1} = y_n + \frac{1}{2}(l_1 + l_2),$$

where each stage requires computation of both a k and a l.

Problems for Lecture 53

1. Write down the modified Euler method for the system of equations given by

$$\dot{x} = f(t, x, y, z), \qquad \dot{y} = g(t, x, y, z), \qquad \dot{z} = h(t, x, y, z).$$

Solutions to the Problems

Lecture 54 | Adaptive Runge-Kutta methods

View this lecture on YouTube

An adaptive ode solver automatically finds the best integration step-size Δt at each time step. The Dormand-Prince method, which is implemented in MATLAB's most widely used solver, `ode45.m`, determines the step size by comparing the results of fourth- and fifth-order Runge-Kutta methods. This solver requires six function evaluations per time step, and saves computational time by constructing both fourth- and fifth-order methods using the same function evaluations.

Suppose the fifth-order method finds x_{n+1} with local error $O(\Delta t^6)$, and the fourth-order method finds X_{n+1} with local error $O(\Delta t^5)$. Let ε be the requested error tolerance, and let e be the actual error. We can estimate e from the difference between the fifth- and fourth-order methods; that is,

$$e = |x_{n+1} - X_{n+1}|.$$

Now, we know that e is of $O(\Delta t^5)$. Let $\Delta\tau$ be the estimated step size required for the requested error ε. Then, by the scaling of the errors, we have

$$e/\varepsilon = (\Delta t)^5/(\Delta\tau)^5,$$

or solving for $\Delta\tau$,

$$\Delta\tau = \Delta t \left(\frac{\varepsilon}{e}\right)^{1/5}.$$

On the one hand, if the actual error is greater than the desired error, $e > \varepsilon$, then we reject the integration step and redo the time step using the smaller value $\Delta\tau$. On the other hand, if the actual error is less that the requested error, $e < \varepsilon$, then we accept x_{n+1} and increase the next time step to the larger value $\Delta\tau$. In practice, one usually includes a safety factor when computing $\Delta\tau$, i.e.,

$$\Delta\tau = S\Delta t \left(\frac{\varepsilon}{e}\right)^{1/5},$$

with $S = 0.9$, say, to prevent the wastefulness of too many failed time steps.

Problems for Lecture 54

1. Using the Dormand-Prince method, suppose that a user requests an error tolerance of $\varepsilon = 10^{-6}$, and suppose the time step attempted was $\Delta t = 0.01$ and that $e = |x_{n+1} - X_{n+1}| = 1.1 \times 10^{-6}$. Is the current time step accepted? What time step will be used next? Assume a safety factor of 0.9.

Solutions to the Problems

Lecture 55 | Integrating odes in MATLAB (Part A)

View this lecture on YouTube

The general purpose MATLAB ode solver is `ode45`, and we discuss how to use it here. A single first-order ode is easily integrated using a technique similar to what we have already learned for both root finding and quadrature. For example, to numerically solve the logistic equation $\dot{x} = rx(1-x)$ and plot x versus t, we can code

```
dxdt = @(x,r) r*x*(1-x);
r=1; x0=0.01; tspan=[0 10];
[t,x] = ode45(@(t,x) dxdt(x,r), tspan, x0);
plot(t,x);
```

The right-hand side of the ode is defined as the function handle `dxdt` and this is passed as the first argument to `ode45`. Here, the right-hand side of the ode depends on x and a single parameter r, but in general, it can also depend on t and have any number of parameters. The integrator `ode45`, however, expects a function that has the independent variable as the first argument, and the dependent variables (written as a vector) as the second argument. To satisfy this requirement, we give `ode45` the handle to an anonymous function, so that its first argument looks like `@(t,x) dxdt(x,r)`. For another example, if the right-hand side of the ode also depends on t and another parameter s, we may give as the first argument to `ode45` the expression `@(t,x) dxdt(t,x,r,s)`.

For a more complicated system of odes, it may be more convenient to define a sub-function. For example, to solve the Lotka-Volterra equations, $\dot{x} = rx(1-y)$, $\dot{y} = \frac{1}{r}y(x-1)$ and plot x and y versus t as well as y versus x, we can code

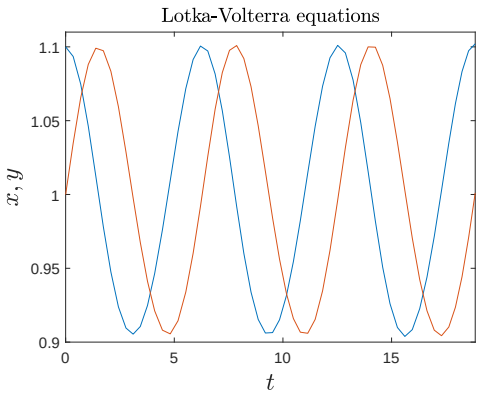
```
r=1; x0=1.1; y0=1; tspan=[0 6*pi];
[t,xy]=ode45(@(t,xy) lv(xy,r),tspan,[x0 y0]);
figure(1); plot(t,xy); xlabel('t'); ylabel('x, y');
figure(2); plot(xy(:,1),xy(:,2)); xlabel('x'); ylabel('y');

function dxydt = lv(xy,r)
x=xy(1); y=xy(2);
dxydt=[r*x*(1-y); (1/r)*y*(x-1)];
end
```

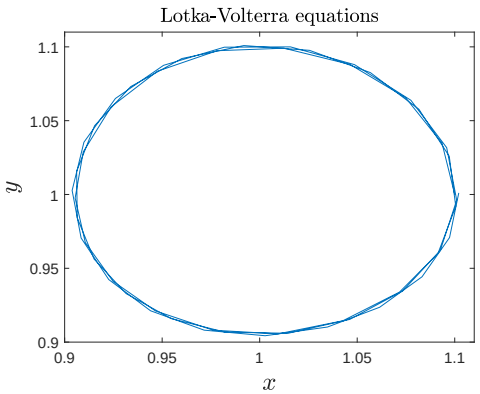
Some comments on the syntax. First, the independent variable t is returned as a column vector and the dependent variables `x,y` are returned as the two columns of a matrix. Second, the function `ode45.m` knows we are solving a system of two first-order odes because we give it a two-component initial condition. Third, the right-hand sides of the differential equations `dxydt` must be written as a column vector. If they are written as a row vector, you will get an error message.

Lecture 56 | Integrating odes in MATLAB (Part B)

View this lecture on YouTube

The graphs output by MATLAB from the Lotka-Volterra code are shown below:





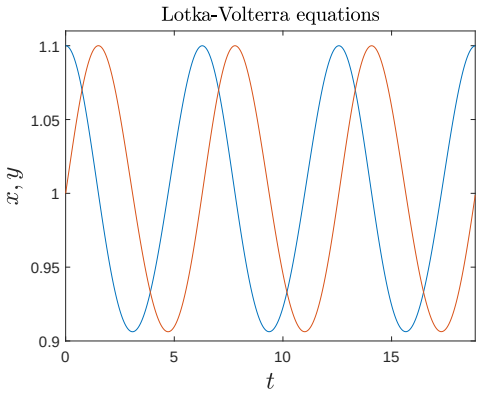
A careful inspection of the figures reveals two anomalies. First, in the plot of x and y versus t, there are not enough data points in the output for a smooth plot, particularly at the maxima and minima of the functions where the linear interpolation of the data is evident. Second, in the phase-space plot of y versus x, it is unclear whether the solution is exactly periodic since overlapping curves are evident.

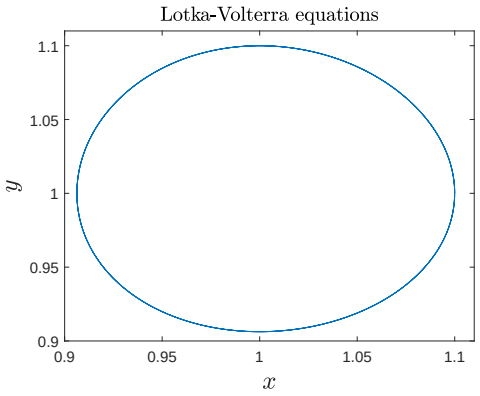
We can modify our call to `ode45` to improve these plots. The integrator `ode45` can do its own interpolation at little expense if you define `tspan` to have more than two elements. For example, we can replace `tspan=[0 6*pi]` by the code `tspan=linspace(0,6*pi,1000)`.

After rerunning the code, you will observe that although the plot of x and y versus t is now smooth, the plot of y versus x still shows overlapping curves. The problem here is with the default error tolerance. There are two tolerances that are set: `RelTol` and `AbsTol`. Roughly speaking, `RelTol` sets the number of significant digits in the computed solution, and `AbsTol` sets the error tolerance as the solution approaches zero. The default values of these two tolerances are `RelTol = 1.e-3`, corresponding to approximately three significant digits, and `AbsTol=1.e-6`. Here, the phase-space plot may be improved by decreasng `RelTol`. The appropriate change to the MATLAB program is done in the following two lines of code:

```
options = odeset('RelTol',1.e-4);
[t,xy]=ode45(@(t,xy) lv(xy,r),tspan,[x0; y0],options);
```

The improved plots are shown on the next page. This code runs fast, but it should be noted that for more computationally intensive calculations, smaller values for the error tolerances will result in longer computational times.


Lotka-Volterra equations
1.1
1.05
1
0.95
0.9
x, y
0
5
10
15
t


Lotka-Volterra equations
1.1
1.05
1
0.95
0.9
y
0.9
0.95
1
1.05
1.1
x

Problems for Lecture 56

1. The Lorenz equations are a system of nonlinear odes that pioneered the study of chaos. The Lorenz equations are given by

$$\dot{x} = \sigma(y - x), \qquad \dot{y} = x(r - z) - y, \qquad \dot{z} = xy - \beta z,$$

where σ, β and r are constants. Edward Lorenz studied the solution for $\sigma = 10$, $\beta = 8/3$ and $r = 28$, and the result is now known as the Lorenz attractor, an example of what is now more generally known as a strange attractor. Compute the Lorenz attractor and plot z versus x and y. Use the MATLAB function `plot3.m`. Remove the transient before plotting.

Solutions to the Problems

Lecture 57 | Shooting method for boundary value problems

View this lecture on YouTube

We consider the general second-order ode of the form

$$\frac{d^2y}{dx^2} = f(x, y, dy/dx),$$

with the two-point boundary values $y(x_0) = y_0$ and $y(x_f) = y_f$. To develop a solution method, we first formulate the second-order ode as a system of two first-order odes. We have

$$\frac{dy}{dx} = z, \qquad \frac{dz}{dx} = f(x, y, z).$$

One initial condition is known, $y(x_0) = y_0$, but the second initial condition for $z(x_0)$ is unknown. The aim of the shooting method is to determine the value of $z(x_0)$ that results in $y(x_f) = y_f$.

This is a root-finding problem. Let the variable ξ denote the unknown value $z(x_0)$. We define $F = F(\xi)$ as

$$F(\xi) = y(x_f) - y_f,$$

where $y(x_f)$ is the value obtained by integrating the differential equations to $x = x_f$ using the initial conditions $y(x_0) = y_0$ and $z(x_0) = \xi$. The solution to $F(\xi) = 0$ determines the value of $\xi = z(x_0)$ that solves the two-point boundary value problem. This method is called *shooting* because the slope of the function $y = y(x)$ at $x = x_0$ plays the role of aiming the gun, with the intention to hit the target at $y(x_f) = y_f$.

To determine the value of ξ that solves $F(\xi) = 0$, we can make use of the MATLAB function `fzero`, together with the differential equation solver `ode45`.

Problems for Lecture 57

1. The dimensionless, unforced pendulum equation is given by

$$\ddot{\theta} + \alpha\dot{\theta} + \sin\theta = 0,$$

where α is the only free parameter.

Consider initial conditions with the mass at the bottom, $\theta(0) = 0$. Using the shooting method, determine the smallest positive value of $\dot{\theta}(0)$ such that the mass becomes exactly balanced at the top ($\theta = \pi$). Plot this value of $\dot{\theta}(0)$ versus α for $0 \leq \alpha \leq 2$.

Solutions to the Problems

Lecture 58 | Project V: Two-body problem (Part A)

View this lecture on YouTube

Consider two masses m_1 and m_2 with position vectors $\mathbf{r}_1$ and $\mathbf{r}_2$. Newton's second law and the universal law of gravitation yields for the governing equations

$$m_1\ddot{\mathbf{r}}_1 = \mathbf{F}, \qquad m_2\ddot{\mathbf{r}}_2 = -\mathbf{F}, \quad \text{where} \quad \mathbf{F} = -Gm_1m_2\frac{\mathbf{r}_1 - \mathbf{r}_2}{|\mathbf{r}_1 - \mathbf{r}_2|^3},$$

where G is the gravitational constant. The two-body problem can be reduced to a one-body problem by a change of coordinates. Let $\mathbf{r}$ be the relative position vector and $\mathbf{R}$ be the coordinate of the center of mass; that is,

$$\mathbf{r} = \mathbf{r_1} - \mathbf{r_2}, \qquad \mathbf{R} = \frac{m_1\mathbf{r_1} + m_2\mathbf{r_2}}{m_1 + m_2}.$$

Solving for $\mathbf{r_1}$ and $\mathbf{r_2}$ in terms of $\mathbf{r}$ and $\mathbf{R}$ gives

$$\mathbf{r_1} = \mathbf{R} + \frac{m_2\mathbf{r}}{m_1 + m_2}, \qquad \mathbf{r_2} = \mathbf{R} - \frac{m_1\mathbf{r}}{m_1 + m_2}.$$

Now, changing coordinates in Newton's equations from $\mathbf{r_1}$ and $\mathbf{r_2}$ to $\mathbf{r}$ and $\mathbf{R}$ results in

$$m_1\ddot{\mathbf{R}} + \frac{m_1m_2\ddot{\mathbf{r}}}{m_1 + m_2} = \mathbf{F}, \qquad m_2\ddot{\mathbf{R}} - \frac{m_1m_2\ddot{\mathbf{r}}}{m_1 + m_2} = -\mathbf{F};$$

and adding and subtracting these two equations yields the result

$$\ddot{\mathbf{R}} = 0, \qquad \mu\ddot{\mathbf{r}} = \mathbf{F},$$

where μ is called the reduced mass and is given by

$$\mu = \frac{m_1m_2}{m_1 + m_2}.$$

The center-of-mass coordinate system is an inertial coordinate system (one moving at constant velocity) with the center of mass fixed at the origin ($\mathbf{R} = 0$). The remaining equation for the relative coordinate has the form of a one-body problem, and can be written as

$$\ddot{\mathbf{r}} = -k\frac{\mathbf{r}}{r^3}, \qquad \text{where} \quad k = G(m_1 + m_2).$$

Once $\mathbf{r}$ is determined, the coordinates of the two masses in the center of mass coordinate system can be found from

$$\mathbf{r_1} = \left(\frac{m_2}{m_1 + m_2}\right)\mathbf{r}, \qquad \mathbf{r_2} = -\left(\frac{m_1}{m_1 + m_2}\right)\mathbf{r}.$$

Problems for Lecture 58

1. Consider the two-body problem where the solution for the relative coordinates is a circular orbit of unit radius, that is,

$$\mathbf{r} = \cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}.$$

Sketch the orbits of m_1 and m_2 for (a) $m_1 = m_2$ and (b) $m_1 = 3m_2$.

Solutions to the Problems

Lecture 59 | Project V: Two-body problem (Part B)

View this lecture on YouTube

The equivalent one-body problem can be simplified further. The position vector $\mathbf{r}$ and the velocity vector $\dot{\mathbf{r}}$ form a plane, and since the acceleration vector points in the direction of the position vector, the one body motion will be restricted to this plane. We can place our coordinate system so that $\mathbf{r}$ lies in the x-y plane with $z = 0$, and we can write $\mathbf{r} = x\mathbf{i} + y\mathbf{j}$.

The relevant units are length and time, and they may be nondimensionalized using the constant parameter k (which has units l^3/t^2) and the distance of closest approach of the two masses (the minimum value of r). The dimensionless governing equations, in component form, are then given by

$$\ddot{x} = -\frac{x}{(x^2+y^2)^{3/2}}, \qquad \ddot{y} = -\frac{y}{(x^2+y^2)^{3/2}}.$$

We can further orient the x-y axes so that the solution $\mathbf{r} = \mathbf{r}(t)$ is symmetric about the x-axis, and the minimum value of r occurs at $x = -1$ and $y = 0$, where symmetry also requires $\dot{x} = 0$. Setting our initial conditions at the point of minimum r, we can write

$$x(0) = -1, \quad y(0) = 0, \quad \dot{x}(0) = 0, \quad \dot{y}(0) = \sqrt{1+e},$$

where we have parameterized $\dot{y}(0)$ using e. The parameter e is called the eccentricity of the orbit, and some further analysis can show that closed elliptical orbits correspond to $0 < e < 1$ and open hyperbolic orbits correspond to $e > 1$. The orbit with $e = 0$ is circular and the orbit with $e = 1$ is parabolic. Furthermore, the period of a closed orbit is given by $T = 2\pi/(1-e)^{3/2}$.

For this project, you will write a code which solves the two-body problem. Given the initial conditions with specified value of e, the solution for $\mathbf{r} = \mathbf{r}(t)$ for several periods of an orbit can be computed using the MATLAB function `ode45.m` by solving a system of four first-order differential equations. The motion of the corresponding two masses can then be simulated after specifying the ratio of the masses.

Problems for Lecture 59

1. By solving a system of differential equations, determine the orbit of two masses using Newton's law and the universal law of gravitation. Display an animation of the orbit.

Solutions to the Problems

Week VI

Partial Differential Equations

In this week's lectures, we learn how to find numerical solutions of some partial differential equations. This is a vast topic, and research areas such as computational fluid dynamics have many specialized solution methods.

Here, we only provide a taste of this subject. We divide the numerical solutions of pdes into boundary value problems and initial value problems, and apply the finite difference method of solution. We first show how to solve the Laplace equation, a boundary value problem. Two methods are illustrated: a direct method where the solution is found by Gaussian elimination; and an iterative method, where the solution is approached asymptotically. Second, we show how to solve the one-dimensional diffusion equation, an initial value problem. The Crank-Nicolson method of solution is derived. We also show how to apply the Von Neumann stability analysis to determine the stability of our time-integration schemes.

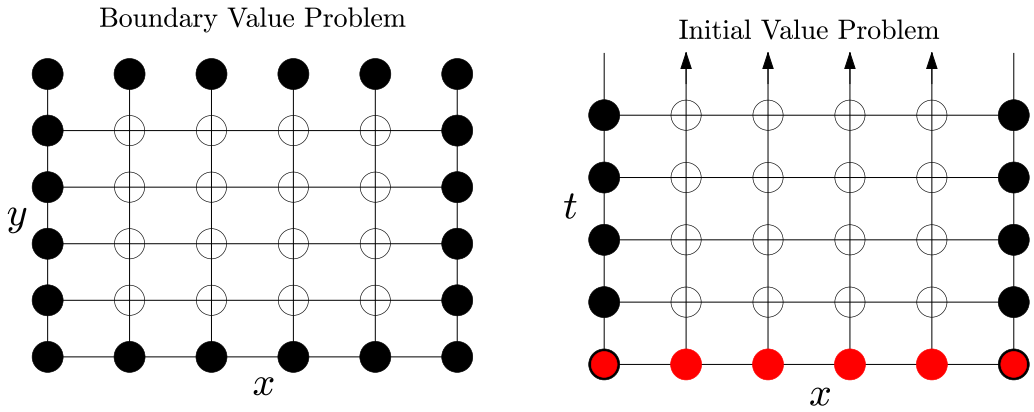
Your programming project will be to write a MATLAB *code that computes the solution of the two-dimensional diffusion equation using the Crank-Nicolson method.*

Lecture 60 | Boundary and initial value problems

View this lecture on YouTube

A pde for which a numerical solution is sought most generally divides into two fundamental types. For a boundary value problem, the function or its derivatives are known on the boundary of a domain, and we want to solve for the function inside the domain. Usually, the solution is in some type of equilibrium or steady state. For an initial value problem, the function is known at $t = 0$ throughout its spatial domain, and we want to know how the function evolves in time. Sometimes, it is possible to obtain the solution of a boundary value problem by evolving an initial value problem to a steady state. But it is often faster to solve the boundary value problem directly.

