MAT 331: Computer Assisted Mathematical Problem Solving
Fall 2015
Raluca Tanase

Course Information

Home Course Information Lectures & Projects

Course Description

Exploration of the use of the computer as a tool to gain insight into complex mathematical problems through a project-oriented approach. Students learn both the relevant mathematical concepts and ways that the computer can be used (and sometimes misused) to understand them. Interesting applications of mathematics to computer science are also discussed. Some of the specific topics that we will try to study this semester include linear algebra, graph theory and Markov chains, number theory and cryptography, dynamical systems, fractals, differential equations and computer graphics.

Click here to download a copy of the course syllabus. Please visit the **course website on Blackboard**.

Lectures & Office Hours

Lectures: Tuesdays & Thursdays 11:30-12:50pm in Mathematics S235;
Office hours: Tuesdays 10:30 - 11:30am in MLC;
Wednesdays 12 - 1pm and Thursdays 1 - 2:30pm in Math Tower 4-120, or by appointment.

Software

We will use *Mathematica*, which is a computational software program developed by Wolfram Research and used in many scientific, engineering, mathematical and computing fields, based on symbolic mathematics. *Mathematica* has a **comprehensive documentation** that we will make use of. *Mathematica* 10.2 is available for most operating systems (Windows, Macintosh, Linux, etc.).

Stony Brook students can download the Windows/Mac/Linux version of *Mathematica* from **Softweb**. You need your Stony Brook netID and netID password to log in to Softweb. To obtain an Activation Key for *Mathematica* you must visit the **Wolfram User Portal**. If it's your first time visiting the Wolfram User Portal, you must create a Wolfram ID and follow the steps in there to request an **Activation Key**.

In addition, you can use any of the campus SINC sites, or you can access the **Virtual SINC site**.

Grading Policy

There will be no exams. Grades will be computed using the following scheme:

- Homework – 20%
- Lab Activity – 15%
- Projects – 65%

Students are expected to attend class regularly and to keep up with the material presented in the lecture and the assigned reading. There will be roughly four or five homework assignments (containing short exercises involving mathematical proofs and Mathematica code) as well as three or four projects. You may work together on your homework assignments and projects, and you are encouraged to do so. However, all solutions must be written up independently.
A project is more like a term paper and you will be expected to devote a significant amount of time to doing it, as well as taking care with the presentation. The project should contain a detailed description of the problem or topic, what means were used to

solve it, the mathematical solution and the computer program (interactive model in Mathematica). The last project of the class may include also a short oral presentation at the end of the semester.

Raluca Tanase

Institute for Mathematical Science
Stony Brook University

office: Math Tower 4-120
phone: (631) 632-4005
e-mail: rtanase@math.stonybrook.edu

Home **Research** **Teaching** **MAT 331** **MAT 303**

About me

I am currently a Milnor Lecturer at the **Institute for Mathematical Sciences** at Stony Brook University. I obtained my Ph.D. in Mathematics from **Cornell University** in 2013, under the supervision of John H. Hubbard. I also have a M.S. in Computer Science from **Cornell University** and a M.S. in Mathematics from **Scoala Normala Superioara, Bucharest**.

I got my B.Sc. degree in Mathematics from the **University of Bucharest**, and my B.Eng. in Computer Science and Engineering from the **Polytechnic University of Bucharest**.

Research Interests

Dynamical Systems and Ergodic Theory, Complex Analysis in one or several variables,
Computer Science especially Databases and Large Scale Architecture Systems.

For a more detailed description of my research interests, please take a look at my **research page**.

Teaching

This spring I am teaching **MAT 331**, an introductory course about the interplay between mathematics and computer science. Click here for the old version of this course from **Fall 2015**. Last fall I taught **MAT 303**, an introductory course in ordinary differential equations and their applications. To see a list of courses that I have taught in the past at **Cornell University** and **Stony Brook University** please click **here** or select "Teaching" from the top menu.

Other events

I organized a **Special Session on "Holomorphic Dynamics"** at the **AMS Sectional Meeting** at Stony Brook University, March 19–20, 2016.

Last updated December 2016

MAT 331: Computer Assisted Mathematical Problem Solving

Fall 2015

Raluca Tanase

Lectures, Homework & Projects

Home Course Information Lectures & Projects

Schedule

A set of lecture notes for each class is available in .pdf format and .nb (*Mathematica* notebook). Please log in to Blackboard with your netID and password to download the solutions to the homework assignments.

Date	Topic	Reading	Assignments
Aug 25	Introduction & **Syllabus**	Intro (**nb**, **pdf**)	**HW 1** Due Sept 10 **Solutions**
Aug 27	Getting started with *Mathematica* (Linear Algebra)	Notes (**nb**, **pdf**)	
Sept 1	Getting started (Functions, Conditional Statements)	Notes (**nb**, **pdf**)	
Sept 3	Loop Constructions & Graph Theory in *Mathematica*	Notes (**nb**, **pdf**)	**HW 2** Due Sept 24 **Solutions**
Sept 8	*no class (Labor Day)*		
Sept 10	Graph Theory in *Mathematica*	Notes (**nb**, **pdf**)	
Sept 15	Mathematics of Web Search	Notes (**nb**, **pdf**)	
Sept 17	Building interactive models	Notes (**nb**, **pdf**), **Map**	
Sept 22	Local variables & Interactive models	Notes (**nb**, **pdf**)	**Project 1** Due Oct 18
Sept 24	Eigenvalues and eigenvectors of a matrix	**Lectures 1** & **2**	
Sept 29	Perron-Frobenius Theorem & Page Rank	**Lectures 3** & **4**	
Oct 1	Project Discussion	Notes (**nb**, **pdf**)	
Oct 6	Module & DynamicModule	Notes (**nb**, **pdf**)	
Oct 8	Symbolic & Numerical Solvers	Notes (**nb**, **pdf**)	**HW 3** Due Oct 29 **Solutions**
Oct 13	Differential equations	Lecture (**pdf**)	
Oct 15	Solving Differential Equations with *Mathematica*	Notes (**nb**, **pdf**)	

Oct 20	Systems of Linear & Nonlinear Differential Equations	Lecture (**pdf**)	**Project 2** Due Nov 18
Oct 22	Solving Systems of Diff Equations in *Mathematica*	Notes (**nb**, **pdf**)	
Oct 27	Stability of Equilibrium Points	Notes (**nb**, **pdf**)	
Oct 29	Phase Portraits for Systems of Nonlinear ODEs	Notes (**nb**, **pdf**)	
Nov 3	Limits Cycles and Chaotic Behavior	Notes (**nb**, **pdf**)	
Nov 5	Exercise: Using Locator in Interactive Models	Notes (**nb**, **pdf**)	
Nov 10	An Introduction to Cryptography	Notes (**nb**, **pdf**)	**Lab Exercises**
Nov 12	Modular Arithmethic & Affine Ciphers	Notes (**nb**, **pdf**)	
Nov 17	Cryptanalysis of Monoalphabetic/Polyalphabetic Ciphers	Notes (**nb**, **pdf**) Vigenere (**nb**, **pdf**)	
Nov 19	Euler's Theorem and the RSA Cryptosystem	Lecture (**pdf**)	**Project 3** Due Dec 7
Nov 24	RSA	Notes (**nb**, **pdf**)	
Nov 26	*no class (Thanksgiving)*		
Dec 1	Digital Signatures	Notes (**nb**, **pdf**)	
Dec 3	Project Discussion & Further Directions		
Dec 9	Project Presentation - Wednesday, December 9, 5:30-7:30pm		

Last updated November 2015

MAT 331: COMPUTER-ASSISTED MATHEMATICAL PROBLEM SOLVING
FALL 2015

GENERAL INFORMATION

Instructor. Raluca Tanase
Email: raluca.tanase@stonybrook.edu
Office: Math Tower 4-120; Phone: (631) 63**2-4005**
Office hours: Tuesdays 10:30–11:30am in MLC; Thursdays 1:00-2:30pm in Math Tower 4-120.

Lectures. Tuesdays & Thursdays 11:30–12:50pm in Mathematics S-235.

Blackboard. Grades and some course administration will take place on Blackboard. You will also use Blackboard to submit the projects and homework. Please log in using your NetID at http://blackboard.stonybrook.edu.

Courses Description. Exploration of the use of the computer as a tool to gain insight into complex mathematical problems through a project-oriented approach. Students learn both the relevant mathematical concepts and ways that the computer can be used (and sometimes misused) to understand them. Interesting applications of mathematics to computer science are also discussed. Some of the specific topics that we will try to study this semester include linear algebra, graph theory and Markov chains, number theory and cryptography, dynamical systems, fractals, differential equations and computer graphics.

Prerequisites. C or higher in MAT 203 or 205 or 307 or AMS 261.

TECH Objective. MAT 331 fulfills the ”Understand Technology (TECH)” objective:

1. Demonstrate an ability to apply technical tools and knowledge to practical systems and problem solving.
2. Design, understand, build, or analyze selected aspects of the human-made world. The human-made world is defined for this purpose as artifacts of our surroundings that are conceived, designed, and/or constructed using technological tools and methods.

WRTD Objective. Students may use two of their MAT 331 projects to satisfy part of the Upper Division Writing Requirement for the major, or the “Write Effectively within One's Discipline (WRTD)” objective for the Stony Brook Curriculum (SBC):

1. Collect the most pertinent evidence, draw appropriate disciplinary inferences, organize effectively for one's intended audience, and write in a confident voice using correct grammar and punctuation.

Students who want to use two of the MAT 331 projects for this purpose should sign up for MAT 459: *Write Effectively in Mathematics* as a zero-credit course, with me as instructor.

Software. Most lectures will be held in the Math computer lab (Math Tower S-235). No previous experience with computers is needed.

We will use *Mathematica*, which is a computational software program developed by Wolfram Research and used in many scientific, engineering, mathematical and computing fields, based on symbolic mathematics. *Mathematica* has a comprehensive documentation, also available online at http://reference.wolfram.com/language/.

Mathematica 10 is available for most operating systems (Windows, Macintosh, Linux, etc.). Stony Brook students can download the Windows/Mac/Linux version of *Mathematica* from Softweb: http://softweb.cc.stonybrook.edu/. You need your Stony Brook netID and netID password to log in to Softweb. To obtain an Activation Key for *Mathematica* you must visit the Wolfram User Portal https://user.wolfram.com/portal/login.html. If it's your first time visiting the Wolfram User Portal, you must create a Wolfram ID and follow the steps in there to request an Activation Key.

In addition, you can use any of the campus SINC sites, or you can access the Virtual SINC site at http://it.stonybrook.edu/services/virtual-sinc-site.

Reading resources. We will try to follow several sources, depending on the topic which we are covering. A set of notes written by Scott Sutherland and Santiago Simanca is available online at http://www.math.stonybrook.edu/~scott/Book331/331book.pdf. For the first part of the course we will use a set of lecture notes written by Raluca Tanase and Remus Radu about *The Mathematics of Web Search*, available at http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/. Other useful materials and lecture notes will be posted on the course website on Blackboard as we advance in the semester.

Grading policy. There will be no exams. Grades will be computed using the following scheme:

Lab	15%
Homework	20%
Projects	65%

Students are expected to attend class regularly and to keep up with the material presented in the lecture and the assigned reading. There will be roughly four or five homework assignments (containing short exercises involving mathematical proofs and *Mathematica* code) as well as three or four projects. You may work together on your homework assignments and projects, and you are encouraged to do so. However, all solutions must be written up independently. A project is more like a term paper and you will be expected to devote a significant amount of time to doing it, as well as taking care with the presentation. The project should contain a detailed description of the problem or topic, what means were used to solve it, the mathematical solution and the computer program (interactive model in Mathematica). The last project of the class may include also a short oral presentation at the end of the semester.

Extra Help. You are welcome to attend the office hours and ask questions about the lectures and about the homework. In addition, math tutors are available at the Math Learning Center (MLC): http://www.math.stonybrook.edu/MLC.

Information for students with disabilities. If you have a physical, psychological, medical or learning disability that may impact your course work, please contact Disability Support Services, ECC (Educational Communications Center) Building, Room 128, (631) 632-6748, or at the following website http://studentaffairs.stonybrook.edu/dss/index.shtml. They will determine with you what accommodations, if any, are necessary and appropriate. All information and documentation is confidential.