Schematics for a two-dimensional boundary value problem and one-dimensional initial value problem are shown below:



The black and red circles represent given boundary and initial values, respectively. The open circles represent the unknown solution.

Practice Quiz | Classify partial differential equations

Classify the following pde problems:

1. The steady flow of a fluid past an infinite cylinder.
   a) Boundary value problem
   b) Initial value problem
2. The turbulent flow of a fluid past an infinite cylinder.
   a) Boundary value problem
   b) Initial value problem
3. The steady-state salt concentration in a reservoir.
   a) Boundary value problem
   b) Initial value problem
4. The transient salt concentration in a reservoir after a heavy rain.
   a) Boundary value problem
   b) Initial value problem
5. The electrostatic potential surrounding two conductors held at constant potential.
   a) Boundary value problem
   b) Initial value problem
6. The transitory electrostatic potential when a conductor is moved.
   a) Boundary value problem
   b) Initial value problem

Solutions to the Practice quiz

Lecture 61 | Central difference approximation

View this lecture on YouTube

Finite difference methods are often used to solve pdes, and we need first to derive finite difference formulas for derivatives. To determine a formula for the derivative of a function $y = y(x)$, consider the Taylor series for $y(x+h)$ and $y(x-h)$ about x:

$$y(x+h) = y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \dots,$$
$$y(x-h) = y(x) - hy'(x) + \frac{1}{2}h^2y''(x) - \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \dots.$$

The standard definitions of the derivatives give the first-order approximations,

$$y'(x) = \frac{y(x+h) - y(x)}{h} + \mathrm{O}(h), \qquad y'(x) = \frac{y(x) - y(x-h)}{h} + \mathrm{O}(h),$$

where here we use the big-Oh notation, as in $\mathrm{O}(h^p)$, to denote omitted terms that go to zero no slower than h^p as h goes to zero.

The more widely-used second-order approximation for the first derivative is called the central-difference approximation and is given by

$$y'(x) = \frac{y(x+h) - y(x-h)}{2h} + \mathrm{O}(h^2).$$

The second-order approximation for the second derivative is also a central-difference approximation and can be found from considering

$$y(x+h) + y(x-h) = 2y(x) + h^2y''(x) + \frac{1}{12}h^4y''''(x) + \dots,$$

from which we obtain

$$y''(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} + \mathrm{O}(h^2).$$

Problems for Lecture 61

1. Using Taylor series approximations for $y(x+2h)$, $y(x+h)$, $y(x-h)$ and $y(x-2h)$, derive a central difference approximation for the first derivative $y'(x)$ that is accurate to $O(h^4)$.

Solutions to the Problems

Lecture 62 | Discrete Laplace equation

View this lecture on YouTube

As an example of a boundary value problem and the finite difference method, we discretize the two-dimensional Laplace equation,

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\Phi = 0,$$

on the rectangular domain $[0, L_x] \times [0, L_y]$. We form a two-dimensional grid with N_x and N_y intervals in the x- and y-direction, respectively, and $n_x = N_x + 1$ and $n_y = N_y + 1$ grid points. With grid spacing $\Delta x = L_x/N_x$ and $\Delta y = L_y/N_y$, the x and y coordinates of the grid points are given by

$$x_i = (i-1)\Delta x, \quad i = 1, 2, \ldots, n_x; \qquad\qquad y_j = (j-1)\Delta y, \quad j = 1, 2, \ldots, n_y.$$

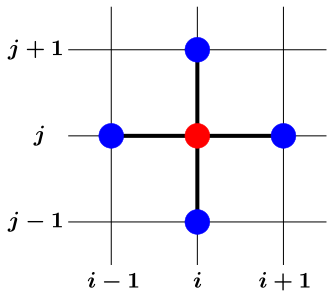
Furthermore, we denote the value of $\Phi(x_i, y_j)$ by $\Phi_{i,j}$. The central difference approximations for the second derivatives at the point (x_i, y_j) are given by

$$\frac{\partial^2\Phi}{\partial x^2} \approx \frac{1}{(\Delta x)^2}\left(\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}\right), \qquad \frac{\partial^2\Phi}{\partial y^2} \approx \frac{1}{(\Delta y)^2}\left(\Phi_{i,j+1} - 2\Phi_{i,j} + \Phi_{i,j-1}\right).$$

We consider here only square grids with $\Delta x = \Delta y$. Combining the formulas for the second derivatives and rearranging terms results in the discrete Laplace equation given by

$$4\Phi_{i,j} - \Phi_{i+1,j} - \Phi_{i-1,j} - \Phi_{i,j+1} - \Phi_{i,j-1} = 0.$$

The five terms in this equation are shown in the figure on the right, where the red colored point is the central position at which the derivatives have been computed. A direct solution of the Laplace equation writes this difference equation in matrix form, with boundary conditions incorporated into the equations, and solves using Gaussian elimination.



Problems for Lecture 62

1. Show that the solution of the discrete Laplace equation at grid point (i, j) on a uniform grid is just the average value of the solution at the neighboring four grid points.

Solutions to the Problems

Lecture 63 | Natural ordering

View this lecture on YouTube

To convert the discrete Laplace equation into matrix form, we will need to place the function values $\Phi_{i,j}$ into a column vector. The commonly used ordering, called natural ordering, starts at the bottom left of the grid and moves left-to right along rows. This results in the column vector

$$\Phi = [\Phi_{1,1}, \Phi_{2,1}, \ldots, \Phi_{n_x,1}, \Phi_{1,2}, \Phi_{2,2}, \ldots, \Phi_{n_x,2} \ldots, \Phi_{n_x,n_y}]^T,$$

which has a total of $n_x n_y$ components. The double index then maps into a single index, with mapping $(i,j) \rightarrow k = i + (j-1)n_x$, and the discrete Laplace equation transforms from

$$4\Phi_{i,j} - \Phi_{i+1,j} - \Phi_{i-1,j} - \Phi_{i,j+1} - \Phi_{i,j-1} = 0$$

to

$$4\Phi_k - \Phi_{k+1} - \Phi_{k-1} - \Phi_{k+n_x} - \Phi_{k-n_x} = 0.$$

This equation is valid when k labels an interior point.

Boundary values for Φ need to satisfy prescribed boundary conditions. Here, we consider Dirichlet boundary conditions, where the values of Φ itself are specified on the boundaries of the domain. The coordinates (i,j) on the boundaries are Bottom (B): $j = 1$; Left (L): $i = 1$; Top (T): $j = n_y$; and Right (R): $i = n_x$; and the ranges of the k indices (using the MATLAB colon operator), are given by

(B) $1 : n_x$; (L) $1 : n_x : 1 + (n_y - 1)n_x$;

(T) $1 + (n_y - 1)n_x : n_x n_y$; (R) $n_x : n_x : n_x n_y$.

In the next lecture, we learn how to construct the matrix equation for Φ.

Problems for Lecture 63

1. What are the k coordinates for the four corners of a rectangular box with n_x and n_y grid points in the x- and y-directions, respectively?

Solutions to the Problems

Lecture 64 | Matrix formulation

View this lecture on YouTube

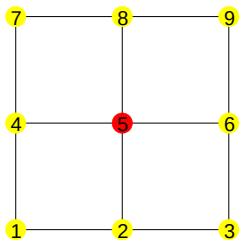
To solve the discrete Laplace equation, given by

$$4\Phi_k - \Phi_{k+1} - \Phi_{k-1} - \Phi_{k+n_x} - \Phi_{k-n_x} = 0,$$

with Φ_k specified on the boundaries, we must first write it as a matrix equation $\mathrm{A}\Phi = \mathrm{b}$, with A an $n_x n_y$-by-$n_x n_y$ square matrix.

When k indexes an internal grid point, the discrete Laplace equation goes into row k of matrix A with a zero in row k of vector b. When k indexes a boundary point, row k of matrix A will be row k of the identity matrix, and the known value of Φ_k will go into row k of b. These boundary rows, then, serve the purpose of assigning the known boundary values to Φ.

For illustration, we construct the matrix equation for a three-by-three grid, shown at right, where the k indexing is labeled. Here we see that there is only one internal grid point (red) and eight boundary points (yellow), though usually there are many more internal points than boundary points. The discrete Laplace equation is only used in row five of the matrix, and the other rows are given by the corresponding rows of the identity matrix. When k indexes a boundary point, we assume that $\Phi_k = b_k$ is known.



The matrix equation is given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ 0 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{pmatrix}$$

In general, the $n_x n_y$-by-$n_x n_y$ matrix A can be constructed using the MATLAB function `spdiags`, suitable for a sparse banded matrix. First A is constructed with five diagonals corresponding to the five terms in the discrete Laplace equation, and then the rows corresponding to boundary points can be replaced by the corresponding rows of the identity matrix. The right-hand side b has zeros in rows corresponding to interior points, and the boundary values of Φ in rows corresponding to boundary points.

We will discuss the MATLAB code that solves the Laplace equation in the next Lecture.

Problems for Lecture 64

1. Construct the matrix equation for the discrete Laplace equation on a four-by-four grid. When k indexes a boundary point, assume that $\Phi_k = b_k$ is known.

2. On a rectangular grid with n_x and n_y grid points, how many interior points are there and how many boundary points? What percentage of grid points are boundary points when $n_x = n_y = 100$, and what percentage when $n_x = n_y = 1000$?

Solutions to the Problems

Lecture 65 | MATLAB solution of the Laplace equation (direct method)

View this lecture on YouTube

We solve the Laplace equation in a rectangle with side lengths `Lx` and `Ly`. The resolution is set by the parameters `Nx` and `Ny` with `nx=Nx+1` and `ny=Ny+1` grid points in each direction. The accuracy of the solution is improved by increasing the values of `Nx` and `Ny`. The beginning of the code defines the rectangle and the grid:

```
Lx=1; Ly=1; %rectangle dimensions
Nx=100; Ny=100; %number of intervals in x,y directions
nx=Nx+1; ny=Ny+1; %number of gridpoints in x,y directions
dx=Lx/Nx; dy=Ly/Ny; %grid length in x,y directions
x=(0:Nx)*dx; y=(0:Ny)*dy; %x,y values on the grid
```

Next, the A matrix is constructed. The k indices of the boundary points are defined in the vector `boundary_index`, the five diagonals are placed in the A matrix using the MATLAB `spdiags` function, and the rows associated with the boundary points are replaced by the corresponding rows of the identity matrix using `speye`:

```
boundary_index=[1:nx, 1:nx:1+(ny-1)*nx, 1+(ny-1)*nx:nx*ny, nx:nx:nx*ny];
diagonals = [4*ones(nx*ny,1), -ones(nx*ny,4)];
A=spdiags(diagonals,[0 -1 1 -nx nx], nx*ny, nx*ny);
I=speye(nx*ny);
A(boundary_index,:)=I(boundary_index,:);
```

We next construct the right-hand side of the matrix equation. We use the boundary conditions $\Phi = 0$ on the bottom, left and right sides of the square, and $\Phi = 4x(1-x)$ at the top. Since $0 \leq x \leq 1$, Φ will equal zero on the corners and one at the midpoint of the top. The simplest and clearest way to construct `b` is to first use (i,j) indexing and then turn `b` into a column vector using the MATLAB `reshape` function (or, more simply, the colon operator):

```
b=zeros(nx,ny); %interior rows are zero
b(:,1)=0; %bottom
b(1,:)=0; %left
b(:,ny)=4*x.*(1-x); %top
b(nx,:)=0; %right
b=reshape(b,nx*ny,1); %make column vector using natural ordering [same as b=b(:)]
```

We solve for Φ and return to (i,j) indexing using `reshape`:

```
Phi=A\b; %solution by Gaussian elimination
Phi=reshape(Phi,nx,ny); %back to (i,j) indexing
```

Finally, we plot the solution. We use the MATLAB `meshgrid` to set up the grid points, and choose the contour levels we want to plot in the vector `v`. The contour plot is made using the MATLAB function `contour`. We use `axis equal` so that the graph has the same aspect ratio as the problem domain.

To understand the required data structures of `Phi`, `meshgrid` and `contour`, suppose that the x-y grid is defined by `x=[x1 x2 x3]` and `y=[y1 y2 y3]`. Then `[X, Y]=meshgrid(x,y)` results in the matrices

$$\texttt{X} = \begin{bmatrix} \texttt{x1} & \texttt{x2} & \texttt{x3} \\ \texttt{x1} & \texttt{x2} & \texttt{x3} \\ \texttt{x1} & \texttt{x2} & \texttt{x3} \end{bmatrix}, \quad \texttt{Y} = \begin{bmatrix} \texttt{y1} & \texttt{y1} & \texttt{y1} \\ \texttt{y2} & \texttt{y2} & \texttt{y2} \\ \texttt{y3} & \texttt{y3} & \texttt{y3} \end{bmatrix},$$

which are used to specify every point on the grid. The (i, j) data structure for Φ, where the first index gives the x-location and the second index gives the y-location, is given by

$$\texttt{Phi} = \begin{bmatrix} \texttt{Phi11} & \texttt{Phi12} & \texttt{Phi13} \\ \texttt{Phi21} & \texttt{Phi22} & \texttt{Phi23} \\ \texttt{Phi31} & \texttt{Phi32} & \texttt{Phi33} \end{bmatrix}.$$

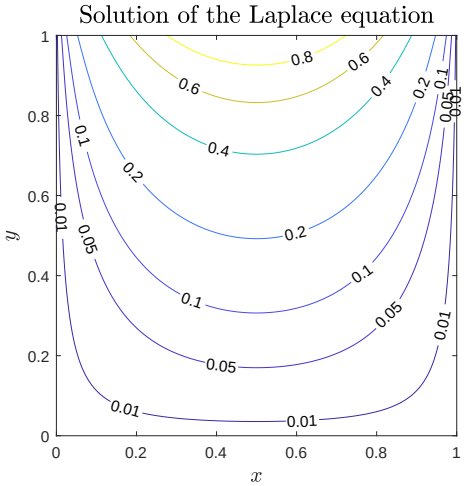
So when calling `contour`, to match the values of `Phi` with the locations in the `X`, `Y` grid, we need to use the transpose, that is,

$$\texttt{Phi'} = \begin{bmatrix} \texttt{Phi11} & \texttt{Phi21} & \texttt{Phi31} \\ \texttt{Phi12} & \texttt{Phi22} & \texttt{Phi32} \\ \texttt{Phi13} & \texttt{Phi23} & \texttt{Phi33} \end{bmatrix}.$$

The code which plots the contours is then written as

```
[X,Y]=meshgrid(x,y);
v=[0.8  0.6  0.4 0.2 0.1 0.05 0.01]; %contour levels
contour(X,Y,Phi',v,'ShowText','on'); %requires transpose
axis equal;
set(gca,'YTick',[0 0.2 0.4 0.6 0.8 1]);
set(gca,'XTick',[0 0.2 0.4 0.6 0.8 1]);
xlabel('$x$','Interpreter','latex','FontSize',14 );
ylabel('$y$','Interpreter','latex','FontSize',14);
title('Solution of the Laplace equation','Interpreter','latex','FontSize',16);
```

The resulting plot looks like this:



Problems for Lecture 65

1. Using the direct method, solve the Laplace equation inside a unit square. Set the boundary conditions to be zero on the left and bottom sides, and to go from zero to one across the top, and from one to zero down the right side. Model these boundary conditions as

$$\Phi = x(2-x) \quad \text{for} \quad y = 1; \qquad \Phi = y(2-y) \quad \text{for} \quad x = 1.$$

Solutions to the Problems

Lecture 66 | Jacobi, Gauss-Seidel and SOR methods

View this lecture on YouTube

Iterative methods are often used for solving systems of equations arising from pdes. Here, we explain the related iterative methods that go by the names of Jacobi, Gauss-Seidel, and Successive Over Relaxation (SOR) methods.

For illustration, we consider again the discrete Laplace equation given by

$$4\Phi_{i,j} - \Phi_{i+1,j} - \Phi_{i-1,j} - \Phi_{i,j+1} - \Phi_{i,j-1} = 0.$$

The Jacobi method simply solves this equation iteratively for $\Phi_{i,j}$. Adding superscripts to denote the iteration step, we iterate the following equation until convergence:

$$\Phi_{i,j}^{(n+1)} = \frac{1}{4}\left(\Phi_{i+1,j}^{(n)} + \Phi_{i-1,j}^{(n)} + \Phi_{i,j+1}^{(n)} + \Phi_{i,j-1}^{(n)}\right).$$

Historically, the FORTRAN language required two different arrays to implement the Jacobi method: the first to store the current solution and the second to store the updated solution. When only one array was used, $\Phi_{i-1,j}$ and $\Phi_{i,j-1}$ were updated before $\Phi_{i,j}$ and the method was instead called Gauss-Seidel. Mathematically, it looked like

$$\Phi_{i,j}^{(n+1)} = \frac{1}{4}\left(\Phi_{i+1,j}^{(n)} + \Phi_{i-1,j}^{(n+1)} + \Phi_{i,j+1}^{(n)} + \Phi_{i,j-1}^{(n+1)}\right).$$

The SOR method first rewrites the Jacobi (or Gauss-Seidel) method by adding and subtracting $\Phi_{i,j}^{(n)}$ on the right-hand side to obtain

$$\Phi_{i,j}^{(n+1)} = \Phi_{i,j}^{(n)} + \frac{1}{4}\left(\Phi_{i+1,j}^{(n)} + \Phi_{i-1,j}^{(n)} + \Phi_{i,j+1}^{(n)} + \Phi_{i,j-1}^{(n)} - 4\Phi_{i,j}^{(n)}\right).$$

In this form, we see that the Jacobi method updates the value of $\Phi_{i,j}$ at each iteration. We can either magnify or diminish this update by multiplying the correction by λ. After a bit of algebra, we find

$$\Phi_{i,j}^{(n+1)} = (1-\lambda)\Phi_{i,j}^{(n)} + \frac{\lambda}{4}\left(\Phi_{i+1,j}^{(n)} + \Phi_{i-1,j}^{(n)} + \Phi_{i,j+1}^{(n)} + \Phi_{i,j-1}^{(n)}\right).$$

When $1 < \lambda < 2$, the method is called *Successive Over Relaxation*, or SOR, and convergence may be accelerated. For some nonlinear equations, however, it may be necessary to choose $0 < \lambda < 1$ for stability reasons, and perhaps a better name for the method would be *Successive Under Relaxation*. When under relaxing, the convergence of the iteration will be slowed, but that is the price that sometimes must be paid.