Academic integrity. Each student must pursue his or her academic goals honestly and be personally accountable for all submitted work. Representing another person's work as your own is always wrong. Faculty is required to report any suspected instances of academic dishonesty to the Academic Judiciary. Faculty in the Health Sciences Center (School of Health Technology & Management, Nursing, Social Welfare, Dental Medicine) and School of Medicine are required to follow their school-specific procedures. For more comprehensive information on academic integrity, including categories of academic dishonesty please refer to the academic judiciary website at http://www.stonybrook.edu/uaa/academicjudiciary.

Critical Incident Management. Stony Brook University expects students to respect the rights, privileges, and property of other people. Faculty are required to report to the Office of University Community Standards any disruptive behavior that interrupts their ability to teach, compromises the safety of the learning environment, or inhibits students' ability to learn. Faculty in the HSC Schools and the School of Medicine are required to follow their school-specific procedures. Further information about most academic matters can be found in the Undergraduate Bulletin, the Undergraduate Class Schedule, and the Faculty-Employee Handbook.

MAT 331: Computer Assisted Mathematical Problem Solving
Spring 2017
Raluca Tanase

Course Information

Home Course Information Lectures & Projects

Course Description

Exploration of the use of the computer as a tool to gain insight into complex mathematical problems through a project-oriented approach. Students learn both the relevant mathematical concepts and ways that the computer can be used (and sometimes misused) to understand them. Interesting applications of mathematics to computer science are also discussed. Some of the specific topics that we will try to study this semester include linear algebra, graph theory and Markov chains, number theory and cryptography, dynamical systems and fractals, differential equations and computer graphics.

Click here to download a copy of the course syllabus. Please visit the **course website on Blackboard**.

Lectures & Office Hours

Instructor: Raluca Tanase
Lectures: Tuesdays & Thursdays 11:30-12:50pm in Mathematics S235;
Office hours: Tuesdays 1-2pm in MLC (next to the computer lab)
Thursdays 1-3pm in Math Tower 4-120, or by appointment.

Teaching Assistant: Nancy Hong
Office hours: Wednesdays 12-1pm in Mathematics S235 (computer lab).

Software

We will use *Mathematica*, which is a computational software program developed by Wolfram Research and used in many scientific, engineering, mathematical and computing fields, based on symbolic mathematics. *Mathematica* has a **comprehensive documentation** that we will make use of. *Mathematica* 10.3 is available for most operating systems (Windows, Macintosh, Linux, etc.).

Stony Brook students can download the Windows/Mac/Linux version of *Mathematica* from **Softweb**. You need your Stony Brook netID and netID password to log in to Softweb. To obtain an Activation Key for *Mathematica* you must visit the **Wolfram User Portal**. If it's your first time visiting the Wolfram User Portal, you must create a Wolfram ID and follow the steps in there to request an **Activation Key**.

In addition, you can use any of the campus SINC sites, or you can access the **Virtual SINC site**.

Grading Policy

There will be no exams. Grades will be computed using the following scheme:

- Homework – 20%
- Lab Activity – 15%
- Projects – 65%

Students are expected to attend class regularly and to keep up with the material presented in the lecture and the assigned reading. There will be roughly five homework assignments (containing short exercises involving mathematical proofs and *Mathematica* code) as well as three or four projects. You may work together on your homework assignments and projects, and you are encouraged to do so. However, all solutions must be written up independently.

A project is more like a term paper and you will be expected to devote a significant amount of time to doing it, as well as taking care with its presentation. The project should contain a detailed description of the problem or topic, what means were used to solve it, the mathematical solution and proofs, and the computer program (interactive model in Mathematica). The last project of the class may include also a short oral presentation at the end of the semester.

Last updated February 2017

Getting Started with *Mathematica*

Mathematica

"For more than 25 years, Mathematica has defined the state of the art in technical computing and provided the principal computation environment for millions of innovators, educators, students, and others around the world."

Mathematica is based on the Wolfram Language, that you may have already seen when you used the online tool Wolfram Alpha http://www.wolframalpha.com.

A comprehensive Documentation for *Mathematica* is available online at
http://reference.wolfram.com/language/

Mathematica provides a simple framework for doing interactive models.
Wolfram Demonstration Projects http://demonstrations.wolfram.com/

In this course, we will learn how to use *Mathematica* to do a variety of numeric and symbolic computations. Then we will learn how to turn our static computations into dynamic interactive models.

Mathematica Notebooks (*.nb)

When Mathematica is first started, it displays an empty notebook with a blinking cursor. You can start typing in it right away. Let's start by doing a simple computation:

```
1 + 3 + 2 * 5
```

```
14
```

Mathematica by default will interpret your text as input. After you enter Mathematica input into the notebook, type Shift+Return (it's the same as Shift+Enter) to make Mathematica process your input. If your keyboard has a numeric keypad, you can use its Enter key instead of Shift+Return.

With a notebook interface, you just type in your computation, say 1+3+2*5. Mathematica will label your input with In[n]:=1+3+2*5. It labels the corresponding output Out[n]=14. Labels are added automatically and they reflect the order in which the input is evaluated by the *Mathematica* Kernel. By default, input/output pairs are grouped using rectangular cell brackets displayed in the right margin.

Notebooks as Documents

Mathematica notebooks are structured interactive documents that are organized into a sequence of cells. Each cell may contain text, graphics, sounds or *Mathematica* expressions in any combination. The extent of each cell is indicated by a bracket on the right.

Particularly in larger notebooks, it is common to have chapters, sections and so on, each represented by groups of cells. Having a structured document makes it easier later one to convert your notebook into a slide presentation. The various kinds of cells available are listed in Format->Style. The grouping of cells in a notebook is indicated by nested brackets on the right. Font, color, spacing and other properties of the appearance of cells can also be changed from Format.

Example of a Section

Section

Subsection I (click Alt-Return to create below another cell of the same type)

Subsection II

```
This subsection has a lot of text and some numbered lists. The text
 is blue because this cell has not been properly formated to text,
so Mathematica sees it as some sort of long command. As we continue to type in,
Mathematica will suggest that we convert the cell to text or to free -
 form linguistic input.
```

1. Numbered list
   1.1. And a sublist
   1.2. .....
2. The first list *continues* here

Subsection III

Notebook Style

To change the style of a cell, click the cell bracket. The bracket is higlighted. Select a style from Format->Style. The cell will immediately reflect the change.

To change the overall look of a notebook, choose Format->Stylesheet. Select a stylesheet from the menu. All cells in the notebook will change appearance, based on the definitions in the new stylesheet.

Free-Form Input

Free-Form Input allows users to enter plain English and get immediate results, as well as the Mathematica correct input for further exploration—without the need for syntax.

In the beginning, you can learn Mathematica syntax by entering a query in plain English and then viewing the free-form queries translated in precise Mathematica commands.

Click on the Cell Insertion Assistant (+ Sign at the left) and choose Free-Form Input. Alternatively, click Insert->Inline Free Form Input. An orrange square with an equal sign will appear in the cell. The cursor will be blinking to the right of the orange square.

Use Plain English after the equal sign to describe the command that you would like to do, say "integral of cos x"

After clicking Shift+Enter, we will see the command that Mathematica translated our free-form input into. You need to have an active Internet Connection to use Free-Form Input. Use the + Sign to the right of the Free-Form Input to show all results.

```
integral of cos x
Integrate[Cos[x], x]
```

```
Sin[x]
```

Using Palettes

Here is one way to enter a particular expression.

```
2^10 + 1/2 + Sqrt[2]
```

$$\frac{2049}{2}+\sqrt{2}$$

We can use the Palettes to typeset various basic commands and text.
Select Palettes->Basic Math Assistant, or **Palettes->Classroom Assistant**. We can then write expressions like square roots $\sqrt{\blacksquare}$, integrals $\int_{\square}^{\square}\square\,d\square$ and many others, by simply using the predefined templates from the Palletes. The same mathematical expression from above can be written more elegantly as follows:

$$2^{10}+\frac{1}{2}+\sqrt{2}$$

$$\frac{2049}{2}+\sqrt{2}$$

To get a numerical approximation of the expression we need to use N[..]. The percent sign % refers to the output of the last evaluation that *Mathematica* performed.

```
N[%]
```

```
1025.91
```

Matrices and Linear Algebra

The Mathematica front end provides an **Insert→Table/Matrix** submenu for creating and editing arrays with any specified number of rows and columns. Once you have such an array, you can edit it to fill in whatever elements you want.

Some examples of vectors and matrices:

$$A=\begin{pmatrix}1 & 0 & 0\\ 1 & 1 & 1\end{pmatrix}$$

```
{{1, 0, 0}, {1, 1, 1}}
```

```
IdentityMatrix[3]
```

```
{{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}
```

From these examples, we can see that Mathematica treats a matrix like a list of lists.

```
Transpose[A]
```

```
{{1, 1}, {0, 1}, {0, 1}}
```

Transpose[A] finds the transpose of the matrix A and displays it as a list of lists. To see the transpose in standard matrix form, use the command MatrixForm[..] with argument Transpose[A].

```
MatrixForm[%]
```

$$\begin{pmatrix}1 & 1\\ 0 & 1\\ 0 & 1\end{pmatrix}$$

Vectors

A vector in *Mathematica* is a list.

```
w = {1, 2, 3}
```

```
{1, 2, 3}
```

```
MatrixForm[w]
```

$$\begin{pmatrix}1\\ 2\\ 3\end{pmatrix}$$

We can also use the Insert->Table/Matrix submenu to create a column vector with a given number of elements, as a matrix with n rows and one column $\begin{pmatrix}\square\\\square\\\square\end{pmatrix}$. If we use this template, the vector is interpreted as a list of lists.

$$\mathbf{v} = \begin{pmatrix}\mathbf{1}\\\mathbf{2}\\\mathbf{3}\end{pmatrix}$$

```
{{1}, {2}, {3}}
```

Matrix and Vector Operations

We can perform matrix addition A+B and scalar multiplication 3A in the usual way. We can use either a space or the symbol * to do the multiplication between the scalar 3 and the matrix A.

```
MatrixForm[3 A]
```

$$\begin{pmatrix}3 & 0 & 0\\3 & 3 & 3\end{pmatrix}$$

```
MatrixForm[A + A]
```

$$\begin{pmatrix}2 & 0 & 0\\2 & 2 & 2\end{pmatrix}$$

However, to do matrix multiplication we cannot use the symbol * , instead we have to use a dot!

```
A.Transpose[A]
```

```
{{1, 1}, {1, 3}}
```

```
A.w
```

```
{1, 6}
```

```
A.v
```

```
{{1}, {6}}
```

```
MatrixForm[A.v]
```

$$\begin{pmatrix}\mathbf{1}\\\mathbf{6}\end{pmatrix}$$

Mathematica can also do symbolic computations:

```
M = {{a, b}, {c, d}}; x = {x1, x2};
MatrixForm[M.x]
```

$$\begin{pmatrix}a\,x1 + b\,x2\\c\,x1 + d\,x2\end{pmatrix}$$

For square matrices, we can compute the power of a matrix A^n using the command **MatrixPower[matrix, power]**

```
B = {{1, 1}, {1, 1}}; MatrixPower[B, 2]
```

```
{{2, 2}, {2, 2}}
```

```
M3 = MatrixPower[M, 3];
MatrixForm[M3]
```

$$\begin{pmatrix}a\left(a^2 + b\,c\right) + b\left(a\,c + c\,d\right) & a\left(a\,b + b\,d\right) + b\left(b\,c + d^2\right)\\c\left(a^2 + b\,c\right) + d\left(a\,c + c\,d\right) & c\left(a\,b + b\,d\right) + d\left(b\,c + d^2\right)\end{pmatrix}$$

MAT 331: Homework 1

Problem 1.1

Write a command in *Mathematica* to generate a square nxn matrix *M*, whose diagonal elements $M_{ii}$are equal to n-1 and the other elements M_{ij} i≠j are equal to 1. Test it for n=3 and n=10.