Problems for Lecture 66

1. The Jacobi, Gauss-Seidel and SOR methods can also be used to solve a system of linear equations. Consider the system of equations given by

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3. \end{aligned}$$

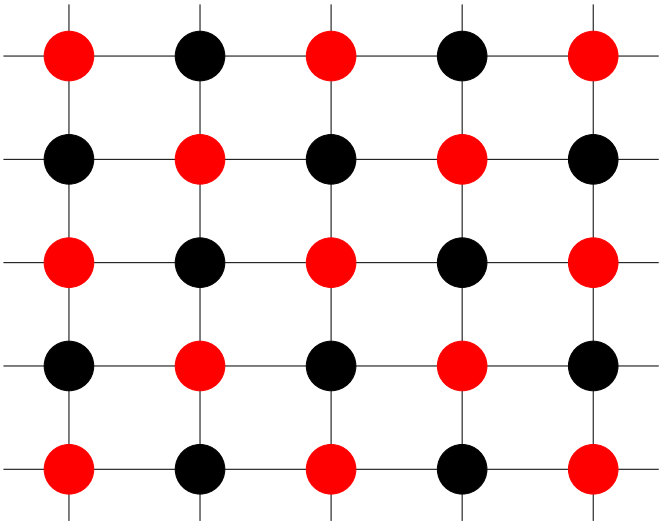
By solving the ith equation for x_i, write down the Jacobi iteration method for this system.

Solutions to the Problems

Lecture 67 | Red-black ordering

View this lecture on YouTube

A final variant of the classical iteration methods uses red-black ordering of the grid points. In this algorithm, the grid is viewed as a checkerboard with alternating red and black grid points, as shown below.



An updating of $\Phi_{i,j}$ is done in two passes: in the first pass, $\Phi_{i,j}$ is updated only at the red grid points; in the second pass, only on the black grid points. Because of the structure of the discrete Laplace equation, the updated values of Φ on the red squares depend only on the values of Φ on the black squares, and the updated values of Φ on the black squares depend only on the values of Φ on the red squares. All updating terms, then, are at the same level of iteration. This could result in faster and more stable convergence, or may be useful when implementing parallel programming algorithms.

Lecture 68 | MATLAB solution of the Laplace equation (iterative method)

View this lecture on YouTube

The code begins in the same way as the direct method:

```
Lx=1; Ly=1; %rectangle dimensions
Nx=100; Ny=100; %number of intervals in x,y directions
nx=Nx+1; ny=Ny+1; %number of gridpoints in x,y directions
dx=Lx/Nx; dy=Ly/Ny; %grid length in x,y directions
x=(0:Nx)*dx; y=(0:Ny)*dy; %x,y values on the grid
```

Next, some additional parameters and index vectors are defined:

```
eps=1.e-6; %convergence criteria for each value of Phi
index_x=2:nx-1;  index_y=2:ny-1; %internal grid points
```

The solution matrix Φ is initialized with zero at all internal grid points, and with the boundary values fixed:

```
Phi=zeros(nx,ny);%matrix with solution and boundary conditions
Phi(:,1)=0; Phi(1,:)=0; Phi(:,ny)=4*x.*(1-x); Phi(nx,:)=0; %boundary grid points BLTR
```

We now iterate the Jacobi method until each internal value of Φ changes less than `eps` with each iteration.

```
Phi_old=Phi;
error=2*eps; ncount=0;
while (error > eps)
   ncount=ncount+1;
   Phi(index_x,index_y)=0.25*(Phi(index_x+1,index_y) ...
    +Phi(index_x-1,index_y)+Phi(index_x,index_y+1)+Phi(index_x,index_y-1));
   error=max(abs(Phi(:)-Phi_old(:)));
   Phi_old=Phi;
end
```

Finally, we plot the solution as we did for the direct method.

```
[X,Y]=meshgrid(x,y);
v=[0.8  0.6  0.4 0.2 0.1 0.05 0.01];
[C,h]=contour(X,Y,Phi',v); clabel(C,h);
axis equal;
set(gca, 'YTick', [0 0.2 0.4 0.6 0.8 1]);
set(gca, 'XTick', [0 0.2 0.4 0.6 0.8 1]);
xlabel('$x$','Interpreter','latex','FontSize',14 );
ylabel('$y$','Interpreter','latex','FontSize',14);
title('Solution of the Laplace equation','Interpreter','latex','FontSize',16);
```

Problems for Lecture 68

1. Using the Jacobi method, solve the Laplace equation inside a unit square. Set the boundary conditions to be zero on the left and bottom sides, and to go from zero to one across the top, and from one to zero down the right side. Model these boundary conditions as

$$\Phi = x(2-x) \quad \text{for} \quad y = 1; \qquad \Phi = y(2-y) \quad \text{for} \quad x = 1.$$

Solutions to the Problems

Lecture 69 | Explicit methods for solving the diffusion equation

View this lecture on YouTube

The initial value problem considered here is the one-dimensional diffusion equation for $u = u(x,t)$, given by

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2},$$

with boundary conditions $u(-L_x,t) = u(L_x,t) = 0$ and initial conditions $u(x,0) = u_0(x)$.

We divide the line into N_x intervals using $n_x = N_x + 1$ grid points. With grid spacing $\Delta x = 2L_x/N_x$, the x coordinates of the grid points are given by

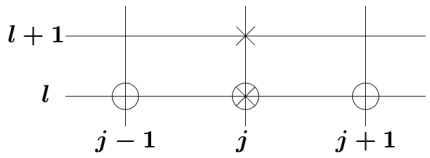
$$x_j = -L_x + (j-1)\Delta x, \qquad \text{for } j = 1,2,\ldots,n_x.$$

Time is also discretized, and with time step Δt, the solution is computed at the times

$$t_l = (l-1)\Delta t, \qquad \text{for } l = 1,2,\ldots.$$

We will denote the value of $u(x_j,t_l)$ by u_j^l. Here, we use the convention that subscripts on u denote the spacial grid point and superscripts on u denote the time step.

The Forward Time Centered Space (FTCS) discretization, with the relevant grid points shown on the right, uses the second-order central difference approximation for the second derivative and the first-order Euler method for the time integration.



The discrete diffusion equation then becomes

$$u_j^{l+1} = u_j^l + \frac{\Delta t D}{(\Delta x)^2}\left(u_{j+1}^l - 2u_j^l + u_{j-1}^l\right), \qquad \text{for } j = 2,3,\ldots,n_x-1 \text{ and } l = 1,2,\ldots.$$

This type of method is called explicit because the solution at the time step $l+1$ can be written explicitly in terms of the solution at the earlier time step l. Higher-order time-stepping methods can also be formulated using Runge-Kutta methods. We shall see in the next lecture, however, that stability issues may be more important than the order of the method.

Problems for Lecture 69

1. Use the second-order Runge-Kutta method known as the modified Euler method to write a two-step process for solving the one-dimensional diffusion equation.

2. Consider the one-dimensional advection equation given by

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}.$$

Using the second-order central difference approximation for the spatial derivative and the first-order Euler method for the time integration, derive the FTCS scheme for the advection equation.

Solutions to the Problems

Lecture 70 | Von Neumann stability analysis

View this lecture on YouTube

We will analyze the stability of the FTCS scheme for the diffusion equation, given by

$$u_j^{l+1} = u_j^l + \frac{\Delta t D}{(\Delta x)^2}\left(u_{j+1}^l - 2u_j^l + u_{j-1}^l\right).$$

We look for solutions of the form

$$u_j^l = \xi^l e^{ikj\Delta x},$$

where $i = \sqrt{-1}$; k is called the wavenumber of the mode and can be any value; ξ (here raised to the lth power) is the unknown parameter that we will try to determine. If we find that $|\xi| > 1$ for any value of the wavenumber k, then we say that the scheme is unstable.

Substitution into the FTCS scheme for the diffusion equation results in

$$\xi^{l+1} e^{ikj\Delta x} = \xi^l e^{ikj\Delta x} + \frac{\Delta t D}{(\Delta x)^2}\left(\xi^l e^{ik(j+1)\Delta x} - 2\xi^l e^{ikj\Delta x} + \xi^l e^{ik(j-1)\Delta x}\right).$$

Dividing by $\xi^l e^{ikj\Delta x}$ leads to

$$\xi = 1 + \frac{\Delta t D}{(\Delta x)^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right).$$

Since $e^{ik\Delta x} + e^{-ik\Delta x} = 2\cos(k\Delta x)$, we obtain

$$\xi = 1 + \frac{2\Delta t D}{(\Delta x)^2}\left(\cos(k\Delta x) - 1\right).$$

The value of $|\xi|$ is largest when $\cos(k\Delta x) = -1$, and the scheme is unstable when

$$1 - \frac{4\Delta t D}{(\Delta x)^2} < -1, \qquad \text{or} \qquad \Delta t > \frac{(\Delta x)^2}{2D}.$$

When a numerical scheme is unstable, the solution blows up in time, which is obviously an unphysical solution of the diffusion equation. Using the FTCS scheme and other explicit methods, stability considerations limit the maximum time step that can be used. We will see that implicit methods will remove this constraint.

Problems for Lecture 70

1. Analyze the stability of the FTCS scheme for the advection equation, given by

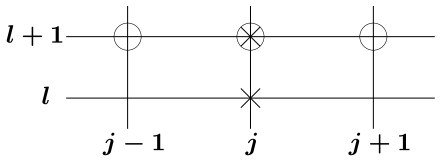
$$u_j^{l+1} = u_j^l - \frac{c\Delta t}{2\Delta x}\left(u_{j+1}^l - u_{j-1}^l\right).$$

Solutions to the Problems

Lecture 71 | Implicit methods for solving the diffusion equation

View this lecture on YouTube

Instead of computing the spatial derivative at the time t_l, we compute it at the forward time t_{l+1} (see the grid points on the right). The discrete diffusion equation is now



$$u_j^{l+1} = u_j^l + \frac{\Delta t D}{(\Delta x)^2}\left(u_{j+1}^{l+1} - 2u_j^{l+1} + u_{j-1}^{l+1}\right).$$

This method is called implicit because finding the solution at the time step $l+1$ requires solving a system of equations.

We can once again perform a stability analysis. With our ansatz

$$u_j^l = \xi^l e^{ikj\Delta x},$$

the implicit discrete diffusion equation results in

$$\xi^{l+1}e^{ikj\Delta x} = \xi^l e^{ikj\Delta x} + \frac{\Delta t D}{(\Delta x)^2}\left(\xi^{l+1}e^{ik(j+1)\Delta x} - 2\xi^{l+1}e^{ikj\Delta x} + \xi^{l+1}e^{ik(j-1)\Delta x}\right).$$

Similar to before, dividing by $\xi^l e^{ikj\Delta x}$ then leads to

$$\xi = 1 + \frac{2\Delta t D\xi}{(\Delta x)^2}\left(\cos\left(k\Delta x\right) - 1\right),$$

where there is now a factor of ξ on the right-hand side. Solving for ξ, we obtain

$$\xi = \frac{1}{1 + \frac{2\Delta t D}{(\Delta x)^2}\left(1 - \cos\left(k\Delta x\right)\right)},$$

which has magnitude always less than or equal to one. The implicit method is unconditionally stable.

Problems for Lecture 71

1. Consider the one-dimensional advection equation given by

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}.$$

a) By computing the spatial derivative at the advanced time step t_{l+1}, derive the implicit discrete advection equation.

b) Analyze its stability.

Solutions to the Problems

Lecture 72 | Crank-Nicolson method for the diffusion equation

View this lecture on YouTube

The most popular discretization of the one-dimensional diffusion equation is called the Crank-Nicolson method and is a stable implicit method that also has the advantage of being second-order in both space and time. To obtain Crank-Nicolson, we average the previous spacial discretizations at the times t_l and t_{l+1} to obtain a central difference approximation for the time derivative of u at the time $t_{l+1/2}$. The discrete diffusion equation becomes

$$u_j^{l+1} = u_j^l + \frac{\Delta t D}{2(\Delta x)^2}\left(\left(u_{j+1}^l - 2u_j^l + u_{j-1}^l\right) + \left(u_{j+1}^{l+1} - 2u_j^{l+1} + u_{j-1}^{l+1}\right)\right).$$

If we define $\alpha = \Delta t D/(\Delta x)^2$ and multiply the equation by two, we can write the Crank-Nicolson method as

$$-\alpha u_{j+1}^{l+1} + 2(1+\alpha)u_j^{l+1} - \alpha u_{j-1}^{l+1} = \alpha u_{j+1}^l + 2(1-\alpha)u_j^l + \alpha u_{j-1}^l,$$

which leads to a tridiagonal matrix equation. For example, with spatial boundary conditions given by $u_1^l = u_{n_x}^l = 0$, and for $n_x = 4$, the matrix equation including the boundary conditions is given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\alpha & 2(1+\alpha) & -\alpha & 0 \\ 0 & -\alpha & 2(1+\alpha) & -\alpha \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1^{l+1} \\ u_2^{l+1} \\ u_3^{l+1} \\ u_4^{l+1} \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha u_1^l + 2(1-\alpha)u_2^l + \alpha u_3^l \\ \alpha u_2^l + 2(1-\alpha)u_3^l + \alpha u_4^l \\ 0 \end{pmatrix}.$$

Although u_1^l and u_4^l are zero on the right-hand side, we include them here for ease of coding. The n_x-by-n_x matrix is independent of time step, and the right-hand side is continuously updated as the solution evolves.

Problems for Lecture 72

1. Consider the one-dimensional advection equation given by

$$\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}.$$

The explicit Lax scheme for the advection equation is given by

$$u_j^{l+1} = \frac{1}{2}(u_{j+1}^l + u_{j-1}^l) - \frac{c\Delta t}{2\Delta x}(u_{j+1}^l - u_{j-1}^l).$$

Analyze its stability and derive the Courant-Friedrichs-Lewy (CFL) stability criterion, which is widely used in fluid turbulence simulations.

Solutions to the Problems

Lecture 73 | MATLAB solution of the diffusion equation

View this lecture on YouTube

We solve the one-dimensional diffusion equation for $|x| \leq L_x$ with u equal zero on the boundaries. We will use the Crank-Nicolson method. The length is set by the parameter `Lx`, and the number of spatial intervals is set by the parameter `Nx`. The number of grid points is `nx=Nx+1`. The spatial grid size is `dx` and the time-step is `dt`. Here we set the time-step at the stability limit for the FTCS scheme.

```
D=1; Lx=1; %Diffusion constant; Domain length
nsteps=10000; %number of time steps
nout=500; %plot every nout time steps
Nx=500; %number of intervals
nx=Nx+1;%number of gridpoints in x direction including boundaries
dx=2*Lx/Nx; %grid size in x
x=-Lx + (0:Nx)*dx; %x values on the grid
dt=(dx)^2/(2*D); %borderline stability of FTCS scheme
```

Next, the `nx`-by-`nx` matrix is constructed using `spdiags`. The top and bottom boundary rows are replaced by the corresponding rows of the identity matrix:

```
alpha=dt*D/dx^2;
diagonals = [2*(1+alpha)*ones(nx,1), -alpha*ones(nx,2)];
A=spdiags(diagonals,[0 -1 1], nx, nx);
I=speye(nx);
A([1 nx],:)=I([1 nx],:); %boundaries
```

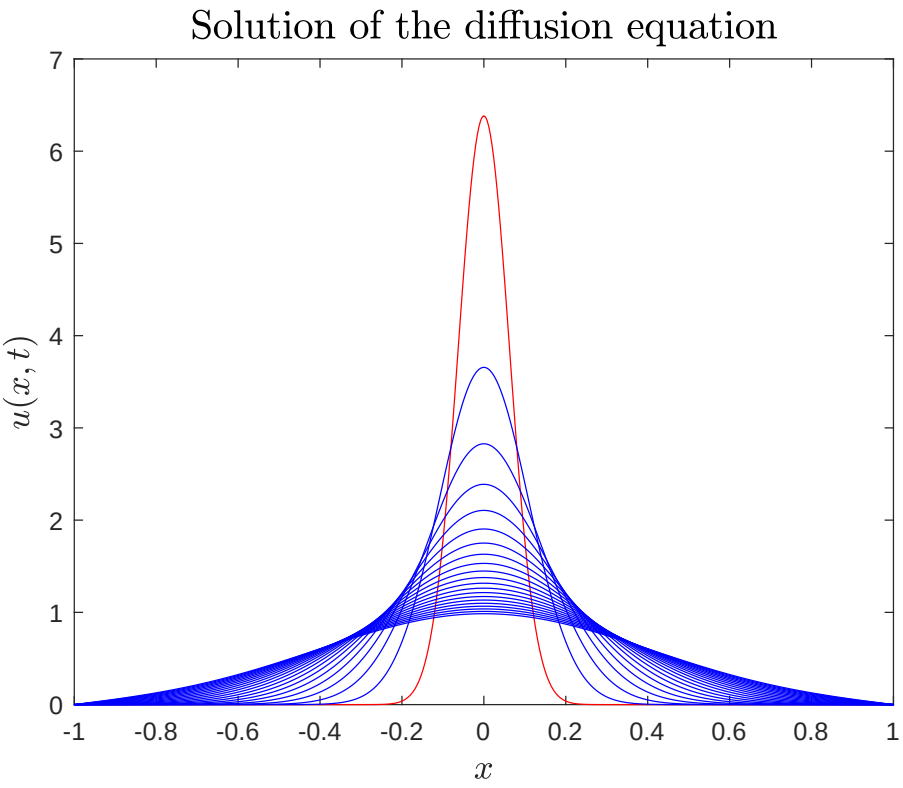
We next initialize the solution and plot the initial conditions. We choose a sharply peaked Gaussian centered at $x = 0$. The values of u on the boundaries will be close enough to zero and will not change.

```
sigma=Lx/16;
u=1/(sigma*sqrt(2*pi))*exp(-0.5*(x/sigma).^2); u=u';
plot(x,u); hold on;
xlabel('$x$','Interpreter','latex','FontSize',14);
ylabel('$u(x, t)$','Interpreter','latex','FontSize',14);
title('Solution of the diffusion equation','Interpreter','latex','FontSize',16);
```

Finally, we evolve the diffusion equation by constructing the right-hand sides and solving using the backslash operator. The solution is plotted every `nout` time steps.

```
for m=1:nsteps
    b=[0; [alpha*u(1:nx-2) + 2*(1-alpha)*u(2:nx-1) + alpha*u(3:nx)]; 0];
    u=A\b;
    if mod(m,nout)==0, plot(x,u), end
end
```

See the next page for the resulting plot.


Solution of the diffusion equation
7
6
5
4
3
2
1
0
$u(x,t)$
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
x

Problems for Lecture 73

1. *a*) Derive a second-order method for the x-derivative at boundary points. When x is a boundary point on the left, use the Taylor series

$$y(x+h) = y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \mathrm{O}(h^3),$$
$$y(x+2h) = y(x) + 2hy'(x) + 2h^2y''(x) + \mathrm{O}(h^3).$$

When x is a boundary point on the right, use the Taylor series

$$y(x-h) = y(x) - hy'(x) + \frac{1}{2}h^2y''(x) + \mathrm{O}(h^3),$$
$$y(x-2h) = y(x) - 2hy'(x) + 2h^2y''(x) + \mathrm{O}(h^3).$$

b) No-flux boundary conditions sets $\partial u/\partial x$ equal to zero on the boundaries. Using the results of Part a), determine boundary conditions for u_1^l and $u_{n_x}^l$.

2. Solve the one-dimensional diffusion equation for $|x| \leq L$ using the Crank-Nicolson method. Assume no flux boundary conditions. Use the results of 1 (b).

Solutions to the Problems

Lecture 74 | Project VI: Two-dimensional diffusion equation

View this lecture on YouTube

Your final project is to solve the two-dimensional diffusion equation. You will combine both the direct method for the Laplace equation and the Crank-Nicolson method.

The two-dimensional diffusion equation for $u = u(x, y, t)$ on a square of side $2L$ is given by

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

where we will assume boundary conditions $u(\pm L, y, t) = u(x, \pm L, t) = 0$.

We form a two-dimensional spatial grid with N intervals and $n = N + 1$ grid points in both the x- and y-directions. With grid spacing $h = 2L/N$, the x and y coordinates of the grid points are given by

$$x_i = -L + (i-1)h, \qquad y_j = -L + (j-1)h,$$

for $i, j = 1, 2, \ldots, n$. With time step given by Δt, we have

$$t_l = (l-1)\Delta t, \qquad \text{for } l = 1, 2, \ldots;$$

and we denote the value of $u(x_i, y_j, t_l)$ by $u^l_{i,j}$.

The Crank-Nicolson method applied to the two-dimensional diffusion equation results in

$$u^{l+1}_{i,j} = u^l_{i,j} + \frac{\Delta t D}{2h^2}\Bigg(\left(u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1} - 4u^l_{i,j}\right) + \left(u^{l+1}_{i+1,j} + u^{l+1}_{i-1,j} + u^{l+1}_{i,j+1} + u^{l+1}_{i,j-1} - 4u^{l+1}_{i,j}\right)\Bigg).$$

We define $\alpha = \Delta t D/h^2$, multiply by two and rearrange terms to obtain

$$2(1+2\alpha)u^{l+1}_{i,j} - \alpha\left(u^{l+1}_{i+1,j} + u^{l+1}_{i-1,j} + u^{l+1}_{i,j+1} + u^{l+1}_{i,j-1}\right) = 2(1-2\alpha)u^l_{i,j} + \alpha\left(u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1}\right).$$

Finally, applying natural ordering with the mapping $(i, j) \to k = i + (j-1)n$, we obtain

the equation

$$2(1+2\alpha)u_k^{l+1} - \alpha\left(u_{k+1}^{l+1} + u_{k-1}^{l+1} + u_{k+n}^{l+1} + u_{k-n}^{l+1}\right) \\ = 2(1-2\alpha)u_k^l + \alpha\left(u_{k+1}^l + u_{k-1}^l + u_{k+n}^l + u_{k-n}^l\right).$$

This is a matrix equation with a time-independent banded matrix with five diagonals. The right-hand side is time dependent. Since you will need to solve a large n^2-by-n^2 matrix equation at each time step, the LU decomposition of the matrix can be performed once and then used to speed-up the code.

Problems for Lecture 74

1. Solve the two-dimensional diffusion equation on a square with u equal to zero on the boundaries.

Solutions to the Problems

Problem solutions and MATLAB learner templates

Solutions to the Problems for Lecture 1

1. A binary number rounded to six places after the binary point will round up if the truncated number is greater than 2^{-7} and will round down if the truncated number is less than 2^{-7}. We have

$$\frac{1}{3} = 0.01010101\overline{01} \approx 0.010101, \quad \frac{1}{5} = 0.00110011\overline{0011} \approx 0.001101, \quad \frac{1}{6} = 0.00101010\overline{10} \approx 0.001011,$$
$$\frac{1}{7} = 0.00100100\overline{1001} \approx 0.001001, \quad \frac{1}{9} = 0.00011100\overline{011100} \approx 0.000111.$$

The numbers $1/3$, $1/7$ and $1/9$ round down and the numbers $1/5$ and $1/6$ round up.

Solutions to the Problems for Lecture 2

1. Using decimal notaton, the number 1 corresponds to $s = 0$, $e = 1023$ and $f = 0$. The binary for 1023 can be found from $1023 = 2^{10} - 1$, so that $e = 100\,0000\,0000 - 000\,0000\,0001 = 011\,1111\,1111$, resulting in the double precision format for 1 to be

$$0011\,1111\,1111\,0000\ldots0000.$$

This number is usually written in hex, which replaces every four bits by a numeral representing the decimal numbers $0-15$. The numerals for $0-9$ are retained, and those for $10-15$ are written using the letters a, b, ..., f. We would therefore write the computer number representing 1 as

$$3ff0\,0000\,0000\,0000.$$

The number $1/2$ corresponds to $s = 0$, $e = 1022$ and $f = 0$. The binary for 1022 is $e = 011\,1111\,1110$, resulting in the double precision format for $1/2$ to be

$$0011\,1111\,1110\,0000\ldots0000,$$

which in hex is given by

$$3fe0\,0000\,0000\,0000.$$

To determine the computer number for $1/3$, we first write

$$\frac{1}{3} = \frac{1}{4}\left(1 + \frac{1}{3}\right).$$

We can observe that $e = 1021$ and $f = 1/3$, which in binary becomes $e = 011\,1111\,1101$ and $f = 0101\ldots0101$, where we have rounded down. We would therefore write the computer number representing $1/3$ as

$$0011\,1111\,1101\,0101\,0101\ldots0101,$$

which in hex is given by

$$3fd5\,5555\,5555\,5555.$$

2. The largest number corresponds to $s = 0$, $e = 2^{11} - 2 - 1023 = 1023$, and $1.f = 1 + (1 - 2^{-52})$. Therefore
`realmax` $= 2^{1023}(2 - 2^{-52}) =$ `1.7977e+308`.

3. The smallest positive normal number corresponds to $s = 0$, $e = 1$, and $1.f = 1 + 0$. Therefore
`realmin` $= 2^{-1022} =$ `2.2251e-308`.

4. The number one is represented as $(-1)^0 \times 2^0 \times 1.0$. The next largest number is $(-1)^0 \times 2^0 \times (1 + 2^{-52}$. Therefore, `eps` $= 2^{-52} =$ `2.2204e-16`.

Solutions to the Problems for Lecture 3

1.

a) `(sqrt(5)-1)/2`
0.6180

b) `(((sqrt(5)+1)/2)^10 - ((sqrt(5)-1)/2 )^10)/sqrt(5)`
55.0000 (Try subtracting 55 from this number.)

c) `2^5/(2^5-1)`
1.0323

d) `(1-1/2^5)^-1`
1.0323

e) `exp(1)^3`
20.0855

f) `log(exp(1)^3)`
3.0000 (Try subtracting 3 from this number.)

g) `sin(pi/6)`
0.5000 (Try subtracting 0.5 from this number.)

h) `cos(pi/6)`
0.8660

i) `sin(pi/6)^2 + cos(pi/6)^2`
1 (Notice the format, and try subtracting 1 from this number.)

Solutions to the Problems for Lecture 4

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
function F = Fibonacci(n)
% Assign Phi, phi, and F below.  Do not change these variable names.
Phi =
phi =
F =
end
```

Here is a test of the finished function:

```
>> Fibonacci(30)
ans =
       832040
>>
```

Solutions to the Problems for Lecture 5

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template. Script was saved as `trig_values.m`.

```
% Assign x below to a row vector.  Do not change this variable name.
x =
fprintf('x:     '); fprintf('%7.4f ', x); fprintf('\n');
fprintf('cos(x):'); fprintf('%7.4f ', cos(x)); fprintf('\n');
fprintf('sin(x):'); fprintf('%7.4f ', sin(x)); fprintf('\n');
```

```
>> trig_values
x:      0.0000  0.5236  0.7854  1.0472  1.5708
cos(x): 1.0000  0.8660  0.7071  0.5000  0.0000
sin(x):  0.0000  0.5000  0.7071  0.8660  1.0000
>>
```

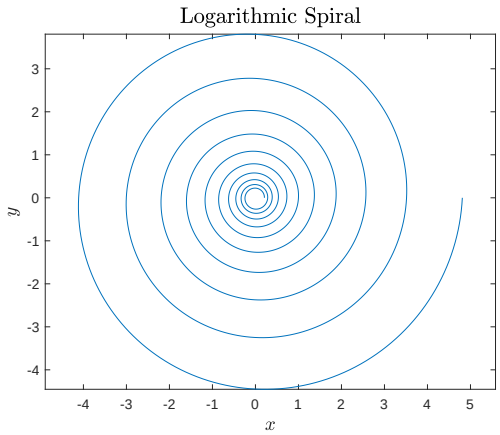
Solutions to the Problems for Lecture 6

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
k = 0.05;
% Assign theta, x, and y below. Do not change these variable names.
theta =
x =
y =
% Graphics
plot(x,y)
axis equal
```

```
xlabel('$x$', 'Interpreter', 'latex', 'FontSize', 14)
ylabel('$y$', 'Interpreter', 'latex', 'FontSize', 14)
title('Logarithmic Spiral','Interpreter','latex','FontSize', 16)
```

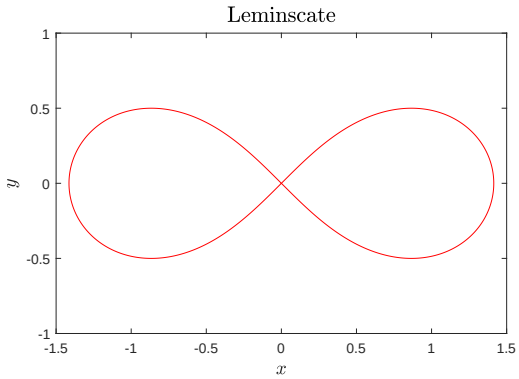
Your plot should look like this:



2. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
% Assign theta, x, and y below. Do not change these variable names.
theta =
x =
y =
% graphics
plot(x,y,'r',-x,y,'r')
axis equal;
axis([-1.5 1.5 -1 1])
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Leminscate', 'Interpreter', 'latex', 'FontSize', 16)
```

Your plot should look like this:



Solutions to the Problems for Lecture 7

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
% Assign the matrix A.  Use the ones.m function and the colon operator.
A =
B = A([1,3],[2,4])
C = A(:,[1,4:6])
D = A([2,3],:)
```

2. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
function  [A, B, C] = banded_matrices(n)
    % Assign A, B and C below.
    A =
    B =
    C =
end
```

Solutions to the Problems for Lecture 8

1.

```
>> 14>15/3
ans =
     1
>> 8/2<5*3+1>9
ans =
     0
>> 8/(2<5)*3+(1>9)
ans =
     24
>> 2+4+3~=60/4-1
ans =
     1
>>
```

2.

```
>> u=[4 -2 1]; v=[0 2 1];
>> u<=v
ans =
     0    1    1
>> u==v
ans =
```

```
     0     0     1
>> u<u+v
ans =
     0     1     1
>> (u<v)+u
ans =
     4    -1     1
>>
```

Solutions to the Problems for Lecture 9

1.

```
function  [p, q] = quadratic_formula_1(a, b, c)
% standard quadratic formula
p=(-b+sqrt(b.^2 - 4.*a.*c))./(2*a);
q=(-b-sqrt(b.^2 - 4.*a.*c))./(2*a);
end
```

```
>> [p, q] = quadratic_formula_1(1, -1, -6)
p =
      3
q =
     -2
>> [p, q] = quadratic_formula_1(1, 1, -6)
p =
      2
q =
     -3
>> [p, q] = quadratic_formula_1(1, -1e12, 1)
p =
   1.0000e+12
q =
     0
>>
```

```
function  [p, q] = quadratic_formula_2(a, b, c)
% better computational quadratic formula
if b<0
    p=(-b+sqrt(b.^2 - 4.*a.*c))./(2*a);
    q=(2*c)./(-b+sqrt(b.^2 - 4.*a.*c));
else
    p=(2*c)./(-b-sqrt(b.^2 - 4.*a.*c));
    q=(-b-sqrt(b.^2 - 4.*a.*c))./(2*a);
end
```

```
>> [p, q] = quadratic_formula_2(1, -1, -6)
p =
      3
```

```
q =
     -2
>> [p, q] = quadratic_formula_2(1, 1, -6)
p =
      2
q =
     -3
>> [p, q] = quadratic_formula_2(1, -1e12, 1)
p =
   1.0000e+12
q =
   1.0000e-12
>>
```

Solutions to the Problems for Lecture 10

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
function F = Fibonacci(n)
% Returns the nth Fibonacci number using recursion relation
n=round(n); % n must be an integer
```

Here are some sample function calls:

```
>> Fibonacci(9)
ans =
     34
>> Fibonacci(-9)
ans =
     34
>> Fibonacci(30)
ans =
       832040
>> Fibonacci(-30)
ans =
     -832040
>>
```

Solutions to the Problems for Lecture 11

1. We solve

$$x = f\big(f(x)\big) = f\big(\mu x(1-x)\big) = r\big(\mu x(1-x)\big)\big(1-\mu x(1-x)\big).$$

Expanding, we obtain the quartic equation

$$\mu^3 x^4 - 2\mu^3 x^3 + \mu^2(1+\mu)x^2 + (1-\mu^2)x = 0.$$

Two solutions corresponding to period-1 cycles are known: $x = 0$ and $x = 1 - 1/\mu$. We factor out these two solutions, the second by using long division, and divide by μ. The result is the quadratic equation

$$\mu^2 x^2 - \mu(\mu+1)x + (\mu+1) = 0.$$

The period-2 cycle, then, corresponds to the two roots of this quadratic equation; that is,

$$x_1 = \frac{1}{2\mu}\left((\mu+1) + \sqrt{(\mu+1)(\mu-3)}\right), \quad x_2 = \frac{1}{2\mu}\left((\mu+1) - \sqrt{(\mu+1)(\mu-3)}\right).$$

These roots are valid solutions for $\mu \geq 3$. Exactly at $\mu = 3$, $x_1 = x_2 = 2/3$, which coincides with the value of the fixed point $1 - 1/\mu = 2/3$. At $\mu = 3$, we will see that the fixed point of the logistic map bifurcates to a period-2 cycle.

Solutions to the Problems for Lecture 12

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
mu_min=2.4; mu_max=4; %range of mu values
n_mu=500; %number of mu pixels
n_x=400; %number of x pixels
mu_edges=linspace(mu_min,mu_max,n_mu+1); %edges of mu pixels
mu=(mu_edges(1:n_mu)+mu_edges(2:n_mu+1))/2; %values of mu on which to perform computation
x_edges=linspace(0,1,n_x+1); %edges of x pixels

n_trans=200000; %transient iterations
n_data=100000;  %number of x values per mu value

x_data=zeros(n_data,n_mu); %x-data used to construct figure

x_0=0.5; %initial condition

% WRITE THE COMPUTATIONAL ENGINE OF THE CODE.
% USE THE ALREADY DEFINED PARAMETERS AND VARIABLES: n_mu, mu, x_0, n_trans, n_data.
% YOUR FINAL RESULT WILL BE THE VARIABLE x_data, and this variable will be assessed.
```