Problem 1.2

Demonstrate that two matrices do not commute in general, i.e. that AB ≠ BA, by defining two 5x5 random matrices using the function Table, computing both sides (AB and BA) using matrix multiplication, and comparing the outputs using the function TrueQ[...] in *Mathematica*. Of course, there exist matrices which do commute so you will have to chose your examples well. This means that you may have to generate several random 5x5 matrices, before you hit a pair for which AB ≠ BA**.**

Problem 1.3

1. Use free-form input to find some *Mathematica* functions which help you find the dimensions of a matrix (that is, the number of columns and rows).
2. Write a function g[A_,B_] that takes as input two matrices A and B and checks whether the number of columns of A is the same as the number of rows of B. If the dimensions match, then the function returns the product AB, otherwise it returns the message “Error, the matrices cannot be multiplied”.

Problem 1.4

Mathematica has built-in routines for finding the eigenvalues and eigenvectors (real and complex) of a numerical square matrix. *Recall that If A is an n × n matrix, then the number λ is an eigenvalue of A if there exists a non-zero vector x such that* ***Ax=λx.*** *The vector x is called an eigenvector of A with corresponding eigenvalue λ. There are at most n distinct eigenvalues of the matrix A, and at most n linearly independent eigenvectors with real or complex entries.*

0.1. The function Eigenvalues[A] returns a list with the eigenvalues of the square matrix A.

0.2. Sometimes it is necessary to know the eigenvectors of a square matrix A, as well as its eigenvalues. This can be done using the function Eigensystem[A], which finds both the eigenvalues and a complete linearly independent set of eigenvectors for each eigenvalue. The output is given in the form { list of eigenvalues, list of eigenvectors }.

1. Use the function Eigensystem[...] to find the eigenvalues and a set of linearly independent eigenvectors for the matrix M that you defined in Problem I.I. You may choose n=3 and n=10 in Problem I.I.
2. Define a matrix *P* whose columns are the eigenvectors returned by the function Eigensystem[...]. Display *P* in matrix form, then compute the product $P^{-1}MP$ where $P^{-1}$is the inverse of the matrix *P*. What do you observe?

3. Let M^Tbe the transpose of the matrix *M*. Find the eigenvalues and eigenvectors of $M^T$and *M*. What do you observe? What can you say about the eigenvalues and the eigenvectors of $M^T$and *M* in case *M* is a random matrix?

Problem 1.5

One of the main applications of matrix algebra is to solve systems of linear equations, usually in a large number of variables. A system of m such equations in the n variables x_1, x_2, ..., x_n, can be written explicitly:

$a_{11}x_1 + a_{12}x_2 + ... + a_{1\,n}\,x_n = b_1$

$a_{21}x_1 + a_{22}x_2 + ... + a_{2\,n}\,x_n = b_2$

....

$a_{\mathrm{m}1}x_1 + a_{\mathrm{m}2}x_2 + ... + a_{\mathrm{mn}}\,x_n = b_m$

The same system can be written in the more convenient matrix notation as Ax=b, where A is the m × n coefficient matrix, x is the (column) vector of length n containing the variables, and b is the (column) vector of length m of the coefficients on the right hand sides of the equalities.

By general theory, a system of linear equations has no solution, exactly one solution or infinitely many solutions. In *Mathematica*, we can use the command Solve[expression,variables], to solve the system Ax=b for the variable x.

In the code below we define a 2x2 matrix A and a vector b of length 2, both with random entries 0 or 1. Then we use the function Solve[A.x==b,x] to solve the system Ax=b, where x is a vector of 2 variables x[1] and x[2]. The function Solve *r*eturns an empty list if there is no solution, a list with one solution if the system has one solution, or a list which contains the dependecy between the free variables and the dependent variables if the system has infinitely many solutions (like x[1]=-x[2]).

```
A = Table[RandomInteger[], {i, 2}, {j, 2}]; Print["A=" MatrixForm[A]];
b = Table[RandomInteger[], {j, 2}]; Print["b=" MatrixForm[b]];
X = Table[x[j], {j, 2}];
Solve[A.X == b, X]
```

Run the code several time to generate a couple of random 2x2 matrices and see how the solution set is displayed. Then modify the code above to answer the following questions:

1. Let A be a random 3x3 matrix with integer coefficients. Find the solution set of the system Ax=$\begin{pmatrix}1\\0\\0\end{pmatrix}$.

2. Find the inverse of a random 3x3 matrix A with integer coefficients (if it exists!), without using the *Mathematica* function Inverse[...]. *Recall that the matrix A is called invertible if there exists a (unique) 3x3 matrix B such that AB=BA=I_3, where I_3 is the identity matrix. If A is invertible, then the matrix B is called the inverse of the matrix A, and it is denoted by A^{-1}.*

Getting Started with *Mathematica*

Basic calculations and variables

Mathematica is a powerful tool for doing mathematics. It can handle both symbolic and numeric expressions. In the simplest case you can use it just like a calculator: you type in questions, and *Mathematica* prints back answers.

```
27 + 5 ^ 3
```

You can also use variables or put several commands, separated by semicolons, on one line. A semicolon also has the effect of suppressing the corresponding output. To write the power as a superscript, go to Palettes->Basic Math Assistant.

In[21]:= $x = 5;\ \ y = 27;\ \ y + x^3$

```
Out[21]= 152
```

We can also write each command on a separate line (enter/return) within the same cell.

$$\begin{array}{l} x = 5 \\ y = 27; \\ y + x^3 \end{array}$$

Writing several commands not separated by semicolons on the same line can lead to different results. By default, a space between two expressions is interpreted as multiplication! Try writing something like x=5 27+x^3! What do you get?

You might have noticed that Mathematica assigns numbers to every command you input and to the corresponding output. In particular, all results are stored and you can use them in any following calculations as long as you do not quit *Mathematica* (more precisely the kernel).

```
Out[1] 3
```

In addition, the shortcut % can be used to refer to the previous result

```
% / 3
```

Moreover, Mathematica is case sensitive.

```
X + x
```

Symbolic versus Numeric Computations

$$\begin{array}{l} a = 2^{10} + \frac{1}{2} + \sqrt{2} + \text{Pi} \\ b = \text{N}[a] \\ c = \text{N}[a,\ 10] \\ d = 2^{10} + \frac{1}{2} + \sqrt{2.0} + \text{Pi} \end{array}$$

You can use the **N[..]** command to get a numeric answer. In addition, this allows to give the number of digits you want as an optional argument.

Exercise 1 : Use the sinus function Sin[..] to compute sinus of angle 2, with precision of one hundred decimal points

Matrices and Linear Algebra

The *Mathematica* front end provides an Insert->Table/Matrix submenu for creating and editing arrays with any specified number of rows and columns. Once you have such an array, you can edit it to fill in whatever elements you want.

Some examples of vectors and matrices:

Mathematica treats a matrix like a list of lists. To see it in standard matrix form, use the command MatrixForm[..]

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

```
IdentityMatrix[3]
MatrixForm[%]
```

A vector can be given as a list.

```
w = {1, 2, 3}
MatrixForm[w]
```

We can also use the Insert->Table/Matrix submenu to create a column vector with a given number of elements, as a matrix with n rows and one column $\begin{pmatrix} \square \\ \square \\ \square \end{pmatrix}$. If we use this template, the vector is interpreted as a list of lists.

$$v = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Matrix and Vector Operations

https : // reference.wolfram.com/language/guide/MatrixOperations.html

We can perform matrix addition A+B and scalar multiplication 3A in the usual way. However, to do matrix multiplication we cannot use the symbol * , instead we have to use a dot!

```
A + A; MatrixForm[%]
MatrixForm[3 A]
A.A
A.Transpose[A]
```

```
A.w
A.v
MatrixForm[A.w]
MatrixForm[A.v]
```

Transpose[matrix] -- Computes the transpose
MatrixPower[matrix, power] -- For square matrices, we can compute the power of a matrix A^n
Eigenvalues[matrix] -- Finds the eigenvalues of a square matrix
Eigensystem[matrix] -- Finds finds both the eigenvalues and a complete linearly independent set of eigenvectors for each eigenvalue.
Eigenvectors[matrix] -- Finds the eigenvectors of a square matrix

```
B = {{1, 1}, {1, 1}}; MatrixPower[B, 3]
```

Exercise 2 : Find the eigenvalues and eigenvectors of matrix B.

Changing Elements

To access different matrix or vector elements use [[..]]

```
In[7]:=  w = {10, 20, 30}; w[[1]]

Out[7]=  10
```

```
In[8]:=  B = {{10, 20}, {30, 40}};
         B[[1]]

Out[9]=  {10, 20}
```

```
In[10]:=  B[[1]][[1]]
          B[[1, 1]]

Out[10]=  10

Out[11]=  10
```

To change elements of a matrix or vector:

```
In[12]:=  w[[1]] = 100; w

Out[12]=  {100, 20, 30}
```

Length[..] -- returns the number of elements in a list.

Exercise 3 : What is the length of w? How about the length of B?

Exercise 4 : Write a *Mathematica* command that gives the sum of the elements of the vector w.

Other ways of generating a matrix:

Table[f, {i,m}, {j,n}] builds an mxn matrix by evaluating the function f with arguments i and j, where i ranges from 1 to m and j ranges from 1 to n. The lower bound is implicitly 1, so we only specify the upper bound.

The following commands generate random 4x4 matrix with entries between 0 and 1:

```
In[13]:=  Rn = Table[Random[], {i, 4}, {j, 4}]

Out[13]=  {{0.727347, 0.0091694, 0.716267, 0.20713}, {0.597192, 0.089866, 0.550936, 0.0729925},
           {0.24518, 0.516913, 0.86244, 0.519886}, {0.823348, 0.798585, 0.986709, 0.0177835}}
```

```
In[14]:=  Ri = Table[RandomInteger[], {4}, {4}]

Out[14]=  {{1, 0, 1, 0}, {1, 1, 0, 1}, {1, 1, 1, 1}, {0, 0, 1, 0}}
```

We may also want to define our own function f.

```
In[15]:=  f[i_, j_] := i + j
```

In this expression the underscore represents a pattern. A single underscore will match a single expression (argument) and x is the name of the pattern (it can be used to refer to the matched expression on the right hand side). The colon tells Mathematica not to evaluate the right hand side. It will only be evaluated when you invoke the function. This is why you get no output. Now you can use your function:

```
In[16]:=  f[2, 3]

Out[16]=  5
```

Try to assign some values to i and j, then compute f[2, 3]. What do you notice? Does f use the global or the local values of i and j when computing f[2, 3]?

```
In[17]:=  i = 0;
          j = -2;
          f[2, 3]

Out[19]=  5
```

◇ **Warning:** Do not omit the colon when defining functions!

```
In[22]:=  g[x_] = x^2 + Sin[x]

Out[22]=  25 + Sin[5]
```

Can you explain this result? What is the function g? You can look at the definition of a function by using a question mark:

```
? f
```

```
? g
```

To clear the definition of function g, use the command **Clear[g].**

We can then use function f to generate a 4x4 matrix whose i,j entry is i+j.

```
In[23]:=  NewM = Table[f[i, j], {i, 4}, {j, 4}];
          MatrixForm[NewM]

Out[24]//MatrixForm=
          ⎛2  3  4  5⎞
          ⎜3  4  5  6⎟
          ⎜4  5  6  7⎟
          ⎝5  6  7  8⎠
```

Exercise 5: Generate a 5x5 matrix C
with Cij=i if i>=j and
Cij=0 otherwise

The syntax of the “If” command in *Mathematica* is
If[condition, t, f] -- the “If” command returns t if condition evaluates to true and f if condition evaluates to false.

```
If[3 > 4, 10, 5]
```

```
If[3 < 4, 10, 5]
```

Functions, Conditional Statements

Functions

Mathematica has a large amount of functions already built in. The arguments of a function are put in between square brackets and separated by commas. Some basic functions are given below. Note that all built in functions and variables start with capital letters. To avoid potential conflicts it is a good idea to start your own functions and variables with lower case letters.