```
%%%%% bin data and plot image %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_histogram=zeros(n_x,n_mu); %binned values of x
for i=1:n_mu
x_histogram(:,i)=histcounts(x_data(:,i),x_edges);
x_histogram(:,i)=255*x_histogram(:,i)/max(x_histogram(:,i));
end
colormap(flipud(gray(256))); brighten(-0.8); cmap=colormap;
im=image([mu_edges(1) mu_edges(end)], [x_edges(1) x_edges(end)], x_histogram);
set(gca,'YDir','normal');
xlabel('$\mu$','Interpreter','latex','FontSize',14);
ylabel('$x\;\;$','Interpreter','latex','FontSize',14);
title('Logistic Map Bifurcation Diagram','Interpreter','latex','FontSize',16)
```

Solutions to the Problems for Lecture 13

1. Now $\sqrt{3}$ is the zero of the function $f(x) = x^2 - 3$, and $f(x_0 = 1) = -2$ and $f(x_1 = 2) = 1$, so that the two initial guesses bracket the root. We iterate the algorithm. We have

$$x_2 = 2 - \frac{2-1}{2} = \frac{3}{2} = 1.5.$$

Now, $f(x_2) = 9/4 - 3 = -3/4 < 0$ so that x_1 and x_2 bracket the root. Therefore,

$$x_3 = \frac{3}{2} - \frac{\frac{3}{2} - 2}{2} = \frac{7}{4} = 1.75.$$

Now, $f(x_3) = 49/16 - 3 = 1/16 > 0$ so that x_2 and x_3 bracket the root. Therefore,

$$x_4 = \frac{7}{4} - \frac{\frac{7}{4} - \frac{3}{2}}{2} = \frac{13}{8} = 1.624,$$

and so on.

Solutions to the Problems for Lecture 14

1. We solve $f(x) = 0$, where $f(x) = x^2 - 3$. We use $f'(x) = 2x$. Therefore, Newton's method iterates

$$x_{n+1} = x_n - \frac{x_n^2 - 3}{2x_n}.$$

Choosing an initial guess $x_0 = 1$, we have

$$x_1 = 1 - \frac{-2}{2} = 2,$$
$$x_2 = 2 - \frac{4-3}{4} = \frac{7}{4} = 1.75,$$
$$x_3 = \frac{7}{4} - \frac{\frac{7^2}{4^2} - 3}{\frac{14}{4}} = \frac{388}{224} = 1.73214,$$

already accurate to four significant digits.

Solutions to the Problems for Lecture 15

1. We solve $f(x) = 0$, where $f(x) = x^2 - 3$. The secant method iterates

$$\begin{aligned} x_{n+1} &= x_n - \frac{(x_n^2 - 3)(x_n - x_{n-1})}{x_n^2 - x_{n-1}^2} \\ &= x_n - \frac{(x_n^2 - 3)}{x_n + x_{n-1}}. \end{aligned}$$

With $x_0 = 1$ and $x_1 = 2$, we have

$$\begin{aligned} x_2 &= 2 - \frac{4-3}{3} = \frac{5}{3} = 1.66666, \\ x_3 &= \frac{5}{3} - \frac{\frac{25}{9} - 3}{\frac{5}{3} + 2} = \frac{57}{33} = 1.72727, \\ x_4 &= \frac{57}{33} - \frac{\left(\frac{57}{33}\right)^2 - 3}{\frac{57}{33} + \frac{5}{3}} = \frac{1067}{616} = 1.73214, \end{aligned}$$

and so on.

Solutions to the Problems for Lecture 16

1. Use $|\epsilon_{n+1}| = 0.5|\epsilon_n|^p$.

	Error		
Iteration #	$p = 1$	$p = 1.6$	$p = 2$
0	1	1	1
1	0.5	0.5	0.5
2	0.25	0.165	0.125
3	0.125	2.80e-02	7.81e-03
4	0.0625	1.64e-03	3.05e-05
5	3.13e-02	1.74e-05	4.66e-10

Solutions to the Problems for Lecture 17

1. We start with

$$x_{n+1} = x_n - \frac{(x_n - x_{n-1})f(x_n)}{f(x_n) - f(x_{n-1})}.$$

a) Let $\epsilon_n = r - x_n$. Then $x_n = r - \epsilon_n$ and $x_{n-1} = r - \epsilon_{n-1}$, so that

$$\begin{aligned} x_n - x_{n-1} &= (r - \epsilon_n) - (r - \epsilon_{n-1}) \\ &= (\epsilon_{n-1} - \epsilon_n). \end{aligned}$$

Subtract both sides of the secant method from r to obtain

$$\epsilon_{n+1} = \epsilon_n + \frac{(\epsilon_{n-1} - \epsilon_n) f(r - \epsilon_n)}{f(r - \epsilon_n) - f(r - \epsilon_{n-1})}.$$

b) We Taylor series f for small ϵ, using $f(r) = 0$, to obtain

$$\begin{aligned} f(r - \epsilon_n) &= -\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \ldots, \\ f(r - \epsilon_{n-1}) &= -\epsilon_{n-1} f'(r) + \frac{1}{2}\epsilon_{n-1}^2 f''(r) + \ldots, \end{aligned}$$

so that

$$\begin{aligned} f(r - \epsilon_n) - f(r - \epsilon_{n-1}) &= (\epsilon_{n-1} - \epsilon_n) f'(r) + \frac{1}{2}(\epsilon_n^2 - \epsilon_{n-1}^2) f''(r) + \ldots \\ &= (\epsilon_{n-1} - \epsilon_n) \left(f'(r) - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n) f''(r) + \ldots \right), \end{aligned}$$

and the secant method becomes

$$\epsilon_{n+1} = \epsilon_n + \frac{-\epsilon_n f'(r) + \frac{1}{2}\epsilon_n^2 f''(r) + \ldots}{f'(r) - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n) f''(r) + \ldots}.$$

c) We rewrite the previous equation to obtain

$$\epsilon_{n+1} == \epsilon_n - \epsilon_n \frac{1 - \frac{1}{2}\epsilon_n \frac{f''(r)}{f'(r)} + \ldots}{1 - \frac{1}{2}(\epsilon_{n-1} + \epsilon_n) \frac{f''(r)}{f'(r)} + \ldots}.$$

We now Taylor series expand the denominator to obtain

$$\begin{aligned} \epsilon_{n+1} &= \epsilon_n - \epsilon_n \left(1 - \frac{1}{2}\epsilon_n \frac{f''(r)}{f'(r)} + \ldots \right) \left(1 + \frac{1}{2}(\epsilon_{n-1} + \epsilon_n) \frac{f''(r)}{f'(r)} + \ldots \right) \\ &= -\frac{1}{2}\frac{f''(r)}{f'(r)} \epsilon_n \epsilon_{n-1} + \ldots, \end{aligned}$$

or to leading order

$$|\epsilon_{n+1}| = \frac{1}{2} \left| \frac{f''(r)}{f'(r)} \right| |\epsilon_n| |\epsilon_{n-1}|. \tag{1}$$

d) To determine the order of convergence, we look for a solution of the form

$$|\epsilon_{n+1}| = k |\epsilon_n|^p, \qquad |\epsilon_n| = k |\epsilon_{n-1}|^p.$$

Substitution into (1) results in

$$k^{p+1} |\epsilon_{n-1}|^{p^2} = \frac{k}{2} \left| \frac{f''(r)}{f'(r)} \right| |\epsilon_{n-1}|^{p+1}.$$

Equating the power of ϵ_{n-1} results in

$$p^2 = p + 1.$$

The rate of convergence of the secant method, given by p, is therefore the positive root of the quadratic equation $p^2 - p - 1 = 0$, or

$$p = \frac{\sqrt{5}+1}{2} \approx 1.618,$$

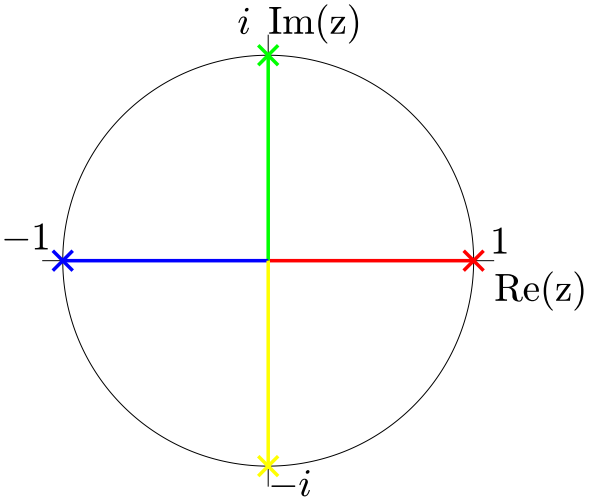
which coincidentally is the famous irrational number that is called the golden ratio, and goes by the symbol Φ.

Solutions to the Problems for Lecture 18

1. We are looking for the four solutions to $z^4 = 1$. Clearly, one and minus one are solutions. Since $i^2 = (-i)^2 = -1$, the four solutions, moving counterclockwise around the unit circle, are evidently

$$r_1 = 1, \qquad r_2 = i, \qquad r_3 = -1, \qquad r_4 = -i.$$

Their location on the unit circle in the complex plane is shown below.

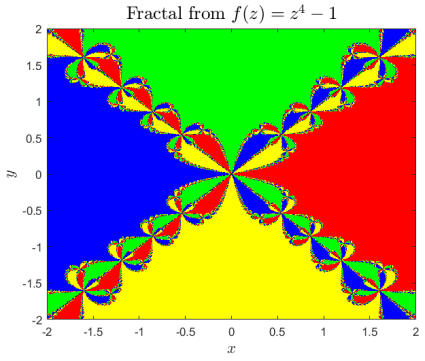


Solutions to the Problems for Lecture 19

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
% code the function and its derivative that will be used in Newton's method
f = @(z) add function here
fp = @(z) add derivative here
% assign the four fourth roots of unity,
% starting at 1 and moving around the unit circle counterclockwise
root1 = 1;
root2 = add second root here
root3 = add third root here
root4 = add fourth root here

nx=1000; ny=1000;
xmin=-2; xmax=2; ymin=-2; ymax=2;

x=linspace(xmin,xmax,nx); y=linspace(ymin,ymax,ny);
[X,Y]=meshgrid(x,y);
Z=X+1i*Y;

nit=50;
for n=1:nit
    Z = Z - f(Z) ./ fp(Z);
end

eps=0.001;
Z1 = abs(Z-root1) < eps; Z2 = abs(Z-root2) < eps;
Z3 = abs(Z-root3) < eps; Z4 = abs(Z-root4) < eps;
Z5 = ~(Z1+Z2+Z3+Z4);

figure;
map = [1 0 0; 0 1 0; 0 0 1; 1 1 0; 0 0 0];
colormap(map); %[red;blue;green;yellow;black]
Z=(Z1+2*Z2+3*Z3+4*Z4+5*Z5);
image([xmin xmax], [ymin ymax], Z); set(gca,'YDir','normal');
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Fractal from $f(z)=z^4-1$', 'Interpreter', 'latex','FontSize', 16)
```

Your plot should look like this:

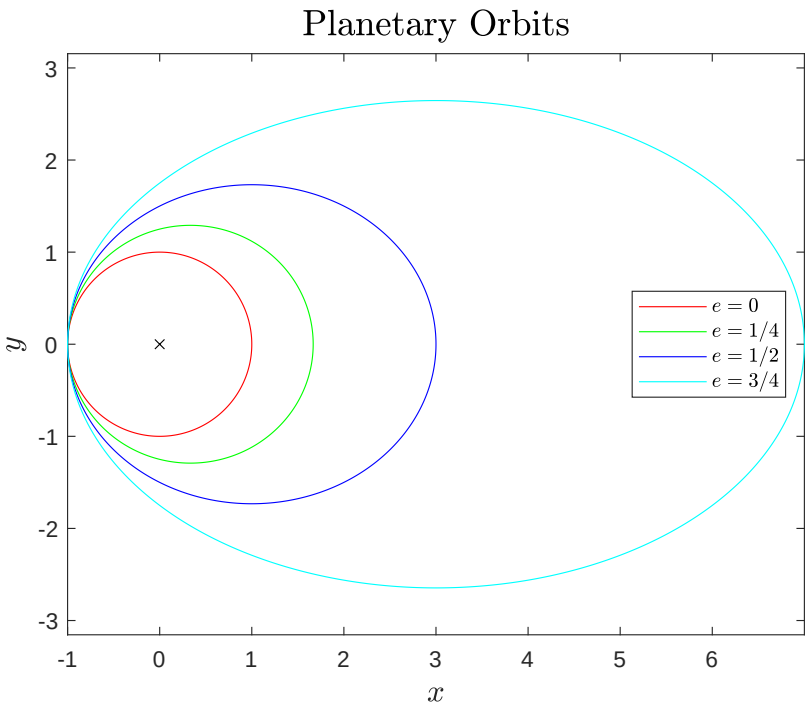


Solutions to the Problems for Lecture 20

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
period=1;  omega=2*pi/period;
e=[0,1/4,1/2,3/4]; color=['r','g','b','c'];
a=1./(1-e); b=sqrt((1+e)./(1-e));
t=linspace(0,period,1000);
x=zeros(length(t),length(e)); y=zeros(length(t),length(e));
for j=1:length(e)
    for i=1:length(t)
        E=fzero(@(E) ... ,0); % add anonymous function for root finding.
                              % Make use of the variables e(j), t(i) and omega.
        x(i,j)= ; % assign x-coordinate.  Make use of the variables a(j), e(j) and E.
        y(i,j)= ; % assign y-coordinate.  Make use of the variables b(j) and E.
    end
end
for j=1:length(e)
    plot(x(:,j),y(:,j),color(j)); axis equal; hold on;
end
plot(0,0,'xk') %mark the origin
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14)
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14)
legend('$e=0$','$e=1/4$','$e=1/2$','$e=3/4$','Interpreter','latex','Location','East')
title('Planetary Orbits', 'Interpreter', 'latex', 'FontSize',16)
```

Your plot should look like this:



Solutions to the Problems for Lecture 21

1.

a) The period-two fixed point equations are given by

$$x_1 = \mu x_0(1 - x_0), \qquad x_0 = \mu x_1(1 - x_1).$$

We can eliminate x_1 to obtain the single equation

$$x_0 = \mu(\mu x_0(1 - x_0))(1 - \mu x_0(1 - x_0)).$$

Substituting $x_0 = 1/2$, we have

$$\frac{1}{2} = \mu(\frac{\mu}{4})(1 - \frac{\mu}{4}).$$

Multiplying by 16 and expanding, we obtain $\mu^3 - 4\mu^2 + 8 = 0$.

b) The long division, complements of the LaTeX Polynom Package, is given by

$$\begin{array}{r|l} & x^2 - 2x - 4 \\ \hline x - 2 & x^3 - 4x^2 + 8 \\ & \underline{-x^3 + 2x^2} \\ & -2x^2 \\ & \underline{2x^2 - 4x} \\ & -4x + 8 \\ & \underline{4x - 8} \\ & 0 \end{array}$$

The roots of

$$\mu^2 - 2\mu - 4 = 0$$

are given by

$$\mu = \frac{2 \pm \sqrt{4 + 16}}{2} = 1 \pm \sqrt{5},$$

and the positive root yields $m_1 = 1 + \sqrt{5}$.

Solutions to the Problems for Lecture 23

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
% Compute the Feigenbaum delta
% Store approximate values in the row vector delta for assessment,
% where length(delta)= num_doublings and delta(2:num_doublings)
% are computed from the algorithm described in Lectures 21-23.
```

```
num_doublings=11; delta=zeros(1,num_doublings); delta(1)=5;
% Write your code here









% Output your results
fprintf('n        delta(n)\n');
for n=1:num_doublings
    fprintf('%2g %18.15f\n',n,delta(n));
end
```

Solutions to the Problems for Lecture 24

1. Using MATLAB as a calculator, we have

```
>> x2=(-4/eps + 1)/(-2/eps - 1)
x2 =
     2
>> x1=(4-2*x2)/eps
x1 =
     0
>> x2=(-4/(2*eps) + 1)/(-2/(2*eps) - 1)
x2 =
     2.0000
>> x1=(4-2*x2)/(2*eps)
x1 =
     3
>>
```

Solutions to the Problems for Lecture 25

1. Using MATLAB as a calculator, we have

```
>> x2=(4-eps)/(2+eps)
x2 =
     2
>> x1=1+x2
x1 =
```

```
     3
>> x2=(4-2*eps)/(2+2*eps)
x2 =
     2.0000
>> x1=1+x2
x1 =
     3.0000
>>
```

Solutions to the Problems for Lecture 26

1. We begin with a row exchange:

$$A = \begin{pmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{pmatrix} \to \begin{pmatrix} 6 & -6 & 7 \\ -3 & 2 & -1 \\ 3 & -4 & 4 \end{pmatrix} = P_{12}A.$$

The first elimination step is

$$P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & -1 & 5/2 \\ 3 & -4 & 4 \end{pmatrix} = M_1 P_{12}A, \qquad \text{where } M_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The second elimination step is

$$M_1 P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & -1 & 5/2 \\ 0 & -1 & 1/2 \end{pmatrix} = M_2 M_1 P_{12}A, \qquad \text{where } M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}.$$

No more row interchanges are needed and the third elimination step is

$$M_2 M_1 P_{12}A \to \begin{pmatrix} 6 & -6 & 7 \\ 0 & -1 & 5/2 \\ 0 & 0 & -2 \end{pmatrix} = M_3 M_2 M_1 P_{12}A = U, \quad \text{where } M_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

Taking the inverse of the matrix multiplying A, we have

$$A = P_{12}(M_3 M_2 M_1)^{-1} U = (P_{12}L)U,$$

where the upper triangular and psychologically lower triangular matrices are given by

$$U = \begin{pmatrix} 6 & -6 & 7 \\ 0 & -1 & 5/2 \\ 0 & 0 & -2 \end{pmatrix}, \qquad (P_{12}L) = P_{12} \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ 1/2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -1/2 & 1 & 0 \\ 1 & 0 & 0 \\ 1/2 & 1 & 1 \end{pmatrix}.$$

Solutions to the Problems for Lecture 27

1. Let T_L be the time to compute recombination with L loci. We have

$$T_L = k3^L,$$

so that

$$T_{L+1} = k3^{L+1} = 3k3^L = 3T_L.$$

If $L = 15$ takes 10 sec, then the time for $L = 16$ is estimated to be 3×10 sec $= 30$ sec.

Solutions to the Problems for Lecture 28

1.

a) The summation denoted as $\sum_{k=1}^{n} 1$ is just the addition of n ones and is equal to n.

b) The simplest way to compute $\sum_{k=1}^{n} k$ is to define this sum to be X, and to write X twice as follows:

$$\begin{aligned} X &= 1+ \quad 2 \quad + \quad 3 \quad +\ldots+n \\ X &= n+(n-1)+(n-2)+\ldots+1. \end{aligned}$$

Adding these two equations yields $2X = n(n+1)$, or $X = n(n+1)/2$.

c) The trick to compute $\sum_{k=1}^{n} k^2$ is to use the expansion $(k-1)^3 = k^3 - 3k^2 + 3k - 1$, and to write $k^3 - (k-1)^3 = 3k^2 - 3k + 1$. Since $\sum_{k=1}^{n} k^3 - (k-1)^3$ is what is called a telescoping series (each new term cancels with part of the preceding term), we have

$$\sum_{k=1}^{n} k^3 - (k-1)^3 = n^3 = 3\sum_{k=1}^{n} k^2 - 3\sum_{k=1}^{n} k + \sum_{k=1}^{n} 1.$$

Therefore,

$$\sum_{k=1}^{n} k^2 = \frac{1}{3}\left(n^3 + \frac{3n(n+1)}{2} - n\right) = \frac{1}{6}n(2n+1)(n+1).$$

Solutions to the Problems for Lecture 29

1. The solution for x_i is found after solving for x_j with $j < i$. The explicit solution for x_i is given by

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\right).$$

The solution for x_i requires i multiplication-additions, and since this must be done for n such $i's$, we have

$$\text{op. counts} = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1).$$

Solutions to the Problems for Lecture 30

1. The two largest eigenvalues are $\lambda_1 = 1$ and $\lambda_2 = 1/2$ with corresponding eigenvectors e_1 and e_2. Suppose $x_0 = c_1 e_1 + c_2 e_2 + \dots$. Then

$$x_p = c_1 \lambda_1^p e_1 + c_2 \lambda_2^p e_2 + \cdots = c_1 e_1 + (1/2)^p c_2 e_2 + \dots .$$

We estimate that to converge to an error of less than 10^{-8}, we need to require

$$(1/2)^p \approx 10^{-8},$$

or $p \approx 8/\log_{10} 2 \approx 27$.

Solutions to the Problems for Lecture 31

1. We apply the power method to

$$\mathrm{A} = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix}.$$

The assumed initial vector and first iteration is given by

$$\mathrm{x}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad \mathrm{x}_1 = \mathrm{A}\mathrm{x}_0 = \begin{pmatrix} -5 \\ 5 \end{pmatrix}.$$

Continuing,

$$\mathrm{x}_2 = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix} \begin{pmatrix} -5 \\ 5 \end{pmatrix} = \begin{pmatrix} 55 \\ -45 \end{pmatrix}; \qquad \mathrm{x}_3 = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix} \begin{pmatrix} 56 \\ 44 \end{pmatrix} = \begin{pmatrix} -545 \\ 455 \end{pmatrix}.$$

Two more iterations give

$$\mathrm{x}_4 = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix} \begin{pmatrix} -545 \\ 455 \end{pmatrix} = \begin{pmatrix} 5{,}455 \\ -4{,}545 \end{pmatrix}; \qquad \mathrm{x}_5 = \begin{pmatrix} -5 & 6 \\ 5 & -4 \end{pmatrix} \begin{pmatrix} 5{,}455 \\ -4{,}545 \end{pmatrix} = \begin{pmatrix} -54{,}545 \\ 45{,}455 \end{pmatrix}.$$

The dominant eigenvalue is approximated from

$$\lambda_1 \approx \frac{\mathrm{x}_4^{\mathrm{T}} \mathrm{x}_5}{\mathrm{x}_4^{\mathrm{T}} \mathrm{x}_4} = \frac{-504{,}135{,}950}{50{,}414{,}050} \approx -9.99991 \approx -10;$$

and the corresponding eigenvector is approximated by x_5. Dividing by the second component,

$$\mathrm{v}_1 = \begin{pmatrix} -54{,}545/45{,}455 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -1.19998 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -6/5 \\ 1 \end{pmatrix}.$$

Solutions to the Problems for Lecture 32

1. Complete your solution on MATLAB. Here is the Learner Template:

```
% Define matrix A
% Define L to be the psychologically lower triangular matrix
% Define U to be the upper triangular matrix
```

2. Complete your solution on MATLAB. Here is the Learner Template:

```
% Define matrix A
% Find eigenvalues, lambda1<lambda2
lambda1= ...
lambda2= ...
% Find eigenvectors associated with lambda1 and lambda2
%     Normalize eigenvectors so that their second component is one
v1= ...
v2= ...
```

Solutions to the Problems for Lecture 33

1. We write down the algorithm to solve

$$f(x,y,z)=0, \qquad g(x,y,z)=0, \qquad h(x,y,z)=0.$$

1. Solve the linear system for Δx_n, Δy_n and Δz_n given by

$$\begin{pmatrix} f_x & f_y & f_z \\ g_x & g_y & g_z \\ h_x & h_y & h_z \end{pmatrix} \begin{pmatrix} \Delta x_n \\ \Delta y_n \\ \Delta z_n \end{pmatrix} = \begin{pmatrix} -f \\ -g \\ -h \end{pmatrix}.$$

2. Advance the iterative solution, using

$$x_{n+1}=x_n+\Delta x_n, \qquad y_{n+1}=y_n+\Delta y_n, \qquad z_{n+1}=z_n+\Delta z_n.$$

Solutions to the Problems for Lecture 34

1. We solve

$$\sigma(y-x)=0, \qquad rx-y-xz=0, \qquad xy-\beta z=0.$$

By inspection, the first fixed-point solution is $(x,y,z)=(0,0,0)$, and another two can be found. The first equation yields $x=y$. Eliminating y from the second equation results in $x(r-1-z)=0$, with solutions $x=0$ or $z=r-1$. The $x=0$ solution corresponds to the

first fixed-point solution. Substituting $y = x$ and $z = r - 1$ into the third equation results in $x^2 = \beta(r-1)$. The square root results in the second and third fixed-point solutions. Our three solutions are

$$(x,y,z) = (0,0,0), \quad (\sqrt{\beta(r-1)}, \sqrt{\beta(r-1)}, r-1), \quad (-\sqrt{\beta(r-1)}, -\sqrt{\beta(r-1)}, r-1).$$

When $r = 28$, $\sigma = 10$ and $\beta = 8/3$, the numerical values are

$$(x,y,z) = (0,0,0), \quad (8.48528, 8.48528, 27), \quad (-8.48528, -8.48528, 27).$$

2. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
r=28; sigma=10; b=8/3;
RelTol=1.e-06; AbsTol=1.e-09;
for nroot=1:3
    if nroot==1, x=1; y=1; z=1; end
    if nroot==2, x=10; y=10; z=10; end
    if nroot==3, x=-10; y=-10; z=-10; end
    error=Inf;
    while error > max(RelTol*max(abs([x,y,z])),AbsTol)
        J=     % DEFINE THE JACOBIAN MATRIX
        rhs =  % DEFINE THE RIGHT-HAND SIDE
        delta_xyz=J\rhs;
        x = x + delta_xyz(1);
        y = y + delta_xyz(2);
        z = z + delta_xyz(3);
        error=max(abs(delta_xyz));
    end
    xroot(nroot)=x; yroot(nroot)=y; zroot(nroot)=z;
end
roots=[xroot;yroot;zroot];
fprintf('steady-state solution:\n')
fprintf('(x, y, z) = (%2.0f,%2.0f,%2.0f)  \n', roots(:,1));
fprintf('(x, y, z) = (%7.5f,%7.5f,%3.0f)  \n', roots(:,2));
fprintf('(x, y, z) = (%7.5f,%7.5f,%3.0f)  \n', roots(:,3));
```

Solutions to the Problems for Lecture 35

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

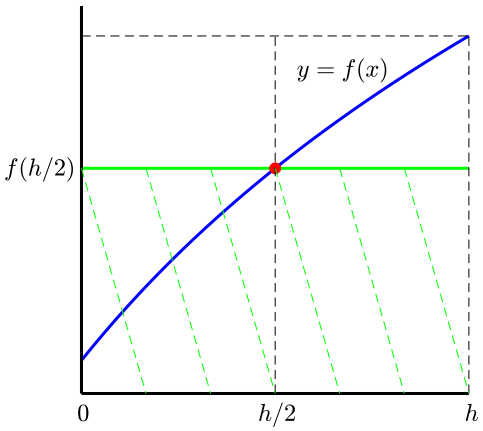
```
r=28; sigma=10; beta=8/3;
x1=0; y1=0; z1=0;
x2=sqrt(beta*(r-1)); y2=sqrt(beta*(r-1)); z2=r-1;
x3=-sqrt(beta*(r-1)); y3=-sqrt(beta*(r-1)); z3=r-1;

nx=500; ny=500;
xmin=-30; xmax=30; ymin=-30; ymax=30;
```

```
x_grid=linspace(xmin,xmax,nx); y_grid=linspace(ymin,ymax,ny);
[X,Y]=meshgrid(x_grid,y_grid);

% Write Newton's method using every gridpoint as the initial condition
% Perform enough iterations at each gridpoint to converge to the correct root
% Save the x-values of the converged roots in the matrix X
% To pass the assessment, every pixel in the figure must be correctly colored

%!!!!!!!!! Set initial value z=-10 for all values (x,y) on the grid !!!!!!!!!!
```

```
eps=0.001;
X1 = abs(X-x1) < eps; X2 = abs(X-x2) < eps; X3 = abs(X-x3) < eps;
X4 = ~(X1+X2+X3);
figure;
map = [1 0 0; 0 1 0; 0 0 1; 0 0 0]; colormap(map); %[red;green;blue;black]
X=(X1+2*X2+3*X3+4*X4);
image([xmin xmax], [ymin ymax], X); set(gca,'YDir','normal');
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Fractal from the Lorenz Equations', 'Interpreter', 'latex','FontSize', 16)
```

Solutions to the Problems for Lecture 36

1. The midpoint rule approximates the area under the curve of $y = f(x)$ from zero to h by the area of a rectangle with base h and height $f(h/2)$, as illustrated in the graph below.



2. We have

$$\begin{aligned}\int_0^h f(x)\,dx &= \int_0^h (a+bx+cx^2)\,dx \\ &= \left(ax+\frac{bx^2}{2}+\frac{cx^3}{3}\right)\Big|_0^h \\ &= ah+\frac{bh^2}{2}+\frac{ch^3}{3} \\ &= h\left(a+\frac{bh}{2}+\frac{ch^2}{3}\right).\end{aligned}$$

Now,

$$f(h/2) = a+\frac{bh}{2}+\frac{ch^2}{4}, \qquad f''(h/2) = 2c.$$

Therefore,

$$\begin{aligned}\int_0^h f(x)\,dx &= h\left(a+\frac{bh}{2}+\frac{ch^2}{3}\right) \\ &= h\left(a+\frac{bh}{2}+\frac{ch^2}{4}\right)+h\left(\frac{ch^2}{3}-\frac{ch^2}{4}\right) \\ &= h\left(a+\frac{bh}{2}+\frac{ch^2}{4}\right)+h\left(\frac{ch^2}{12}\right) \\ &= hf(h/2)+\frac{h^3}{24}f''(h/2).\end{aligned}$$

Solutions to the Problems for Lecture 37

1. We approximate

$$f(x) \approx f(0)+\frac{f(h)-f(0)}{h}x.$$

Then

$$\begin{aligned}\int_0^h f(x)\,dx &= \int_0^h \left(f(0)+\frac{f(h)-f(0)}{h}x\right)\,dx \\ &= \left(f(0)x+\frac{f(h)-f(0)}{2h}x^2\right)\Big|_0^h \\ &= f(0)h+\frac{1}{2}\left(f(h)-f(0)\right)h \\ &= \frac{h}{2}\left(f(0)+f(h)\right).\end{aligned}$$

Solutions to the Problems for Lecture 38

1.

a) Let $g(x) = a + bx + cx^2$. The equations that we want to solve for a, b and c are given by

$$\begin{aligned} g(0) &= a = f(0), \\ g(h) &= a + hb + h^2c = f(h), \\ g(2h) &= a + 2hb + 4h^2c = f(2h), \end{aligned}$$

or in matrix form

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & h & h^2 \\ 1 & 2h & 4h^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} f(0) \\ f(h) \\ f(2h) \end{pmatrix}.$$

We form the augmented matrix and solve by Gaussian elimination:

$$\begin{pmatrix} 1 & 0 & 0 & f(0) \\ 1 & h & h^2 & f(h) \\ 1 & 2h & 4h^2 & f(2h) \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 & f(0) \\ 0 & h & h^2 & f(h) - f(0) \\ 0 & 2h & 4h^2 & f(2h) - f(0) \end{pmatrix}$$

$$\to \begin{pmatrix} 1 & 0 & 0 & f(0) \\ 0 & h & h^2 & f(h) - f(0) \\ 0 & 0 & 2h^2 & f(2h) - 2f(h) + f(0) \end{pmatrix}.$$

Backsubstitution gives

$$\begin{aligned} c &= \frac{1}{2h^2}\big(f(0) - 2f(h) + f(2h)\big), \\ b &= \frac{1}{h}\left(f(h) - f(0) - \frac{1}{2}\big(f(2h) - 2f(h) + f(0)\big)\right) = \frac{1}{2h}\big(-3f(0) + 4f(h) - f(2h)\big), \\ a &= f(0). \end{aligned}$$

b) We approximate $f(x)$ by the quadratic polynomial $g(x)$, that is,

$$f(x) \approx f(0) + \frac{1}{2h}\big(-3f(0) + 4f(h) - f(2h)\big)x + \frac{1}{2h^2}\big(f(0) - 2f(h) + f(2h)\big)x^2.$$

Then

$$\begin{aligned}\int_0^{2h} f(x)\,dx &\approx \int_0^{2h}\Big(f(0)+\frac{1}{2h}\big(-3f(0)+4f(h)-f(2h)\big)x \\ &\qquad +\frac{1}{2h^2}\big(f(0)-2f(h)+f(2h)\big)x^2\Big)dx \\ &=\left(f(0)x+\frac{1}{4h}\big(-3f(0)+4f(h)-f(2h)\big)x^2+\frac{1}{6h^2}\big(f(0)-2f(h)+f(2h)\big)x^3\right)\Big|_0^{2h} \\ &=2hf(0)+h\big(-3f(0)+4f(h)-f(2h)\big)+\frac{4h}{3}\big(f(0)-2f(h)+f(2h)\big) \\ &=\frac{h}{3}\left(f(0)+4f(h)+f(2h)\right).\end{aligned}$$

Solutions to the Problems for Lecture 39

1. We have

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{3h}{8}\left(f_0+3f_1+3f_2+f_3\right)+\frac{3h}{8}\left(f_3+3f_4+3f_5+f_6\right)+\ldots \\ &\qquad +\frac{3h}{8}\left(f_{n-3}+3f_{n-2}+3f_{n-1}+f_n\right) \\ &=\frac{3h}{8}\left(f_0+3f_1+3f_2+2f_3+3f_4+3f_5+2f_6+\cdots+3f_{n-2}+3f_{n-1}+f_n\right),\end{aligned}$$

where the first and last terms have a multiple of one, the terms with indices that have a remainder of one or two when divided by three have a multiple of three, the terms with indices that are a multiple of three have an index of two, and the entire sum is multiplied by $3h/8$.

Solutions to the Problems for Lecture 40

1. The three-point Legendre-Gauss quadrature rule is written as

$$\int_{-1}^{1} f(x)\,dx = w_1 f(x_1)+w_2 f(x_2)+w_3 f(x_3).$$

We will assume that $x_1=-x_3$, $w_1=w_3$ and $x_2=0$. Substituting for the function the basis polynomials $f(x)=1$, x, x^2, x^3, x^4 and x^5, we obtain the six equations given by

$$\begin{aligned}&2=w_1+w_2+w_3,\quad 0=w_1x_1+w_2x_2+w_3x_3,\quad 2/3=w_1x_1^2+w_2x_2^2+w_3x_3^2,\\ &0=w_1x_1^3+w_2x_2^3+w_3x_3^3,\quad 2/5=w_1x_1^4+w_2x_2^4+w_3x_3^4,\quad 0=w_1x_1^5+w_2x_2^5+w_3x_3^5.\end{aligned}$$

Using the symmetry conditions, the third and fifth equations result in

$$2w_1x_1^2=2/3,\qquad 2w_1x_1^4=2/5.$$

Division of the second of these two equations by the first yields

$$x_1^2 = 3/5, \quad \text{or} \quad x_1 = -\sqrt{3/5},$$

and the third of the original equations then yields $w_1 = 5/9$. The first equation yields $w_2 = 8/9$. To summarize, we have found

$$w_1 = 5/9, \quad w_2 = 8/9, \quad w_3 = 5/9; \qquad x_1 = -\sqrt{3/5}, \quad x_2 = 0, \quad x_3 = \sqrt{3/5}.$$

Solutions to the Problems for Lecture 41

1. For $\int_0^h f(x)\,dx$ with $f(x) = x^3$, we have

$$S_1 = \frac{h}{2}\left(f(0) + f(h)\right) = \frac{h^4}{2}, \quad S_2 = \frac{h}{4}\left(f(0) + 2f(h/2) + f(h)\right) = \frac{5h^4}{16}.$$

Since $I = h^4/4$, we have

$$E_1 = I - S_1 = -\frac{h^4}{4}, \quad E_2 = I - S_2 = -\frac{h^4}{16},$$

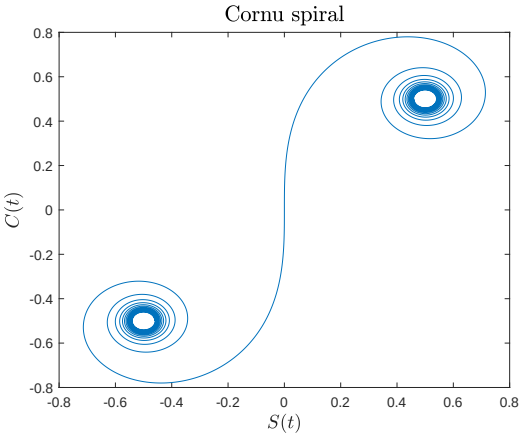
and $E_1 = 4E_2$.

Solutions to the Problems for Lecture 42

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
c = @(x) ... % assign the integrand for C(t)
s = @(x) ... % assign the integrand for S(t)

tmin=-8; tmax=8; nt=2000;
t=linspace(tmin,tmax,nt);
C=zeros(nt,1); S=zeros(nt,1);
for i=1:nt
    C(i)=integral(...); % compute C(i) using integral.m and the integrand c(x) defined on t
    S(i)=integral(...); % compute S(i) using integral.m and the integrand s(x) defined on t
end
plot(S,C)
xlabel('$S(t)$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$C(t)$', 'Interpreter', 'latex', 'FontSize',14);
title('Cornu spiral', 'Interpreter', 'latex', 'FontSize',16);
```

Your plot should look like this:



Solutions to the Problems for Lecture 43

1.

a) We interpolate the points $(0,0)$, $(1,1)$ and $(2,1)$. Let $y = ax^2 + bx + c$. Our equations for a, b and c are

$$c = 0, \qquad a + b + c = 1, \quad 4a + 2b + c = 1.$$

Solving, we find $a = -1/2$, $b = 3/2$ and $c = 0$. Our interpolating quadratic polynomial is

$$g(x) = -\frac{1}{2}x^2 + \frac{3}{2}x,$$

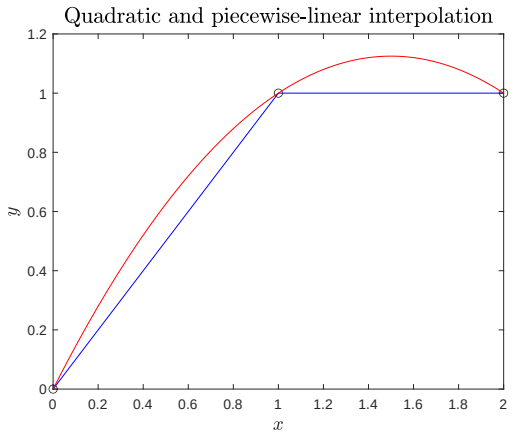
and we find $g(1/2) = 5/8$ and $g(3/2) = 9/8$.

b) The piecewise linear polynomials are

$$g_0(x) = x, \qquad g_1(x) = 1,$$

and we find $g(1/2) = 1/2$ and $g(3/2) = 1$.

c) The points and the two interpolating functions are plotted below.



Solutions to the Problems for Lecture 44

1. We use

$$g_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i, \qquad \text{for } i = 0,\ldots,n-1 \text{ and } x_i \le x \le x_{i+1},$$

and the additional constraints given by $g_0'(x_0) = y_0'$ and $g_{n-1}'(x_n) = y_n'$. The two constraints yield the two additional equations given by

$$\begin{aligned} y_0' &= c_0, \\ y_n' &= 3a_{n-1}h_{n-1}^2 + 2b_{n-1}h_{n-1} + c_{n-1}. \end{aligned}$$

Solutions to the Problems for Lecture 45

1. The points to interpolate are $(0,0)$, $(1,1)$, $(2,1)$ and $(3,2)$. With $h_i = x_{i+1} - x_i$ and $\eta_i = y_{i+1} - y_i$, we have

$$h_i = 1, \qquad \eta_0 = 1, \quad \eta_1 = 0, \quad \eta_2 = 1.$$

The not-a-knot condition yields the two equations

$$b_0 - 2b_1 + b_2 = 0, \qquad b_1 - 2b_2 + b_3 = 0.$$

The full matrix equation for the b-coefficients is then calculated to be

$$\begin{pmatrix} 1 & -2 & 1 & 0 \\ 1/3 & 4/3 & 1/3 & 0 \\ 0 & 1/3 & 4/3 & 1/3 \\ 0 & 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \\ 1 \\ 0 \end{pmatrix}.$$

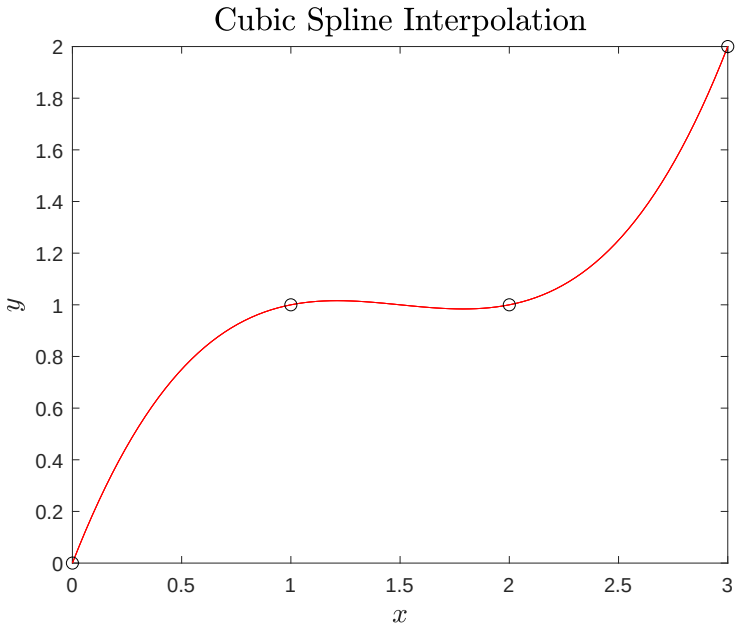
We can use MATLAB as a calculator to solve for b, and then find a, c and d:

$$b = \begin{pmatrix} -3/2 \\ -1/2 \\ 1/2 \\ 3/2 \end{pmatrix}, \quad a = \frac{1}{3}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad c = \begin{pmatrix} 13/6 \\ 1/6 \\ 1/6 \end{pmatrix}, \quad d = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}.$$

The three interpolating cubic polynomials are given by

$$\begin{aligned} g_0(x) &= \frac{1}{3}x^3 - \frac{3}{2}x^2 + \frac{13}{6}x, \\ g_1(x) &= \frac{1}{3}(x-1)^3 - \frac{1}{2}(x-1)^2 + \frac{1}{6}(x-1) + 1, \\ g_2(x) &= \frac{1}{3}(x-2)^3 + \frac{1}{2}(x-2)^2 + \frac{1}{6}(x-2) + 1. \end{aligned}$$

If fact, because of the not-a-knot condition, $g_0 = g_1 = g_2$. The points and the cubic spline interpolant are plotted below.