Sqrt[x]	square root (Sqrt[x])
Exp[x]	exponential (e^x)
Log[x]	natural logarithm (Subscript[log, e] x)
Log[b, x]	logarithm to base b (Subscript[log, b] x)
Sin[x], Cos[x], Tan[x]	trigonometric functions (with arguments in radians)
ArcSin[x], ArcCos[x], ArcTan[x]	inverse trigonometric functions
Abs[x]	absolute value
Round[x]	closest integer to x

For example, we can compute:

```
Exp[3]
N[Exp[3]]
Round[Sqrt[5]]
```

User defined functions

We can define our own functions as follows:

```
f[x_] := x^2 + Sin[x]
```

In this expression the underscore represents a pattern. A single underscore will match a single expression (argument) and **x** is the name of the pattern (it can be used to refer to the matched expression on the right hand side). The colon tells *Mathematica* not to evaluate the right hand side. It will only be evaluated when you invoke the function. This is why you get no output. Now you can use your function:

```
f[angle] + Pi
```

```
f[2 Pi]
```

Try to assign some value to x, then compute f[2Pi]. What do you notice? Does f use the global or the local values of x when computing f[2Pi]?

```
x = 5;
f[2 Pi]
```

◇ **Warning:** Do not omit the colon when defining functions!

```
g[x_] = x^2 + Sin[x]
```

Can you explain this result? You can look at the definition of a function by using a question mark:

```
? f
```

```
? g
```

To clear the function or variable definitions, use Clear[..]

```
Clear[x, f, g]
```

```
f[i_, j_] := i + j
```

The function f gives the sum of two variables. It works for any type of arguments that can be added in *Mathematica*. Let us evaluate f with some integer arguments, and then with some vector arguments.

```
f[2, 3]
f[{1, 2, 3}, {3, 4, 5}]
```

The condition operator /;

Many functions are only valid if certain conditions are met. Mathematica can handle such situations. The Condition operator (/;) can be used to make sure a pattern is used only if a condition is met.

```
fun[x_] := 0 /; x < 0
fun[0] := 1
fun[x_] := Log[x] /; x > 0
```

You can list all definitions associated with a function as before. In particular, this will also show the order in which the definitions are applied.

```
? fun
```

Exercise: Compute fun[0], fun[-5] and fun[9].

You can also place the condition operator in the argument of the function

```
f1[x_ /; x < 0] := 0
f1[0] := 10
f1[x_ /; x > 0] := Log[x]
```

```
f1[5]
```

Some functions only make sense for some type of input, like integers.

IntegerQ[expr] true if expr is an integer, false otherwise
EvenQ[expr] even number
OddQ[expr] odd number
PrimeQ[expr] prime number
NumberQ[expr] explicit number of any kind
NumericQ[expr] numeric quantity
VectorQ[expr] a list representing a vector
MatrixQ[expr] a list of lists representing a matrix
FreeQ[expr, form] form matches nothing in expr
MatchQ[expr, form] expr matches the pattern form
TrueQ[expr] tests whether expr is true

```
f3[x_ /; NumberQ[x]] := x^3
```

Evaluate the function f3

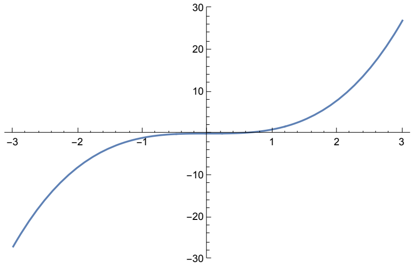
```
f3[3]
```

```
f3[{1, 2, 3}]
```

Plotting a function

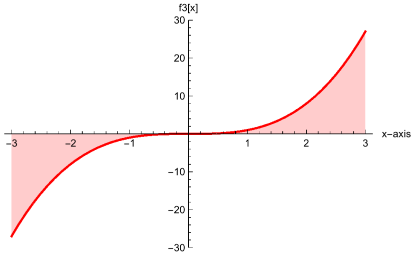
Documentation available at
http : // reference.wolfram.com / language / ref / Plot.html

```
Plot[f3[x], {x, -3, 3}]
```



There are a number of options that can be used with the Plot command, see the *Mathematica* documentation for a comprehensive list of available options and examples. For instance, **AxesLabel** specifies labels for the x and y axes. **PlotStyle** sets the style of the curve. You can fill in the space between the curve and the x-axis using the option **Filling**.

```
Plot[f3[x], {x, -3, 3}, AxesLabel → {"x-axis", "f3[x]"},
 PlotStyle → {Red, Thick}, Filling -> Axis]
```



```
Plot[f1[x], {x, -3, 3}]
```

Conditional Statements

If[condition, t, f] - the “If” command returns t if condition evaluates to true and f if condition evaluates to false.

```
3 > 4
```

```
If[3 > 4, 10, 5]
```

```
If[3 < 4, 10, 5]
```

Elementary numerical relations :

==	(is equal to)	!=	(is not equal to)
<	(is less than)	<=	(is less than or equal to)
>	(is greater than)	>=	(is greater than or equal to)

Elementary logical relations :
|| (or) && (and) ! (not)

```
x = 2; If[x == 0, Print["x is 0"], Print["x is different from 0"]]
```

Exercise : Write some If[..] command that tests whether x is a prime number between 10 and 20. If true, it prints x, otherwise, it prints x + 1.

More on Conditional Statements

Sometimes we only need to perform an action when the condition is true, so we can use the command If[..] with only two arguments.

```
x = 0; If[x == 0, Print["x is 0"]]
```

If we want to execute more commands when x is 0, we put a semicolon ; between the successive commands.

```
x = 0; If[x == 0, Print["x is 0."]; x = x + 2;
 Print["We have assigned the value 2 to x."],
 Print["x is not 0"]]
```

Switch

Switch[expr,form1,value1,form2,value2,…,_,default] - compares expr with each of the form form1, form 2, ... , giving the value associated with the first form it matches. If expr does not match any form, then the default value is returned.

```
expr = 3;
Switch[expr,
       1, Print["expr is 1"],
       2, Print["expr is 2"],
       3, Print["expr is 3"],
       _, Print["expr has some other value"]]
```

Recall from last time

Table[f, {i,m}, {j,n}] builds an mxn matrix by evaluating the function f with arguments i and j, where i ranges from 1 to m and j ranges from 1 to n. The lower bound is implicitly 1, so we only specify the upper bound.

The following commands generate random 4x4 matrix with entries between 0 and 1:

```
Rn = Table[Random[], {i, 4}, {j, 4}]
```

```
Ri = Table[RandomInteger[], {4}, {4}]
```

We can also define our own functions and use them to generate matrices with certain patterns.

```
Clear[f];
f[i_, j_] := i + j
```

We can then use function f to generate a 4x4 matrix whose i,j entry is i+j, using the command **Table[..]**

```
NewM = Table[f[i, j], {i, 4}, {j, 4}];
MatrixForm[NewM]
```

Getting Started with *Mathematica (II)*

Loops (For, While, Do)

Control structures (Break, Return)

Graphs and adjacency matrices

Loops and control structures

Do[expr,{i,max}] - evaluates expr repetitively, with i varying from 1 to max in steps of 1
Do[expr,{i,min,max,di}] - evaluates expr with i varying from min to max in steps of di.
Do[expr,{i,list}] - evaluates expr with i taking on values from a list
Do[expr,{n}] - evaluates expr n times

Examples with Do:

Print the first 10 positive integers:

```
Do[Print[i], {i, 10}]
```

Print only the odd integers between 4 and 10

```
Do[Print[i], {i, 5, 10, 2}]
```

```
Do[If[Mod[i, 2] != 0, Print[i]], {i, 4, 10, 1}]
```

Loops and control structures (Break, Return)

We can use some control flow functions to terminate the loop:
Break[ ] - causes the loop to terminate
Return[expr] - causes the loop/function to terminate and returns the value expr

Print the first odd integer between 4 and 10

```
Do[If[Mod[i, 2] != 0, Print[i]; Break[]], {i, 4, 10, 1}]
```

Print the first odd integer between 4 and 10

```
Do[If[Mod[i, 2] != 0, Return[i]], {i, 4, 10, 1}]
```

Loops (For)

For[start,test,incr,body] - evaluates start, then repetitively evaluates body and incr, until test fails

```
For[i = 1, i < 4, i++, Print[i]]
```

First i is assigned the value 1. As long as the condition i < 5 remains true, i is incremented by 1 (i++ is just a shortening for i = i + 1) and the body of the function For is executed (that is, we print the integer i).

```
v = {1, 2, 5, 4, 10}
```

Recall that to access the first element of the list, we use v[[1]].

```
v[[1]]
```

We can use a For loop to find the product, as follows

```
prod = 1;
For[i = 1, i ≤ Length[v], i++, prod = prod * v[[i]]];
prod
```

Mathematica also has a built-in command for finding the product of the elements of a list/vector.

```
Product[v[[i]], {i, Length[v]}]
```

Loops (While)

While[condition, body] - evaluates body repetitively, so long as condition is True

```
n = 17; While[(n = Floor[n / 2]) ≠ 0, Print[n]]
```

Even if the last value printed by the While function is 1, the current value of n is not 1, as one may assume, but 0. This value was not printed when we ran the While command, because when n is 0, the condition n != 0 is false, so the body of the function While (which would have been Print[0]) is not executed.

```
n
0
```

In the Wolfram Language, both While and For always evaluate the loop test before evaluating the body of the loop. As soon as the loop test fails to be True, While and For terminate. The body of the loop is thus only evaluated in situations where the loop test is True.

```
While[False, Print["False"]]
```

The command While[False, Print["False"]] does not print anything, because the test condition is always False, so the body of the function is never executed.

The functions While and For in the Wolfram Language are similar to the control structures While and For in languages such as C++. Notice, however, that there are a number of important differences. For example, the roles of comma and semicolon are reversed in Wolfram Language For loops relative to C++ language ones.

Exercise: Find the sum of the elements of a matrix.

```
A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
MatrixForm[A]
```

We can use Do:

```
sumElements = 0;
Do[sumElements = sumElements + A[[i]][[j]],
  {i, 1, Length[A]}, {j, 1, Length[A]}];
Print[sumElements]
```

We can use two For loops to find the sum, as follows:

```
sumElements  = 0;
For[i = 1, i ≤ Length[A], i ++,  For[j = 1, j ≤ Length[A],
    j ++, sumElements = sumElements  + A[[i]][[j]] ]];
Print[sumElements]
```

We can use two While loops:

```
sumElements  = 0;
i = 1;
While[i ≤ Length[A],
  j = 1;
  While[j ≤ Length[A],
   sumElements = sumElements + A[[i]][[j]];
   j ++ ];
  i ++
 ];
Print[sumElements]
```

We can also use a built-in command in *Mathematica* for computing the sum, whose syntax resembles that of a Do command.

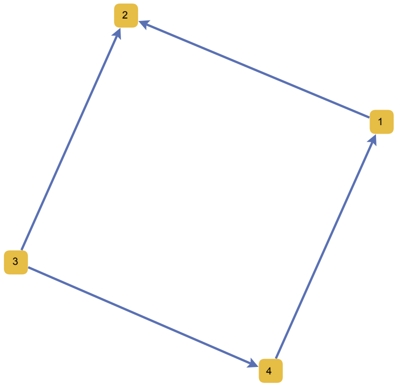
```
Sum[A[[i]][[j]], {i, 1, 3}, {j, 1, 3}]
```

Graphs

Definition: A graph G={V, E} consists of a set of vertices V (also called nodes) and a set of edges E. If there is an edge between the nodes u and v if and only if there is an edge between v and u, then G is called an undirected graph. Otherwise it is called a directed graph.

In *Mathematica*, a graph is represented either by a list of rules of the form {vi->vj,...}, where vi and vj are vertices, and vi->vj is the edge between vi and vj, or by the adjacency matrix of the graph.

```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1}]
```



You can write click on the graph to set the display properties
of the graph (layout, size, arrow shape, vertex shape, etc.)
You can also plot the graph by explicitly listed the display options.

```
GraphPlot[G, VertexLabeling -> True,
 DirectedEdges → {True, "ArrowheadsSize" -> 0.15},
 PlotStyle -> Dashed]
```

Adjacency matrix

The adjacency matrix of a graph G is built by the following rule : the (i, j) entry in the matrix is 1 if there is an edge between i and j in the graph G, and 0 otherwise.

```
AdjacencyMatrix[G]
```

```
SparseArray[ Specifiedelements 4
             Dimensions {4, 4} ]
```

By default, if the graph does not have a lot of edges, its adjacency matrix in *Mathematica* is given as a sparse array (only the non-zero elements are listed), but we can convert it to the regular form by using MatrixForm. In practice, for directed graph we will use more often the transpose of the adjacency matrix.

```
MatrixForm[AdjacencyMatrix[G]]
```

```
0 1 0 0
0 0 0 0
0 1 0 1
1 0 0 0
```

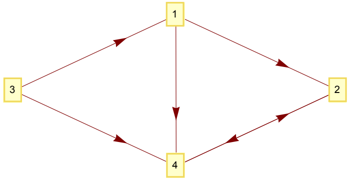
Random Graphs

We can construct a random graph using a random matrix with 0 and 1.

```
Ri = Table[RandomInteger[], {4}, {4}]
```

```
{{0, 1, 0, 1}, {0, 1, 0, 1}, {1, 0, 1, 1}, {0, 1, 0, 1}}
```

```
RG = GraphPlot[Ri, VertexLabeling → True, DirectedEdges → True]
```



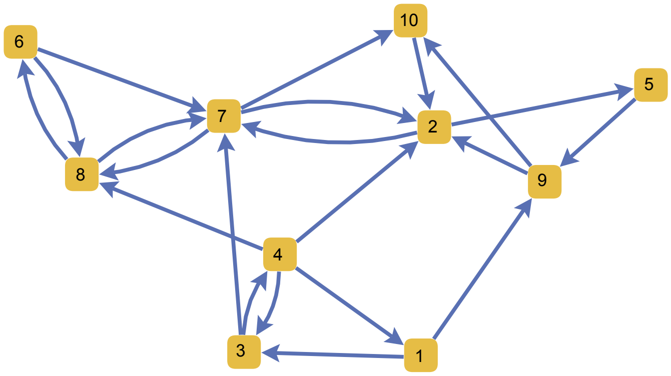
MAT 331: Homework 2

Problem 2.1

In this problem we will explore some of the basic definitions and *Mathematica* functions for graphs. A walk in a graph consists of an alternating sequence of vertices and edges $x_0, e_1, x_1, .., e_n, x_n$ such that $e_i = x_{i-1} \to x_i$ is an edge between vertex $x_{i-1}$and vertex x_i. A walk is closed if $x_0 = x_n$ and open otherwise. The number of edges is the length of a walk. The vertices and edges may be repeated in a walk. A path is (usually defined as) an open walk with no repeated vertices. A cycle is a closed walk with no repeated vertices, other than the first and last vertex. The problem of finding the shortest path between two vertices is very relevant to practical problems, such as finding the way out of a maze or navigating a road network. Cycles are also very important if traversing the whole graph in some way and coming back to the same place is desired instead.

Another relevant feature of graphs representing real systems is community structure, or clustering, i. e. the organization of vertices in clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters. Such clusters, or communities, can be considered as fairly independent compartments of a graph, playing a similar role like, e. g., the tissues or the organs in the human body. Detecting communities is of great importance in sociology, biology and computer science, disciplines where systems are often represented as graphs.

Consider the directed graph G depicted below (*you may need to click on Enable Dynamics first, to view the graph*):



1. Use the function **Graph[...]** to generate the graph G. The option GraphStyle→”SmallNetwork” will give the same display style as above, but you are free to explore additional attributes as well.
2. Find the strongly connected components of G. Use **HighlightGraph[...]** to color the vertices of the graph according to the component they belong to.
3. Use the function **FindGraphCommunities[...]** to find a list of communities in the graph G. This function returns a list of communities, where each community is a list of vertices. The communities are ordered by their length, with the largest community first. Use the command **HighlightGraph[...]** to color the vertices of the graph according to the community they belong to.

4. **ShortestPath[G, v1, v2]** returns a list of vertices representing the shortest path between nodes v1 and v2. Use this function to find the shortest path between node 6 and node 9 and then use **HighlightGraph[...]** to highlight the shortest path between these nodes. First highlight only the vertices, then highlight the edges of the shortest path. Use **GraphicsRow[...]** to print the two highlighted graphs on the same row.
5. **GraphDiameter[G]** gives the greatest distance between any pair of vertices in the graph G. What is the diameter of G? Explain.

Problem 2.2

Consider the graph G from problem 2.1. The command **FindCycle[G,{n}]** returns a cycle of length n in graph G. Use this function to find a cycle of length 3 in G. Suppose next that we want to find all cycles of length 3 in G. There are several ways to do that, but we will choose to work with the adjacency matrix of G for this task. Use your prefered loop structures in *Mathematica* (Do, While, For) to find how many cycles of length exactly 3 are in the graph G and to print all of these cycles. A cycle should be displayed as a list of edges of the form {a→b, b→c, c→a}. Do not print the same cycle three times! For example, {a→b, b→c, c→a}, {b→c, c→a, a→b}, {c→a, a→b, b→c} are different ways of writing the same cycle, which should be counted only once.

Problem 2.3

Write a function **transition[...]** that takes as input a directed graph H with any number of nodes, represented as a list of directed edges, and returns the transition matrix of H. Recall that the (i,j) entry of the transition matrix reflects the probability of transition from node i to node j. A preliminary version of the algorithm for computing the transition matrix of a graph is done is the lecture notes. There are some special cases which were not covered in the lecture (like the case of nodes with no outgoing edges). Test your function on the graph from problem 1, as well as on a randomly generated directed graph with 10 nodes and 20 edges.

Problem 2.4

Let H be any undirected graph. If H is a connected, acyclic graph (acyclic means that it contains no cycles) then we say that H is a tree. Trees are very useful for storing, sorting and quering information that has a natural hierarchical structure. For instance, the file system (folders, subfolders, documents, etc.) in a computer uses a tree architecture.

To build some intuition about trees, use **TreePlot[H, VertexLabeling→True, DirectedEdges→False]** to plot any graph which has the properties listed above. GraphPlot[...] can also be used for plotting trees, however the layout is slightly different.

Show that a graph H is a tree if and only if any one of the following properties is satisfied:

1. Every pair of distinct vertices of H is connected by a unique path.
2. The graph H is connected, but deleting any edge disconnects H.
3. The graph H is acyclic, but adding any edge to H forms a cycle.
4. The graph H is connected and has n-1 edges, where n is the number of vertices of H.

Problem 2.5

In this problem, we want to use the command **Manipulate[...]** to create nice interactive models. We can wrap Manipulate[...] around any Wolfram command. The output you get from evaluating a Manipulate command is an interactive object containing one or more controls (sliders, etc.) that you can use to vary the value of one or more parameters.

1. **RandomGraph[{m,n},k]** generates k random undirected graphs with m vertices and n edges; if we want to generate directed graphs, we need to set the attribute DirectedEdges to True, as shown in the lecture notes. Your first task is to create a nice interactive model that can generate k (directed or undirected!) graphs with *m* vertices and $\frac{m(m-1)}{2}$ edges. The possible values for k should be 1 and 4. The parameter m can be any positive integer between 4 and 10, with default value 6. You should have one extra parameter, called Directed, that can take the values True or False.
2. Let us now focus on part 4 of problem 2.1 and turn it into an interactive model as well. We do not want to change the graph G, but we would like to build an interactive model such that for any given choice of nodes v and u in G, we display the graph G with the shortest path between u and v highlighted. Use the command Manipulate[...] to create a nice interactive model with controls for the parameters u and v.

Graph Theory with *Mathematica*

Construction and representation
Visualization of graphs
Properties of graphs
Connectivity, strong connectivity
Adjacency & Transition Matrices
Making interactive models

Graphs

Definition: A graph G={V, E} consists of a set of vertices V (also called nodes) and a set of edges E.

Graphs provide a structural model that makes it possible to analyze and understand how many separate systems act together. Systems with symmetric relations are represented by undirected graphs, while the others can be modeled usind directed graphs.

Applications:

technological networks (the internet, power grids, telephone networks, transportation networks, …)

social networks (social graphs, affiliation networks, …)

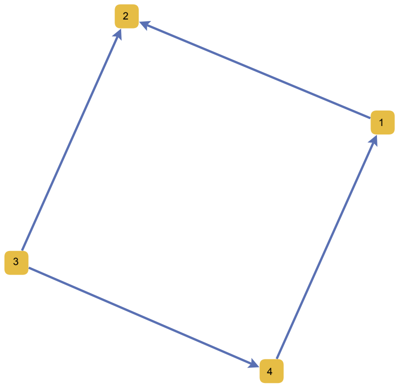
information networks (World Wide Web, citation graphs, patent networks, …),

biological networks (biochemical networks, neural networks, food webs, …), and many more.

In *Mathematica*, graphs are integrated in the Wolfram language, so they cand be used as input and output without importing any libraries. A graph is represented either by a list of rules of the form {vi->vj,...}, where vi and vj are vertices, and vi->vj is the edge between vi and vj, or by the adjacency matrix of the graph. In the *Mathematica* documentation (Graphs and Networks) you can find the graph representations, graph functions, algorithms and lots of examples.

Recall the graph definitions from last time

```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1}]
```



```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1}, GraphStyle → "SmallNetwork"]
```

You can write click on the graph to set the display properties

of the graph (layout, size, arrow shape, vertex shape, etc.)

You can also plot the graph by explicitly listed the display options.

```
GraphPlot[G, VertexLabeling -> True,
 DirectedEdges → {True, "ArrowheadsSize" -> 0.15},
 PlotStyle -> Dashed]
```

```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1}];
Gplot = GraphPlot[G, VertexLabeling -> True,
  DirectedEdges → {True, "ArrowheadsSize" -> 0.15},
  PlotLabel → "G is a graph with 4 vertices"]
```

Adjacency matrix

Definition: The adjacency matrix of a graph G is built by the following rule : the (i, j) entry in the matrix is 1 if there is an edge between i and j in the graph G, and 0 otherwise.

The adjacency matrix of the graph can be obtained using the function AdjacencyMatrix[...]

```
AdjacencyMatrix[G]
```

```
SparseArray[ ⊞ Specifiedelements 4
               Dimensions {4, 4} ]
```

By default, if the graph does not have a lot of edges, its adjacency matrix in *Mathematica* is given as a sparse array (only the non-zero elements are listed), but we can convert it to the regular form by using MatrixForm. In practice, for directed graph we will use more often the transpose of the adjacency matrix.

```
MatrixForm[AdjacencyMatrix[G]]
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Exercise : Define a graph H with the following set of edges and use Mathematica to compute its adjacency matrix.

Edges : 2 -> 3, 4 -> 5, 1 -> 6, 6 -> 2, 6 -> 5, 6 -> 1, 6 -> 3, 6 -> 4, 5 -> 1

Displaying graphics in the same row/column

We want to get a little fancy this time and display the graph and the adjacency matrix on the same row:

```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1}];
Gplot = GraphPlot[G, VertexLabeling -> True,
   DirectedEdges → {True, "ArrowheadsSize" -> 0.15},
   PlotLabel → "G is a graph with 4 vertices"];

Adj = MatrixForm[AdjacencyMatrix[G]];
GraphicsRow[ {Gplot, Adj}]
```

```
GraphicsRow[{Text["The graph"],
  Gplot, Text["has adjacency matrix"], Adj}]
```

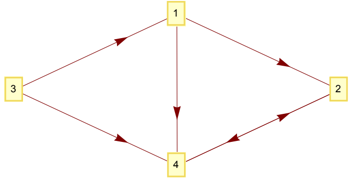
Random Graphs

We can construct a random graph using a random matrix with 0 and 1.

```
Ri = Table[RandomInteger[], {4}, {4}]
```

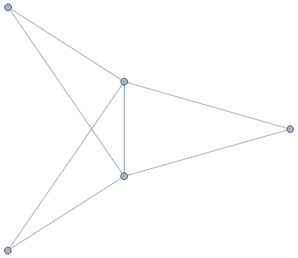
```
{{0, 1, 0, 1}, {0, 1, 0, 1}, {1, 0, 1, 1}, {0, 1, 0, 1}}
```

```
RG = GraphPlot[Ri,
  VertexLabeling → True, DirectedEdges → True]
```

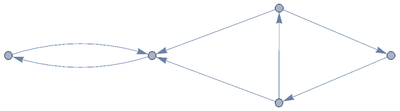


Mathematica also has a command for building random undirected graphs:
RandomGraph[{m,n}] -- where m is the number of vertices and n is the number of edges.
RandomGraph[{m,n},k] -- generates k random graphs with m vertices and n edges.
If we want to make the graph directed, we need to set the attribute DirectedEdges to True.