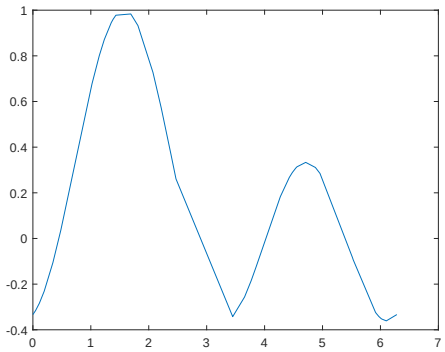


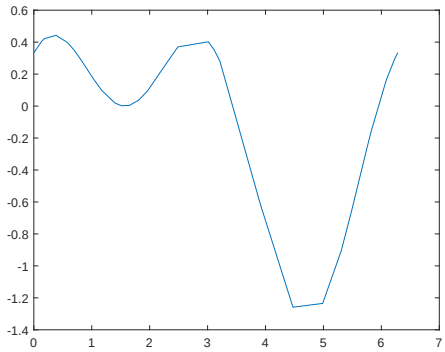
Solutions to the Problems for Lecture 46

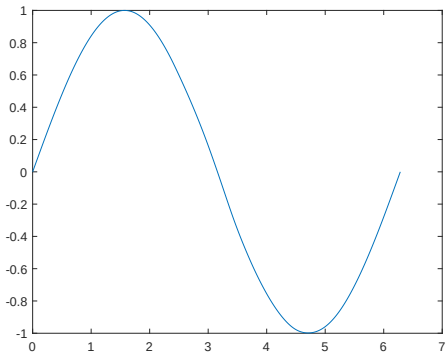
1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
load data1; figure(1); plot(x1,y1);
load data2; figure(2); plot(x2,y2);
xx=linspace(0,2*pi,1000);
yy1=interp1(...); %interpolate y1 to the xx grid using 'spline'
yy2=interp1(...); %interpolate y2 to the xx grid using 'spline'
yyadd=yy1+yy2;
figure(3); plot(xx,yyadd);
```

Here are the resulting plots.







Solutions to the Problems for Lecture 47

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
num_roots=5; num_functions=6;
%initial guess for roots (from Wolfram MathWorld)
zeros_guess=[2.4,3.8,5.1,6,7.5,8.7;...
   5.5,7,8.4,9.7,11,12;...
   8.6 10,11.6,13,14,16;...
   11.8,13,15,16,18,19;...
   15,16.4,18,19.4,21,22];
%Compute first five roots of first six Bessel functions
%Put in variable bzeros with size(bzeros) = [5, 6]



%print table
fprintf('k     J0(x)     J1(x)     J2(x)     J3(x)     J4(x)     J5(x)\n')
for k=1:num_roots
    fprintf('%i',k)
    for n=0:num_functions-1
        fprintf('%10.4f',bzeros(k,n+1));
    end
    fprintf('\n');
end
```

Solutions to the Problems for Lecture 48

1. Let $\dot{x} = b$ and $x(0) = x_0$. The Euler method applied to the first time step yields

$$x_1 = x_0 + b\Delta t,$$

where $x_1 = x(\Delta t)$. The next time step yields

$$x_2 = x_1 + b\Delta t = x_0 + b(2\Delta t),$$

where $x_2 = x(2\Delta t)$. Continuing to march the solution forward, we obtain

$$x_n = x_0 + b(n\Delta t),$$

which is equivalent to

$$x(t) = x_0 + bt,$$

where $t = n\Delta t$.

Solutions to the Problems for Lecture 49

1. Let $\dot{x} = bt$ and $x(0) = x_0$. The Modified Euler method applied to the first time step yields

$$k_1 = 0, \qquad k_2 = b(\Delta t)^2, \qquad x_1 = x_0 + \frac{1}{2}b(\Delta t)^2,$$

where $x_1 = x(\Delta t)$. The next time step yields

$$k_1 = b(\Delta t)^2, \qquad k_2 = 2b(\Delta t)^2, \qquad x_2 = x_1 + \frac{3}{2}b(\Delta t)^2 = x_0 + 2b(\Delta t)^2 = x_0 + \frac{1}{2}b(2\Delta t)^2.$$

Continuing to march the solution forward, we obtain

$$x_n = x_0 + \frac{1}{2}b(n\Delta t)^2,$$

which is equivalent to

$$x(t) = x_0 + \frac{1}{2}bt^2,$$

where $t = n\Delta t$.

Solutions to the Problems for Lecture 51

1. Ralston's method is given by

$$k_1 = \Delta t f(t_n, x_n), \qquad k_2 = \Delta t f\left(t_n + \frac{3}{4}\Delta t, x_n + \frac{3}{4}k_1\right),$$
$$x_{n+1} = x_n + \frac{1}{3}k_1 + \frac{2}{3}k_2.$$

2. Consider the ode given by $dy/dx = f(x)$, with $y(0)$ as the initial value. Separating variables, and integrating from $x = 0$ to h yields

$$\int_{y(0)}^{y(h)} dy = \int_0^h f(x)\,dx, \qquad \text{or} \qquad \int_0^h f(x)\,dx = y(h) - y(0).$$

We now use our second-order Runge Kutta methods to construct elementary quadrature formulas.

(a) Midpoint rule

$$k_1 = hf(0), \qquad k_2 = hf(h/2), \qquad y(h) = y(0) + hf(h/2).$$

The elementary quadrature formula (midpoint rule) is therefore given by

$$\int_0^h f(x)\,dx = hf(h/2).$$

(b) Modified Euler method

$$k_1 = hf(0), \qquad k_2 = hf(h), \qquad y(h) = y(0) + \frac{1}{2}h\left(f(0) + f(h)\right).$$

The elementary quadrature formula (trapezoidal rule) is therefore given by

$$\int_0^h f(x)\,dx = \frac{1}{2}h\left(f(0) + f(h)\right).$$

Solutions to the Problems for Lecture 52

1. Consider the ode given by $dy/dx = f(x)$, with $y(0)$ as the initial value. Separating variables, and integrating from $x = 0$ to $2h$ yields

$$\int_{y(0)}^{y(2h)} dy = \int_0^{2h} f(x)\,dx, \qquad \text{or} \qquad \int_0^{2h} f(x)\,dx = y(2h) - y(0).$$

We now use the fourth-order Runge Kutta methods to construct an elementary quadrature formula. We have

$$k_1 = 2hf(0), \qquad k_2 = 2hf(h), \qquad k_3 = 2hf(h), \qquad k_4 = 2hf(2h),$$

so that

$$y(2h) = y(0) + \frac{h}{3}\left(f(0) + 4f(h) + f(2h)\right).$$

The quadrature rule is therefore

$$\int_0^{2h} f(x)\,dx = \frac{h}{3}\left(f(0) + 4f(h) + f(2h)\right),$$

which is Simpson's rule.

Solutions to the Problems for Lecture 53

1.

$$k_1 = \Delta t f(t_n, x_n, y_n, z_n), \qquad l_1 = \Delta t g(t_n, x_n, y_n, z_n), \qquad m_1 = \Delta t h(t_n, x_n, y_n, z_n);$$

$$k_2 = \Delta t f(t_n + \Delta t, x_n + k_1, y_n + l_1, z_n + m_1), \quad l_2 = \Delta t g(t_n + \Delta t, x_n + k_1, y_n + l_1, z_n + m_1),$$
$$m_2 = \Delta t h(t_n + \Delta t, x_n + k_1, y_n + l_1, z_n + m_1);$$

$$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2), \qquad y_{n+1} = y_n + \frac{1}{2}(l_1 + l_2), \qquad z_{n+1} = z_n + \frac{1}{2}(m_1 + m_2).$$

Solutions to the Problems for Lecture 54

1. Since the requested error is less than the estimated error, the current time step is rejected and redone using a smaller time step. The new time step is predicted to be

$$\Delta\tau = \Delta t \left(\frac{\varepsilon}{e}\right)^{1/5} = 0.01 \times (1/1.1)^{1/5} = 0.0098112.$$

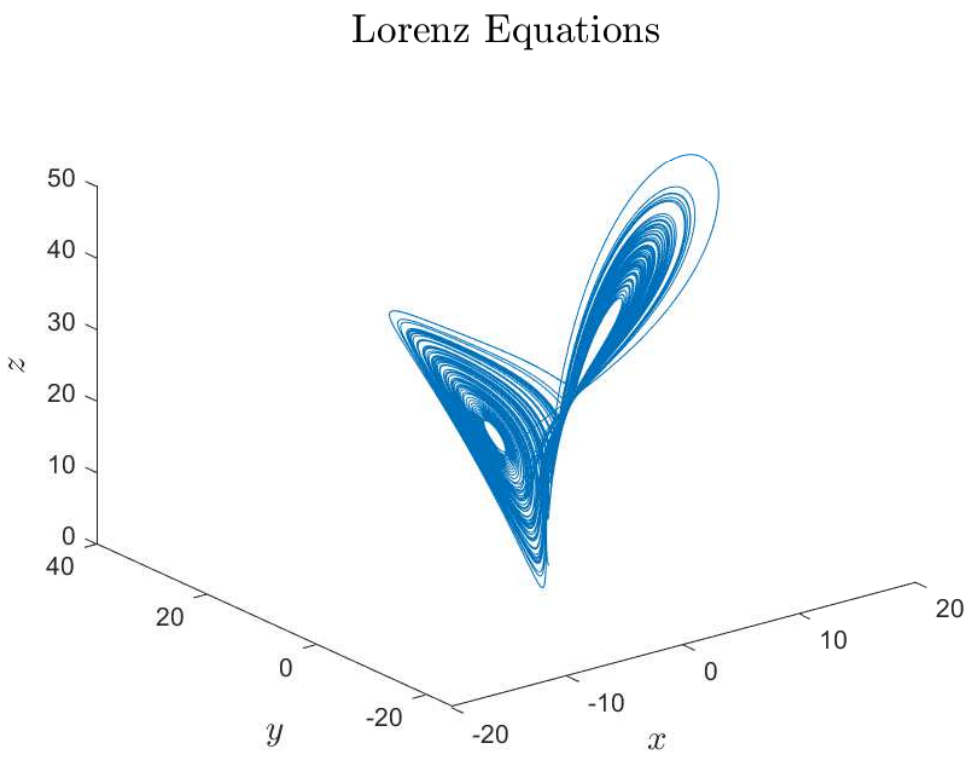
For a margin of safety, one multiplies by the safety factor of 0.9 to obtain $\Delta\tau = 0.0088301$.

Solutions to the Problems for Lecture 56

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
sigma=10; beta=8/3; r=28;
x0=1; y0=1; z0=1; tspan=[0 100];
ntrans=20;
options = odeset('RelTol',1.e-6);
[t,xyz]=ode45(@(t, xyz) lorenz_eqs(PROVIDE
                    FOUR ARGUMENTS HERE), PROVIDE TWO ARGUMENTS HERE, options);
x=xyz(ntrans:end,1); y=xyz(ntrans:end,2); z=xyz(ntrans:end,3);
plot3(x,y,z);
xlabel('$x$','Interpreter','latex','FontSize',14 );
ylabel('$y$','Interpreter','latex','FontSize',14 );
zlabel('$z$','Interpreter','latex','FontSize',14 );
title('Lorenz Equations','Interpreter','latex','FontSize',16);

function dxyzdt = lorenz_eqs(xyz,sigma,beta,r)
x=xyz(1); y=xyz(2); z=xyz(3);
dxyzdt=
 [WRITE THE DIFFERENTIAL EQUATION HERE AS A COLUMN VECTOR. USE VARIABLES x, y AND z.];
end
```

Your plot should look like this:



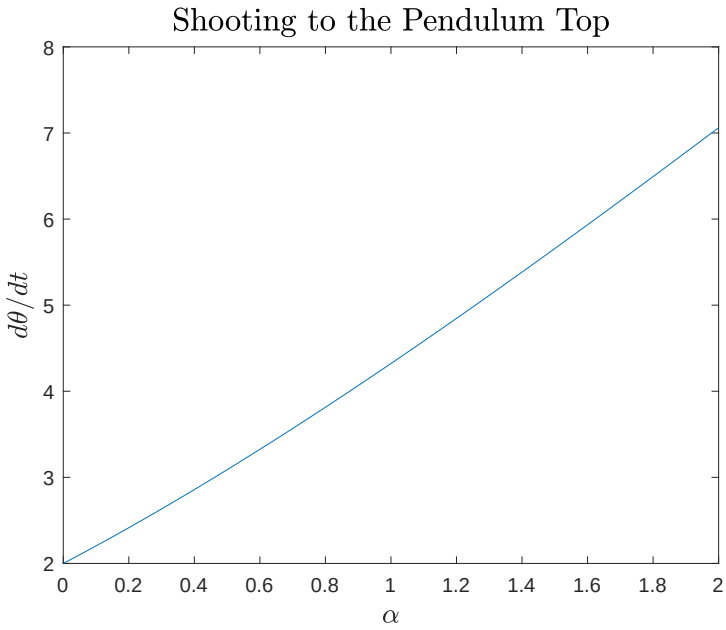
When viewed in the MATLAB environment, you can use your mouse to rotate the attractor for additional clarity.

Solutions to the Problems for Lecture 57

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
theta0=0; u0=2; %initial ode conditions. u0 is initial guess for root.
inf=8*pi; %inf is a large number.  Takes a long time to get to top.
tspan=[0 inf];
options = odeset('RelTol',1.e-6);
%rootfind u0 such that theta(inf)=pi
alpha_i=linspace(0, 2, 100);
u0_i=zeros(100,1);
for i=1:length(alpha_i)
    alpha=alpha_i(i);
    u0_i(i) = fzero(@(u0) F(tspan,theta0,u0,alpha,options), u0);
end
plot(alpha_i, u0_i);
xlabel('$\alpha$','Interpreter','latex','FontSize',14);
ylabel('$d \theta/dt$','Interpreter','latex','FontSize',14);
title('Shooting to the Pendulum Top','Interpreter','latex','FontSize',16);

function y=F(tspan,theta0,u0,alpha,options)
% use ode45 to define the root-finding problem
end

function d_theta_u_dt = pendulum(theta_u,alpha)
% define the differential equation here
end
```

Your plot should look like this:



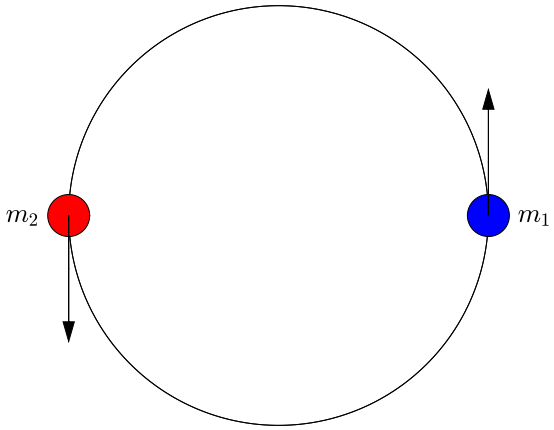
Solutions to the Problems for Lecture 58

1.

a) Consider $m_1 = m_2$. If $\mathbf{r} = \cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}$, then

$$\mathbf{r}_1 = \frac{1}{2}(\cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}), \qquad \mathbf{r}_2 = -\frac{1}{2}(\cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}).$$

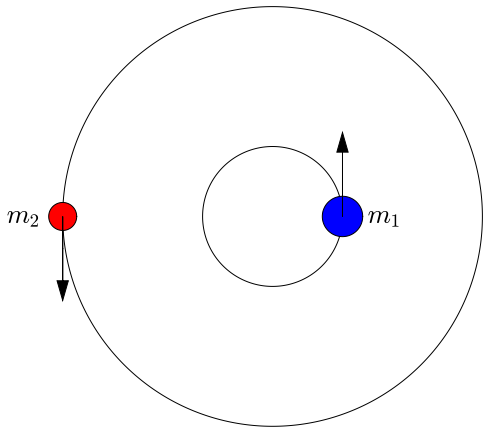
We can sketch the orbits as



b) Consider $m_1 = 3m_2$. If $\mathbf{r} = \cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}$, then

$$\mathbf{r}_1 = \frac{1}{4}(\cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}), \qquad \mathbf{r}_2 = -\frac{3}{4}(\cos(\omega t)\mathbf{i} + \sin(\omega t)\mathbf{j}).$$

We can sketch the orbits as



Solutions to the Problems for Lecture 59

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
e=0.7; m1=1; m2=4;
T=2*pi./(1-e).^1.5; tspan=linspace(0,T,1000);
options=odeset('RelTol',1.e-6);
%%%%% Solve differential equations for x and y using ode45
%%%%% with arguments tspan and options.
%%%%% Determine x1, y1 and x2, y2
%
%
%
%%%%% graphics %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k=0.1;
R1=k*(m1)^(1/3); R2=k*(m2)^(1/3); %radius of masses
theta = linspace(0,2*pi);
figure; axis equal; hold on; set(gcf,'color','w');
axis off;
xlim([-2,5]); ylim([-2.5,2.5]);
planet=fill(R1*cos(theta)+x1(1), R1*sin(theta)+y1(1),'b');
sun=fill(R2*cos(theta)+x2(1), R2*sin(theta)+y2(1),'r');
pause(1);
nperiods=5;  %number of periods to plot
for j=1:nperiods
    for i=1:length(t)
        planet.XData=R1*cos(theta)+x1(i); planet.YData=R1*sin(theta)+y1(i);
        sun.XData=R2*cos(theta)+x2(i); sun.YData=R2*sin(theta)+y2(i);
        drawnow;
    end
end
%%%%% Write local function for differential equations %%%%%%%%%%%%%%%
```

Solutions to the Practice quiz: Classify partial differential equations

1. a. The flow field is determined solely by the boundary conditions.

2. b. The turbulent motion is evolving in time.

3. a. The salt concentration is determined solely by the boundary conditions.

4. b. The salt concentration is evolving in time.

5. a. The potential is determined solely by the boundary conditions.

6. b. The potential is evolving in time.

Solutions to the Problems for Lecture 61

1. We use

$$y(x+2h) = y(x) + 2hy'(x) + 2h^2y''(x) + \frac{4}{3}h^3y'''(x) + \frac{2}{3}h^4y''''(x) + \mathrm{O}(h^5),$$
$$y(x+h) = y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \mathrm{O}(h^5),$$
$$y(x-h) = y(x) - hy'(x) + \frac{1}{2}h^2y''(x) - \frac{1}{6}h^3y'''(x) + \frac{1}{24}h^4y''''(x) + \mathrm{O}(h^5),$$
$$y(x-2h) = y(x) - 2hy'(x) + 2h^2y''(x) - \frac{4}{3}h^3y'''(x) + \frac{2}{3}h^4y''''(x) + \mathrm{O}(h^5).$$

Notice that

$$y(x+h) - y(x-h) = 2hy'(x) + \frac{1}{3}h^3y'''(x) + \mathrm{O}(h^5),$$
$$y(x+2h) - y(x-2h) = 4hy'(x) + \frac{8}{3}h^3y'''(x) + \mathrm{O}(h^5).$$

To eliminate the h^3 term, we can multiply the first expression by eight and subtract the second expression to obtain

$$8(y(x+h) - y(x-h)) - (y(x+2h) - y(x-2h)) = 12hy'(x) + \mathrm{O}(h^5).$$

Solving for the first derivative and rearranging terms, we find

$$y'(x) = \frac{-y(x+2h) + 8y(x+h) - 8y(x-h) + y(x-2h)}{12h} + \mathrm{O}(h^4).$$

Solutions to the Problems for Lecture 62

1. The discrete Laplace equation is given by

$$4\Phi_{i,j} - \Phi_{i+1,j} - \Phi_{i-1,j} - \Phi_{i,j+1} - \Phi_{i,j-1} = 0.$$

Solving for $\Phi_{i,j}$, we obtain

$$\Phi_{i,j} = \frac{1}{4}\left(\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1}\right).$$

The right-hand side is just the average value of Φ at the neighboring four grid points.

Solutions to the Problems for Lecture 63

1. The (i,j) coordinates for the four corners starting from the bottom left and moving clockwise are

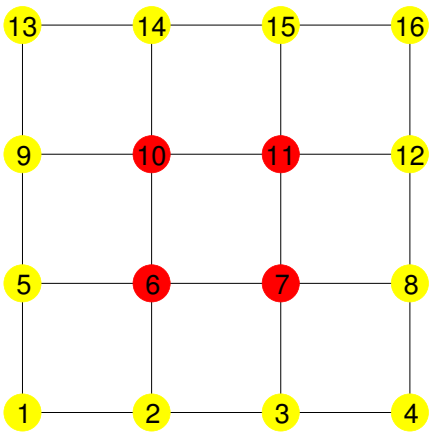
$$(i,j) = (1,1),\ (1,n_y),\ (n_x,n_y),\ (n_x,1).$$

Using the mapping $(i,j) \to k = i + (j-1)n_x$, the corresponding k coordinates are given by

$$k = 1,\ n_x n_y - n_x + 1,\ n_x n_y,\ n_x.$$

Solutions to the Problems for Lecture 64

1. The grid and its k indexing is shown below.



The matrix equation is given by

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16}
\end{pmatrix}$$

$$= \begin{pmatrix}
b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ 0 \\ 0 \\ b_8 \\ b_9 \\ 0 \\ 0 \\ b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \\ b_{16}
\end{pmatrix}$$

2. The number of interior points is the number of points in a rectangle with sides $n_x - 2$ and $n_y - 2$ and is given by $(n_x - 2)(n_y - 2)$. The number of boundary points is given by $2(n_x + n_y) - 4$. When $n_x = n_y = 100$, there are 9,604 interior points and 396 boundary points. When $n_x = n_y = 1000$, there are 996,004 interior points and 3,996 boundary points. The percentage of boundary points to total number of grid points are, respectively, 3.96% and 0.3996%, or about 4% and 0.4%. Boundary points make up a small fraction of a large grid, which is why we can include them in our matrix without much additional cost.

Solutions to the Problems for Lecture 65

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
%%%%% Define the rectangle and grid parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lx=1; Ly=1; %rectangle dimensions
Nx=100; Ny=100; %# of intervals
nx=Nx+1; ny=Ny+1; %# of gridpoints in x,y directions including boundaries
dx=Lx/Nx; dy=Ly/Ny; %grid size in x,y directions
x=(0:Nx)*dx; y=(0:Ny)*dy; %x,y values on the grid
%%%%% Define the indices associated with the boundaries %%%%%%%%%%%%%%%%%%%%
% boundary_index = [bottom, left, top, right]
boundary_index=[          1:nx,    1:nx:1+(ny-1)*nx, ...
             1+(ny-1)*nx:nx*ny,    nx:nx:nx*ny           ];
%%%%% Set up matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
diagonals = [4*ones(nx*ny,1), -ones(nx*ny,4)];
A=spdiags(diagonals,[0 -1 1 -nx nx], nx*ny, nx*ny); %use sparse matrices
I=speye(nx*ny);
A(boundary_index,:)=I(boundary_index,:);
%%%%% SET-UP RIGHT HAND SIDE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
b=zeros(nx,ny);
b(:,1)=...;          %bottom
b(1,:)=...;          %left
b(:,ny)=...;  %top
b(nx,:)=...;  %right
b=reshape(b,nx*ny,1); %make column vector
%%%%% Solve the Laplace equation using Gaussian elimination %%%%%%%%%%%%%%%%
Phi=A\b; %solution step (all the computational time is here)
Phi=reshape(Phi,nx,ny); %make matrix
%%%%% Graphics %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[X,Y]=meshgrid(x,y);
v=[DEFINE APPROPRIATE COUNTOUR LEVELS];
contour(X,Y,Phi',v,'ShowText','on');%requires transpose (read the notes)
axis equal;
set(gca, 'YTick', [0 0.2 0.4 0.6 0.8 1]);
set(gca, 'XTick', [0 0.2 0.4 0.6 0.8 1]);
xlabel('$x$','Interpreter','latex','FontSize',14 );
ylabel('$y$','Interpreter','latex','FontSize',14);
title('Solution of the Laplace equation','Interpreter','latex','FontSize',16);
```

Solutions to the Problems for Lecture 66

1.

$$x_1^{(n+1)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(n)} - a_{13}x_3^{(n)}\right),$$
$$x_2^{(n+1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(n)} - a_{23}x_3^{(n)}\right),$$
$$x_3^{(n+1)} = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(n)} - a_{32}x_2^{(n)}\right).$$

Solutions to the Problems for Lecture 68

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
%%%%% Define the rectangle and grid parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lx=1; Ly=1; %rectangle dimensions
Nx=100; Ny=100; %# of intervals
nx=Nx+1; ny=Ny+1; %# of gridpoints in x,y directions including boundaries
dx=Lx/Nx; dy=Ly/Ny; %grid size in x,y directions
x=(0:Nx)*dx; y=(0:Ny)*dy; %x,y values on the grid
%%%%% Define the iteration parameters and initial condition %%%%%%%%%%%%%%%%
eps=1.e-6; %convergence criteria for each value of Phi
index_x=2:nx-1;  index_y=2:ny-1; %internal grid points
Phi=zeros(nx,ny);%matrix with solution and boundary conditions
%%%%% DEFINE THE BOUNDARY CONDITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set the boundary conditions
Phi(:,1)=...;          %bottom
Phi(1,:)=...;          %left
Phi(:,ny)=...;         %top
Phi(nx,:)=...;         %right
%%%%% Jacobi iteration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Phi_old=Phi;
error=2*eps; ncount=0;
while (error > eps)
   ncount=ncount+1;
   Phi(index_x,index_y)=0.25*(Phi(index_x+1,index_y) ...
    +Phi(index_x-1,index_y)+Phi(index_x,index_y+1)+Phi(index_x,index_y-1));
   error=max(abs(Phi(:)-Phi_old(:)));
   if any(isnan(Phi(:))) || any(isinf(Phi(:)))
        fprintf('iterations diverge\n');
        return;
    end
    Phi_old=Phi;
    %fprintf('%g %e\n',ncount, error);
end
fprintf('%g\n',ncount);
%%%%% graphics %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[X,Y]=meshgrid(x,y);
v=[]; %SET THE CONTOUR LEVELS
contour(X,Y,Phi',v,'ShowText','on');%requires transpose (read the notes)
axis equal;
set(gca, 'YTick', [0 0.2 0.4 0.6 0.8 1]);
set(gca, 'XTick', [0 0.2 0.4 0.6 0.8 1]);
xlabel('$x$','Interpreter','latex','FontSize',14 );
ylabel('$y$','Interpreter','latex','FontSize',14);
title('Solution of the Laplace equation','Interpreter','latex','FontSize',16);
```

Solutions to the Problems for Lecture 69

1. The two-step solution of the one-dimensional diffusion equation using the modified

Euler method is given by

$$\tilde{u}_j^{l+1} = u_j^l + \frac{\Delta t D}{(\Delta x)^2}\left(u_{j+1}^l - 2u_j^l + u_{j-1}^l\right),$$
$$u_j^{l+1} = u_j^l + \frac{\Delta t D}{2(\Delta x)^2}\left[\left(u_{j+1}^l - 2u_j^l + u_{j-1}^l\right) + \left(\tilde{u}_{j+1}^{l+1} - 2\tilde{u}_j^{l+1} + \tilde{u}_{j-1}^{l+1}\right)\right].$$

2. The FTCS scheme for the advection equation is given by

$$u_j^{l+1} = u_j^l - \frac{c\Delta t}{2\Delta x}\left(u_{j+1}^l - u_{j-1}^l\right).$$

Solutions to the Problems for Lecture 70

1. We look for solutions of the form

$$u_j^l = \xi^l e^{ikj\Delta x}.$$

Substitution into the discrete advection equation and dividing by $\xi^l e^{ikj\Delta x}$ results in

$$\xi = 1 - \frac{c\Delta t}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) = 1 - i\frac{c\Delta t}{\Delta x}\sin(k\Delta x).$$

The modulus of ξ is given by

$$|\xi| = \sqrt{1 + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x)}.$$

The value of $|\xi|$ is largest when $\sin^2(k\Delta x) = 1$, and since for this value $|\xi| > 1$ no matter the value of Δt, the method is always unstable.

Solutions to the Problems for Lecture 71

1.

a) The implicit scheme for the advection equation is given by

$$u_j^{l+1} = u_j^l - \frac{c\Delta t}{2\Delta x}\left(u_{j+1}^{l+1} - u_{j-1}^{l+1}\right).$$

b) We look for solutions of the form

$$u_j^l = \xi^l e^{ikj\Delta x}.$$

Substitution into the implicit discrete advection equation and dividing by $\xi^l e^{ikj\Delta x}$ results in

$$\xi = 1 - \frac{c\Delta t\xi}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) = 1 - i\frac{c\Delta t\xi}{\Delta x}\sin(k\Delta x).$$

Solving for ξ, we obtain

$$\xi = \frac{1}{1 + i\frac{c\Delta t}{\Delta x}\sin(k\Delta x)}.$$

The modulus of ξ is given by

$$|\xi| = \frac{1}{\sqrt{1 + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x)}}.$$

The value of $|\xi|$ is largest when $\sin^2(k\Delta x) = 0$, and for this value $|\xi| = 1$. The method is unconditionally stable.

Solutions to the Problems for Lecture 72

1. The Lax scheme for the advection equation is given by

$$u_j^{l+1} = \frac{1}{2}(u_{j+1}^l + u_{j-1}^l) - \frac{c\Delta t}{2\Delta x}(u_{j+1}^l - u_{j-1}^l).$$

We look for solutions of the form

$$u_j^l = \xi^l e^{ikj\Delta x}.$$

Substitution into the Lax scheme and dividing by $\xi^l e^{ikj\Delta x}$ results in

$$\begin{aligned}\xi &= \frac{1}{2}\left(e^{ik\Delta x} + e^{-ik\Delta x}\right) - \frac{c\Delta t}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) \\ &= \cos(k\Delta x) - i\frac{c\Delta t}{\Delta x}\sin(k\Delta x).\end{aligned}$$

The modulus of ξ is given by

$$|\xi| = \sqrt{\cos^2(k\Delta x) + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x)}.$$

Clearly, the Lax scheme is stable provide that

$$\left(\frac{c\Delta t}{\Delta x}\right)^2 \leq 1.$$

which is called the Courant-Friedrichs-Lewy (CFL) stability criterion.

Solutions to the Problems for Lecture 73

1.

a) For x a boundary point on the left, we start with

$$\begin{aligned}y(x+h) &= y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \mathrm{O}(h^3), \\ y(x+2h) &= y(x) + 2hy'(x) + 2h^2y''(x) + \mathrm{O}(h^3).\end{aligned}$$

To eliminate the term proportional to h^2, we multiply the first equation by four and subtract the second equation to obtain

$$4y(x+h) - y(x+2h) = 3y(x) + 2hy'(x) + \mathrm{O(h^3)}.$$

Solving for $y'(x)$, we find

$$y'(x) = \frac{-3y(x) + 4y(x+h) - y(x+2h)}{2h} + \mathrm{O}(h^2).$$

For x a boundary point on the right, we start with

$$y(x-h) = y(x) - hy'(x) + \frac{1}{2}h^2y''(x) + \mathrm{O}(h^3),$$
$$y(x-2h) = y(x) - 2hy'(x) + 2h^2y''(x) + \mathrm{O}(h^3).$$

We multiply the first equation by four and subtract the second equation to obtain

$$4y(x-h) - y(x-2h) = 3y(x) - 2hy'(x) + \mathrm{O(h^3)}.$$

Solving for $y'(x)$, we find

$$y'(x) = \frac{3y(x) - 4y(x-h) + y(x-2h)}{2h} + \mathrm{O}(h^2).$$

b) Written in discretized form, the second-order method for the x- derivative at boundary points become

$$y'_1 = \frac{-3y_1 + 4y_2 - y_3}{2\Delta x}, \qquad y'_{n_x} = \frac{3y_{n_x} - 4y_{n_x-1} + y_{n_x-2}}{2\Delta x}.$$

We apply these equations to u^l_j and set the partial derivatives with respect to x to be zero at the boundary points. We can then solve for u at the boundaries to obtain

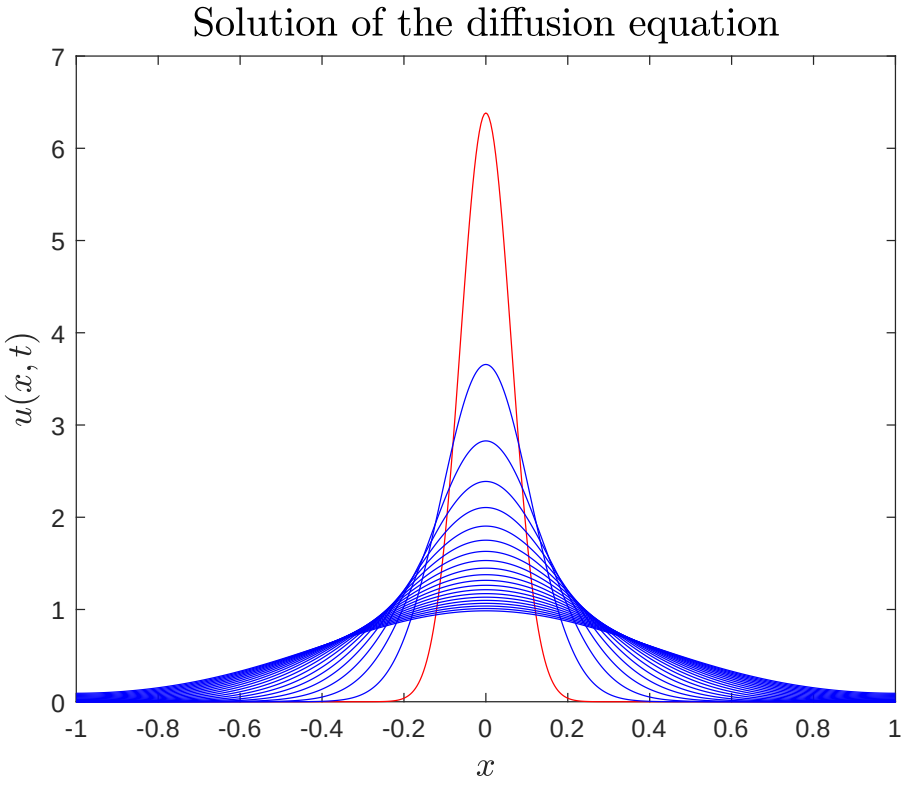
$$u^l_1 = \frac{1}{3}(4u^l_2 - u^l_3), \qquad u^l_{n_x} = \frac{1}{3}(4u^l_{n_x-1} - u^l_{n_x-2}).$$

2. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
D=1; %diffusion coefficient
%%%%% Define the x-domain and x-grid %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lx=1; %domain: -Lx < x < Lx
Nx=500; %# of intervals
nx=Nx+1;%# of gridpoints in x
dx=2*Lx/Nx; %grid length in x
x=-Lx + (0:Nx)*dx; %x values on the grid
%%%%% Time step parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nsteps=10000; %number of time steps
nout=500; %plot every nout time steps
```

```
dt=(dx)^2/(2*D); %borderline stability of FTCS scheme
alpha=dt*D/dx^2; %equation parameter
%%%%% Construct the matrix %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
diagonals = [2*(1+alpha)*ones(nx,1), -alpha*ones(nx,2)];
A=spdiags(diagonals,[0 -1 1], nx, nx);
I=speye(nx);
A([1 nx],:)=I([1 nx],:); %boundaries
%%%%% Define initial conditions and plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sigma=Lx/16;
u=1/(sigma*sqrt(2*pi))*exp(-0.5*(x/sigma).^2); u=u';
plot(x,u,'r'); hold on;
xlabel('$x$','Interpreter','latex','FontSize',14);
ylabel('$u(x, t)$','Interpreter','latex','FontSize',14);
title('Solution of the diffusion equation','Interpreter','latex','FontSize',16);
%%%%% Advance solution and plot %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m=1:nsteps
    b=[NO-FLUX BOUNDARY CONDITION; ...
        [alpha*u(1:nx-2) + 2*(1-alpha)*u(2:nx-1) + alpha*u(3:nx)]; ...
         NO-FLUX BOUNDARY CONDITION];
    u=A\b;
    if mod(m,nout)==0, plot(x,u,'b'), end
end
```

Your plot should look like this:



Solutions to the Problems for Lecture 74

1. Complete your solution on Coursera using MATLAB. Here is the Learner Template:

```
%%%%% Define the square and grid parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L=1;  %square is 2L x 2L
N=100; %# of intervals in x and y directions
n=N+1; %# of gridpoints in x,y directions including boundaries
h=2*L/N;  %grid size in x,y directions
x=-L + (0:N)*h; %x values on the grid
y=-L + (0:N)*h; %y values on the grid
[X,Y]=meshgrid(x,y);
%%%%% Define the indices associated with the boundaries %%%%%%%%%%%%%%%%%%%%
% boundary_index = [bottom, left, top, right]
boundary_index=[          1:n,       1:n:1+(n-1)*n, ...
                1+(n-1)*n:n*n,   n:n:n*n           ];
%%%%% Diffusion constant and time-step parameters
D=1;
dt=h^2/(2*D); %borderline stability of FTCS scheme
alpha=dt*D/h^2; %equation parameter
nsteps=1000; %number of time steps
%%%%% CONSTRUCT THE MATRIX AND COMPUTE LU DECOMPOSITION %%%%%%%%%%%%%%%%%%%%
%
%
%
%
%

%%%%% Define initial conditions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u=zeros(n,n,nsteps);
sigma=L/4;
u(:,:,1)=1/(2*pi*sigma^2)*exp(-0.5*(X.^2+Y.^2)/sigma^2);
u(1,:,1)=0; u(n,:,1)=0; u(:,1,1)=0; u(:,n,1)=0; %b.c.
%%%%% ADVANCE SOLUTION u %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for m=2:nsteps
%
%
%
%
%
%
end
%%%% Plot with animation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure('units','normalized','outerposition',[0 0 1 1])
s=surf(X,Y,u(:,:,1)); zlim([0, 2.6]);
xlabel('$x$','Interpreter','latex','FontSize',14);
ylabel('$y$','Interpreter','latex','FontSize',14);
zlabel('$u(x,y,t)$','Interpreter','latex','FontSize',14);
title('Solution of the 2D diffusion equation','Interpreter','latex','FontSize',16);
pause(1)
for j=2:nsteps
    s.ZData=u(:,:,j); pause(0.01);
end
```