```
RandomGraph[{5, 7}]
```



```
RandomGraph[{5, 7}, DirectedEdges -> True]
```



Mathematica can also generate graphs with a special form, for instance graphs for which any two vertices are connected by an edge; these are called complete graphs).

```
CompleteGraph[5]
```

Properties of Graphs - Strong Connectivity

Definition: A graph is connected if for any two vertices u and v, there is a path in the graph from u to v OR from v to u. A graph is strongly connected if for any two vertices u and v, there is a path in the graph from u to v AND from v to u.

Proposition: Let G be a graph and A its adjacency matrix. If there is a positive integer k such that the matrix S = $I+A+A^2+A^3+\dots+A^k$ is positive (has only positive entries), then the graph is strongly connected.

Idea of the proof: The entry on row *i* and column *j* of the matrix A^k represents the number of paths of length k from i to j. If the (i,j) entry is positive, it means that there exists at least one path of length k from i to j.

Testing strong connectivity using the adjacency matrix and *Mathematica*:

First we compute S= $I+A+A^2+A^3+\dots+A^k$. It's enough to consider k=dimension(A)

```
S = IdentityMatrix[Length[Ri]] +
  Sum[MatrixPower[Ri, k], {k, 1, Length[Ri]} ]
```

```
{{1, 15, 0, 15}, {0, 16, 0, 15}, {4, 22, 5, 26}, {0, 15, 0, 16}}
```

Now we must check the entries of the matrix S to see if there are any 0 elements. If for some i and j the (i,j) entry is equal to 0, it means that there is no path from node i to node j, so the graph is not strongly connected.

```
con = 1;
Do[If[S[[i]][[j]] == 0, con = 0;
   Break[]], {i, 4}, {j, 4}];
If[con > 0, Print["Strongly connected"],
 Print["Not strongly connected"]]
```

```
Not strongly connected
```

```
con = 1;
Do[If[S[[i]][[j]] == 0, con = 0; Return[{i, j}]],
  {i, 4}, {j, 4}];
If[con > 0, Print["Strongly connected"],
 Row[{"Not strongly connected, for example, there is no
     path from node ", %[[1]], " to node ", %[[2]]}]]
```

```
Not strongly connected, for example, there is no path from node
 1 to node 3
```

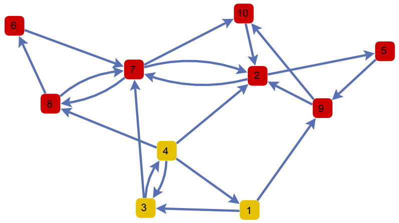
Comments about the previous code: First we assume that the graph is strongly connected, so we introduce a variable con and make it equal to 1. We use a Do loop to check whether there are any 0 entries in the matrix S. Notice the use of Return[{i,j}] in the Do loop. When the first 0 element of the matrix S is found, we set con to 0, exit the Do loop and return the pair {i,j}. We can later use {i,j} to indicate that the graph fails to be strongly connected because there is no path from node %[[1]] to node %[[2]]. If no element of the matrix is 0, then the value of the variable con is never changed, so it remains equal to 1 after the Do command.

Connected components of a graph

We can also use *Mathematica* to get the connected components of a graph:

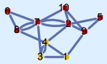
```
Gr = RandomGraph[{10, 20}, DirectedEdges → True];
ConnectedComponents[Gr]
HighlightGraph[Gr, ConnectedComponents[Gr]]
```

```
{{2, 5, 6, 7, 8, 9, 10}, {1, 3, 4}}
```



Mathematica lets you copy the graph and use it anywhere else in your code. You can even assign it to some other variable, or use it in computations.

```
Grrr =
```



```
MatrixForm[AdjacencyMatrix[Grrr]]
```

```
⎛0  0  1  0  0  0  0  0  1  0⎞
⎜0  0  0  0  1  0  1  0  0  0⎟
⎜0  0  0  1  0  0  1  0  0  0⎟
⎜1  1  1  0  0  0  0  1  0  0⎟
⎜0  0  0  0  0  0  0  0  1  0⎟
⎜0  0  0  0  0  0  1  0  0  0⎟
⎜0  1  0  0  0  0  0  1  0  1⎟
⎜0  0  0  0  0  1  1  0  0  0⎟
⎜0  1  0  0  0  0  0  0  0  1⎟
⎝0  1  0  0  0  0  0  0  0  0⎠
```

Transition Matrices

Definition: In a directed graph, for every vertex i there is a number of edges that enter that vertex (i is a head) and a number of edges that exit that vertex (i is a tail). Thus we define the indegree of vertex i as the number of edges for which vertex i is a head. Similarly, the outdegree of vetex i as the number of edges for which i is a tail.

Mathematica has built-in functions for the indegree and the outdegree of the vertices of a graph:
VertexInDegree[graph] -- returns a list with the indegrees of the vertices of the graph specified in the argument
VertexOutDegree[graph] -- returns a list with the indegrees of the vertices of the graph

Definition: The transition matrix A of a directed graph is defined as follows. If there is an edge from i to j and the outdegree of vertex i is d_i, then on column i and row j we put $\frac{1}{d_i}$. Otherwise we mark the entry on column i and row j with zero.

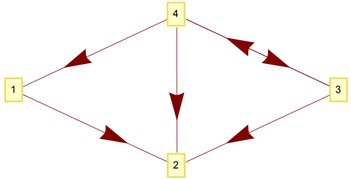
Random Walk on a Graph: We use the transition matrix to model the behavior of a random surfer on a graph. The surfer chooses a node at random, then walks on the outgoing edges to other nodes for as long as he/she wishes. At each step the probability that the surfer moves from node i to node j is zero if there is no link from i to j and $\frac{1}{d_i}$ otherwise. Recall that d_i is the outdegree of vertex i.

Question: What is the probability that a random surfer that starts at one of the nodes of the graph visits say, node j ?

Finding the Transition Matrix using *Mathematica*

Let us try to produce the transition matrix of a directed graph, using the adjacency matrix. Notice that for the transition matrix, we first look at the column, then at the row. So we need to work with the transpose of the adjacency matrix instead.

```
G = Graph[{1 -> 2, 3 → 2, 3 → 4, 4 → 1,  4 → 2,  4 → 3}];
Gplot = GraphPlot[G, VertexLabeling -> True,
  DirectedEdges → {True,  "ArrowheadsSize" -> 0.07}]
Lin = VertexInDegree[G];
Print["Lin=", Lin]
Lout = VertexOutDegree[G]; Print["Lout=", Lout]
```



```
Lin={1, 3, 1, 1}
Lout={1, 0, 2, 3}
```

```
A = AdjacencyMatrix[G];
TA  = Transpose[A];
Print["The transpose matrix is TA=" MatrixForm[TA]]
```

The transpose matrix is TA= $\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

```
f[i_, j_] := TA[[i]][[j]] / Lout[[j]]
```

```
Trans = Table[f[i, j], {i, Length[TA]}, {j, Length[TA]}];
MatrixForm[Trans]
```

$$\begin{pmatrix} 0 & \texttt{Indeterminate} & 0 & \frac{1}{3} \\ 1 & \texttt{Indeterminate} & \frac{1}{2} & \frac{1}{3} \\ 0 & \texttt{Indeterminate} & 0 & \frac{1}{3} \\ 0 & \texttt{Indeterminate} & \frac{1}{2} & 0 \end{pmatrix}$$

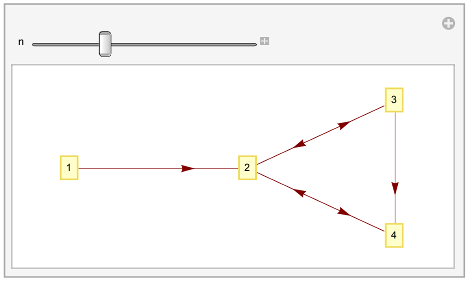
Exercise : Find the transition matrix of a graph. The code in yellow computes the transition matrix, but some cases are overlooked.

Making interactive models

Using the command Manipulate[...] we can turn a static computation or graph into a dynamic and sophisticated model. We can wrap Manipulate[...] around any Wolfram command to create an interactive model. We need to introduce a parameter that we want to manipulate and some bounds for that parameter.

Parameter ranges are given inside curly brackets:
{x, 4, 7} -- specifies an interval [4,7] that the parameter x belongs to.
{x, {4,7}} -- specifies a discrete set of values that x can take on.
{{x, 5, “Some text describing the meaning of the parameter x”}, 4,7}
-- x belongs to the interval [4,7] and the default value is 5

```
Manipulate[
 GraphPlot[Table[RandomInteger[],
   {n}, {n}], VertexLabeling → True,
  DirectedEdges → True],
 {n, 4, 7}]
```



If you click the plus sign at the end of the slider, you get a set of video controls. You can introduce a particular value of the parameter and hit Enter to jump to that value. When you do a presentation, you can also hide the Wolfram commands and show only the interactive model.

Help->Demonstrations gives you a list of pre-built Wolfram Demonstration Projects. You can browse by topic and see the available templates. You can then download the project as a .cdf, which means that you get an interactive version that you can run locally on your computer, or you can download the author code, which means that you get a notebook (.nb) with the necessary code to generate the model.

The Mathematics of Web Search

Random walk on a graph
Search Engine
Page Rank Algorithm
Perron-Frobenius Theorem
Power Method

Transition Matrices

Definition: In a directed graph, for every vertex i there is a number of edges that enter that vertex (i is a head) and a number of edges that exit that vertex (i is a tail). Thus we define the indegree of vertex i as the number of edges for which vertex i is a head. Similarly, the outdegree of vetex i as the number of edges for which i is a tail.

Mathematica has built-in functions for the indegree and the outdegree of the vertices of a graph:
VertexInDegree[graph] -- returns a list with the indegrees of the vertices of the graph specified in the argument
VertexOutDegree[graph] -- returns a list with the indegrees of the vertices of the graph

Definition: The transition matrix A of a directed graph is defined as follows. If there is an edge from i to j and the outdegree of vertex i is d_i, then on column i and row j we put $\frac{1}{d_i}$. Otherwise we mark the entry on column i and row j with 0.

Random Walk on a Graph: We use the transition matrix to model the behavior of a random surfer on a graph. The surfer chooses a node at random, then walks on the outgoing edges to other nodes for as long as he/she wishes. At each step the probability that the surfer moves from node i to node j is zero if there is no link from i to j and $\frac{1}{d_i}$otherwise. Recall that d_i is the outdegree of vertex i.

Question: What is the probability that a random surfer that starts at one of the nodes of the graph visits say, node j ?

Search Engine

- gather information from Web pages.

Crawlers crawl the Web following hyperlinks and they index the documents they find:

document → {list of words, links, etc}

- process and store information in a database

Indexer - computes the forward index

document → {number of occurances of each word}

Sorter - computes the inverted index

keyword → {doc1, doc2, ... , doc*n*}

- query the database in order to answer user's queries

Query engine

- uses the inverted index to compile a list of documents relevant to the keywords and phrases of the query.
- lists the documents found in decreasing order, according to their relevance ("best" documents first)

$\text{Relevance}_{\text{Query }Q}(\text{Page }\pi) = \text{I}(\pi, \text{Q}) \times \text{P}(\pi)$,

where $\text{I}(\pi, \text{Q})$ is a text-based ranking (on-page score of π for query Q) and

$\text{P}(\pi)$ is the quality factor of the page, or the page rank, independent of the textual content of the page.

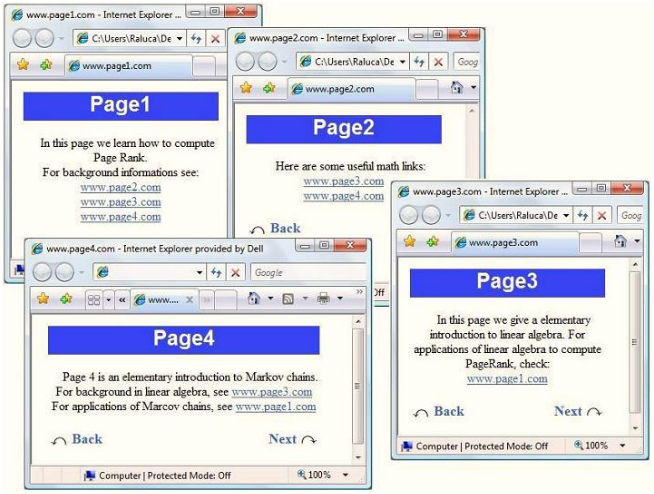
$\text{P}(\pi)$ is a number between 0 and 1, so multiplication by $\text{P}(\pi)$ acts as a damping on the document score of π.

Page Quality

- based on the reference(citation) ranking
- independent of the textual content of the page
- each page is assigned a quality value, based on the amount and quality of the citing pages.

The underlying assumption is that more important websites are likely to receive more links from other websites and therefore, the importance of any website can be judged by looking at the number and quality of the pages that link to it.

A Web citation is simply a link (page i "cites" page j is there is a link from page i to page j). In the next picture, Page 1 cites Pages 2, 3 and 4, Page 2 cites Pages 2 and 3, Page 3 cites Page 1, and finally Page 4 cites Pages 1 and 3.


www.page1.com - Internet Explorer ...
C:\Users\Raluca\De
www.page1.com
www.page2.com - Internet Explorer ...
C:\Users\Raluca\De
Goog
www.page2.com
www.page3.com - Internet Explorer ...
C:\Users\Raluca\De
Goog
www.page3.com
www.page4.com - Internet Explorer provided by Dell
Google
www....
Computer | Protected Mode: Off
100%

Page1

In this page we learn how to compute Page Rank.
For background informations see:
www.page2.com
www.page3.com
www.page4.com

Page2

Here are some useful math links:
www.page3.com
www.page4.com

Back

Page3

In this page we give a elementary introduction to linear algebra. For applications of linear algebra to compute PageRank, check:
www.page1.com

Back Next

Page4

Page 4 is an elementary introduction to Markov chains.
For background in linear algebra, see www.page3.com
For applications of Marcov chains, see www.page1.com

Back Next

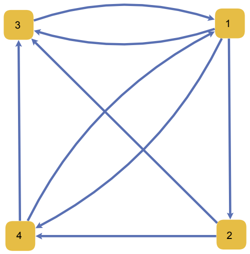
Web Graph

Nodes = Web sites
Edges = Hyperlinks between web sites

We can “translate” our baby Internet model with 4 pages into a directed graph with 4 nodes, one for each web site. When web site i references j, we add a directed edge between node i and node j in the graph. For the purpose of computing their page rank, we ignore any navigational links such as back, next buttons, as we only care about the connections between different web sites.

```
G = Graph[
  {1 → 2, 1 → 3, 1 → 4, 3 → 1, 2 → 3, 2 → 4, 4 → 1, 4 → 3},
      GraphStyle → "SmallNetwork", VertexSize → 0.15]
```

```
Lout = VertexOutDegree[G];
A = AdjacencyMatrix[G];
TA = Transpose[A];
Lout = VertexOutDegree[G];
f[i_, j_] := If[Lout[[j]] ≠ 0, TA[[i]][[j]] / Lout[[j]], 0]
T = Table[f[i, j], {i, Length[TA]}, {j, Length[TA]}];
GraphicsRow[{G, MatrixForm[T]}]
```



$$\begin{pmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

Page Rank Algorithm

Suppose that initially the importance is uniformly distributed among the 4 nodes, each getting ¼. Denote by v the initial rank vector, having all entries equal to ¼. Each incoming link increases the importance of a web page, so at step 1, we update the rank of each page by adding to the current value the importance of the incoming links. This is the same as multiplying the transition matrix T with v .

At step 0, the importance is uniformly distributed, v=(¼, ¼, ¼, ¼)
At step 1, the new importance vector is $v_1 = Tv$.
At step 2, the updated importance vector is $v_2 = Tv_1 = T(Tv) = T^2 v$
....
At step k, the updated importance vector is $v_k = T^k v$

The following code in Mathematica tests this convergence process:

```
v = {0.25, 0.25, 0.25, 0.25}
```

```
{0.25, 0.25, 0.25, 0.25}
```

```
For[k = 1, k ≤ 20, k++, Print[MatrixPower[T, k].v]]
```

```
{0.375, 0.0833333, 0.333333, 0.208333}
{0.4375, 0.125, 0.270833, 0.166667}
{0.354167, 0.145833, 0.291667, 0.208333}
{0.395833, 0.118056, 0.295139, 0.190972}
{0.390625, 0.131944, 0.286458, 0.190972}
{0.381944, 0.130208, 0.291667, 0.196181}
{0.389757, 0.127315, 0.290509, 0.192419}
{0.386719, 0.129919, 0.289786, 0.193576}
{0.386574, 0.128906, 0.290654, 0.193866}
{0.387587, 0.128858, 0.290244, 0.193311}
{0.3869, 0.129196, 0.29028, 0.193625}
{0.387092, 0.128967, 0.290377, 0.193564}
{0.387159, 0.129031, 0.290296, 0.193514}
{0.387053, 0.129053, 0.290325, 0.193568}
{0.38711, 0.129018, 0.290328, 0.193544}
{0.387101, 0.129037, 0.290318, 0.193545}
{0.38709, 0.129034, 0.290324, 0.193552}
{0.3871, 0.12903, 0.290323, 0.193547}
{0.387096, 0.129033, 0.290322, 0.193548}
{0.387096, 0.129032, 0.290323, 0.193549}
```

The sequence converges to a vector w ≃ {0.3870, 0.1290, 0.2903, 0.1935}. Note that this is only an approximation!

```
w = {0.3870, 0.1290, 0.2903, 0.1935};
Print["The sum of the entries of w is approximately ",
 w[[1]] + w[[2]] + w[[3]] + w[[4]]]
```

```
The sum of the entries of w is approximately 0.9998
```

Page Rank Algorithm

- The limit vector w must satisfy the equation Tw=w

If $v_k \to w$ as $k\to\infty$ then $Tv_k \to Tw$ as $k\to\infty$ (because the map $x\to Tx$ is continuous with respect to x).

However, $Tv_k = v_{k+1}$ and $v_{k+1} \to w$ as $k\to\infty$!

Therefore Tw=w, so w belongs to the eigenspace of T corresponding to the eigenvalue 1.

```
eigen = Eigensystem[T]
```

We do not have to compute all of the eigenvalues and eigenvectors of T, because only the eigenspace corresponding to the eigenvalue 1 is relevant for our convergence problem.

```
evector = eigen[[2]][[1]];
Print[
 "The eigenvectors corresponding to the eigenvalue 1
   are scalar multiples of the vector ", evector]
```

```
The eigenvectors corresponding to the
   eigenvalue 1 are scalar multiples of the vector {2, 2/3, 3/2, 1}
```

We normalize so that the sum of the elements is equal to 1.

```
w2 = 1 / (evector[[1]] + evector[[2]] +
      evector[[3]] + evector[[4]]) * evector
N[
 w2]
```

$$\left\{\frac{12}{31}, \frac{4}{31}, \frac{9}{31}, \frac{6}{31}\right\}$$

```
{0.387097, 0.129032, 0.290323, 0.193548}
```

This gives the exact value of the limit vector $\left\{\frac{12}{31}, \frac{4}{31}, \frac{9}{31}, \frac{6}{31}\right\}$.

Meaning of the Page Rank vector

The limit vector w is called the Page Rank vector of our graph. It provides a ranking system for the nodes of the graph. Each node is assigned an importance factor, based on the amount and relative importance of the nodes that link to it.

In our example, Node 1(Page 1) has page rank 0.38 so it is the most important page. Page 3 has a score of 0.29. Page 2 has importance factor 0.12, so it is the least important, and Page 4 has importance factor 0.19.

Random Surfer Model: The page rank of Page i represents the probability that a random surfer on the Internet that opens a browser to any page and starts following hyperlinks, visits Page i.

Perron-Frobenius Theorem

If M is a positive, column stochastic nxn matrix, then the following statements are true:

1. The number 1 is an eigenvalue of M, of multiplicity one (that is, if u and v are two eigenvectors corresponding to the eigenvalue 1, then u is a scalar multiple of v).
2. The eigenvalue 1 is the largest eigenvalue of M; all the other eigenvalues of M are strictly less than 1 in absolute value.
3. Any eigenvector corresponding to 1 has either positive or negative entries.
4. There exists a unique probabilistic eigenvector w corresponding to the eigenvalue 1.

Recall the following definitions:

A square matrix is called *column stochastic* if all its entries are greater than or equal to 0, and the sum of the entries in each column is 1.

A matrix is called *positive* if all of its entries are strictly greater than 0.

A vector is called *probabilistic* (or a *probability distribution vector*) if all its entries are greater than or equal to 0 and the sum of all entries is 1.

Power Method Convergence Theorem

Let M be a positive, column stochastic nxn matrix. Denote by w its unique probabilistic eigenvector corresponding to the eigenvalue 1.
Let v be the column vector with all entries equal to 1/n . Then the sequence v, Mv, ... , M^{k}v converges to the vector w as k goes to ∞.
Let z be any probabilistic vector of size n. Then the sequence z, Mz, ... , M^{k}z converges to the vector w as k goes to ∞.

Sketch of the proof

Proposition 1: If v is a probabilistic vector and M is a positive and column stochastic matrix, then Mv is a probabilistic vector with only positive entries.

idea of proof: If v is a probabilistic vector of dimension n, and M is a column stochastic matrix, then Mv is also a probabilistic vector.
If v is a non-negative vector and M is a positive matrix, then Mv has only positive entries.

Definition: Let u and v be two vectors of size n. We define the distance between u and v to be the non-negative real number $d(u, v) = \frac{1}{2} \sum_{i=1}^{n} | u_i - v_i|$. Here u_i and v_i denote the i-th entries in the vectors u, and respectively v.
Definition: Consider now a sequence of vectors v_k , k>0, of size n. We say that the sequence of vectors v_k converges to w as k tends to ∞, if $d(v_k, w) \to 0$ as $k \to \infty$.

Proposition 2 (Contraction): Let M be a positive and column stochastic nxn matrix. There exists a number r, $0<r<1$, such that if v and u are two probabilistic vectors of size n, then $d(Mv, Mu) \leqslant (1-r)\, d(v,u)$. Therefore the distance between u and v decreases after left multiplication by M.

Proposition 3: Let M be a positive and column stochastic nxn matrix and v any probabilistic vector of size n. Denote by $v_k = M^K v$. Then the sequence v_k converges exponentially fast to a unique limit vector w.

idea of proof: Banach Fixed point Theorem!
Let j and k be any two positive integer. We can estimate the distance between any two terms of the sequence v_j and v_{j+k} in the following way:

$$d(v_{j+k}, v_j) = d(M^{j+k} v, M^j v) \leqslant (1-r)^{j} d(M^k v, v) \leqslant (1-r)^{j} d(v_k, v) \leqslant (1-r)^{j} \to 0$$

as $j \to \infty$, because 1-r is strictly less than 1.
Therefore the sequence $\{v_k\}_{k \geqslant 0}$ is a Cauchy sequence in $\mathbb{R}^n$, so there exists a vector w in $\mathbb{R}^n$ such that $v_k \to w$ as $k \to \infty$. The limit w is unique! Suppose by contradiction, that there would be (at least) two limit vectors, w and z. Then both w and z must satisfy Mw=w and Mz=z. Then we must also have the inequalities $0 \leqslant d(w, z) = d(Mw, Mz) \leqslant (1-r) d(w,z) < d(w,z)$, because 1-r is strictly less than 1. We reached a constradiction, $d(w, z)<d(w, z)$, which shows that our initial assumption was false and w and z must be equal.

Making interactive models

Manipulate[...]

Using the command Manipulate[...] we can turn a static computation or graph into a dynamic and sophisticated model. We can wrap Manipulate[...] around any Wolfram command to create an interactive model. We need to introduce a parameter that we want to manipulate and some bounds for that parameter.

Parameter ranges are given inside curly brackets:
{x, 4, 7}   -- specifies an interval [4,7] that the parameter x belongs to.
{x, {4,7}} -- specifies a discrete set of values that x can take on.
{{x, 5, "Some text describing the meaning of the parameter x"}, 4,7} -- x belongs to the interval [4,7] and the default value is 5

Help->Demonstrations gives you a list of pre-built **Wolfram Demonstration Projects**. They all use the command Manipulate[...]. You can browse by topic and see the available templates. You can then download the project as a .cdf, which means that you get an interactive version that you can run locally on your computer, or you can download the author code, which means that you get a notebook (.nb) with the necessary code to generate the model.

■ Example 1. Make an interactive model that computes n^2, for n in the interval [1,4].

☺ Solution

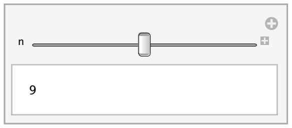
```
In[1]:= Manipulate[n^2, {n, 1, 5}]
```

If you click the plus sign at the end of the slider, you get a set of video controls. You can introduce a particular value of the parameter and hit Enter to jump to that value. When you do a presentation, you can also hide the Wolfram commands and show only the interactive model.

We can also set a default value for n when we call Manipulate[..].

```
In[2]:= Manipulate[n^2, {{n, 3}, 1, 5}]
```

Out[2]=



```
9
```

☺ We can indicate the domain of n in different ways!

In the following example, n takes values between 1 and 4 in steps of 0.25 and respectively in steps of 1. The Manipulate command creates a slider.

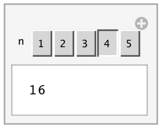
```
Manipulate[n^2, {n, 1, 5, 0.25}]
```

```
Manipulate[n^2, {n, 1, 5, 1}]
```

Instead of incrementing n in steps of 1, we can enumerate the list of possible values for n. The Manipulate command will create a button for each possible value.

```
In[3]:= Manipulate[n^2, {n, {1, 2, 3, 4, 5}}]
```

Out[3]=

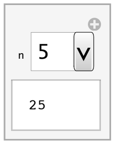


```
16
```

If the list of possible values for n is too large, then a drop - down list will be created.

```
In[4]:= Manipulate[n^2, {n, {1, 2, 3, 4, 5, 6, 7, 8, 9}}]
```

Out[4]=



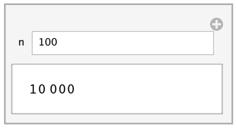
```
25
```

If no interval for n is given, Manipulate will create an empty text box where you can type in the value for n, then click Enter to obtain the result .

In[5]:=

```
Manipulate[n^2, {n}]
```

Out[5]=



n 100

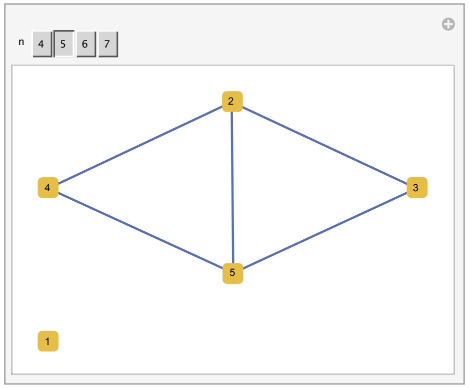
10 000

- Example 2. Make an interactive model that draws a random directed graph with n vertices, for n between 4 and 7.

In[6]:=

```
Manipulate[
 RandomGraph[{n, 5}, GraphStyle → "SmallNetwork", ImageSize → Medium],
 {n, {4, 5, 6, 7}}]
```

Out[6]=



If we do not want the graphics to rescale every time we generate a new graph, we can set the dimensions of the graphical object ImageSize -> {1000, 300}. Or we can set the dimensions of the window diplayed by Manipulate[..] using ContentSize -> {1000, 300}.

In[7]:=

```
Manipulate[
 GraphPlot[Table[RandomInteger[], {n}, {n}],
  VertexLabeling → True, DirectedEdges → True, ImageSize → {1000, 300}],
 {n,
  4,
  7}]
```

```
Manipulate[
 RandomGraph[{n, 5}, GraphStyle → "SmallNetwork", ImageSize → Medium],
 {n, {4, 5, 6, 7}},
 ContentSize → {1000, 500}]
```

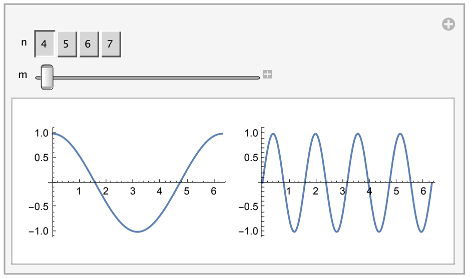
How to manipulate several functions of several parameters

- Example 3. Make an interactive model that draws the graphs of the functions cos(mx) and sin(nx), where m is a continuous parameter in the interval [1,2], and n a discrete parameter in the set {4,5,6,7}.

In[9]:=

```
Manipulate[
 GraphicsRow[{
   Plot[Cos[m * x], {x, 0, 2 * Pi}],
   Plot[Sin[n * x], {x, 0, 2 * Pi}] }],
 {n, {4, 5, 6, 7}},
 {m, 1, 2}
]
```

Out[9]=



It is possible to have correlated sliders.

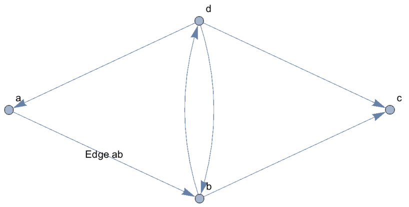
```
Manipulate[
 Column[{"Graphs of Cos(mx) and Sin(nx)",
   Plot[Cos[m * x], {x, 0, 2 * Pi}],
   Plot[Sin[n * x], {x, 0, 2 * Pi}] }],
 {n, m, 2 * m},
 {m, 1, 2}
]
```

Recall :

Recall that we can define a graph by specifying the list of vertices and the list of edges and other options.The generic command is **Graph[VertexList, EdgeList, Options]**. Sometimes it is desired to label the nodes of the graph using something other than numbers.

In[14]:=
```
G = Graph[{1, 2, 3, 4}, {1 → 2, 2 → 3, 2 → 4, 4 → 3, 4 → 1, 4 → 2},
  VertexLabels → {1 → "a", 2 → "b", 3 → "c", 4 → "d"},
  EdgeLabels → {(1 → 2) -> "Edge ab"}]
```

Out[14]=



VertexList[..] and **EdgeList[..]** give the list of vertices and respectively the list of edges of a graph. **PropertyValue[..]** can be used to retrieve the list of labels.

In[15]:=
```
VertexList[G]
EdgeList[G]
```

```
Out[15]= {1, 2, 3, 4}

Out[16]= {1 ↦ 2, 2 ↦ 3, 2 ↦ 4, 4 ↦ 3, 4 ↦ 1, 4 ↦ 2}
```

In[17]:=
```
PropertyValue[G, VertexLabels]
PropertyValue[G, EdgeLabels]
```

```
Out[17]= {3 → c, 4 → d, 2 → b, 1 → a}

Out[18]= {1 ↦ 2 → Edge ab}
```

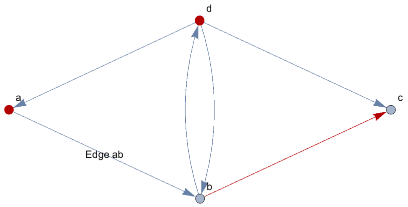
If the ordered list of labels is desired, one can use **Sort[..]**.

In[19]:=
```
L = PropertyValue[G, VertexLabels];
Sort[L, Greater]
Sort[L]
```

```
Out[20]= {3 → c, 4 → d, 2 → b, 1 → a}

Out[21]= {1 → a, 2 → b, 3 → c, 4 → d}
```

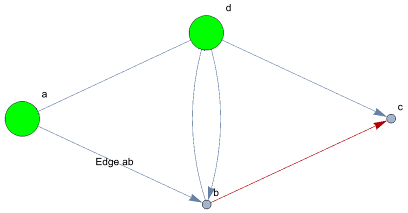
In[22]:=
```
HighlightGraph[G, {1, 4, 2 → 3}]
```

Out[22]=



In[23]:=
```
HighlightGraph[G, {Style[{1, 4}, Green], 2 → 3}, VertexSize → Medium]
```

Out[23]=



Exercise

Open the *Mathematica* notebook Map.nb (MetroStationMap).The variable called **Metro** is a list of 5 graphs with the metro networks in Seoul, New York, Paris, Mexico City and London. The vertices of the graphs are labeled with the names of the metro stations.

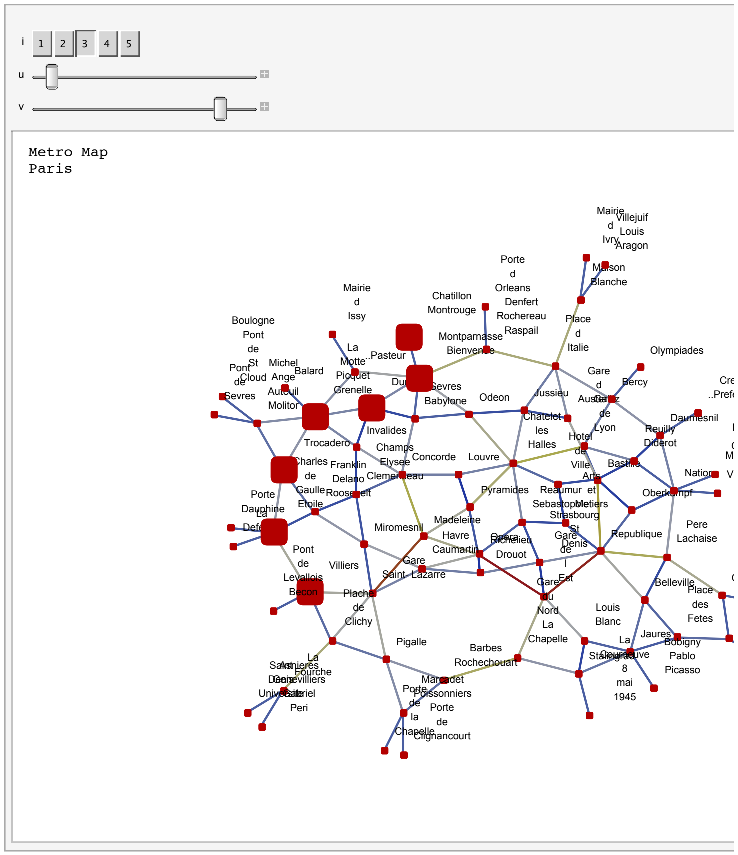
- **(a).** Write an interactive program using **Manipulate[...]** that has a control button with five values 1,2,3,4,5. When the user selects a given value, you display the name of the city and the corresponding metro map.
- **(b).** Expand the model from part (a) with a second control which gets populated with the vertices of the corresponding graph. When the user selects a vertex from the list, you show the corresponding metro station highlighted in green (or with increased size) on the metro network.
- **(c).** Make an interactive model from part (b) with nodes v an u as parameters, that highlights the shortest path from v to u, if there exists such a path in the graph, otherwise it only highlights the two vertices.
- **(d).** Make an interactive model that highlights a vertex v of G2 together with its parents and children (i.e. if there is an edge v->u, then v is called a parent node and u a child node). The function NeighborhoodGraph[...] might be of help.

- **(e). Optional (Combined Interactive Model):** Parameters: nodes v and u, and an extra parameter “option” with values {b,c,d}. If “option” b is selected, then the model outputs the same thing as part (b), If “option” c is selected, then the model outputs the same thing as part (c) etc. If you skiped some part, say (d), then you should allow only two values for “option”, {b,c}.

Partial Answer:

```
NameCity = {"Seoul", "New York", "Paris", "Mexico City", "London"};
```

```
Manipulate[
 Column[{"Metro Map ", NameCity[[i]], HighlightGraph[Metro[[i]],
    FindShortestPath[Metro[[i]], u, v], VertexSize → 1.5]}],
 {i, {1, 2, 3, 4, 5}}, {{u, 1}, VertexList[Metro[[i]] ]},
 {{v, 1}, VertexList[Metro[[i]]]}]
```



```
Metro Map
Paris
```

Lab project - Making Interactive Models

Example 1. Make an interactive model that computes n^2, for n in the interval [1,4].

☺ Solution 1

```
Manipulate[n^2, {n, 1, 4}]
```

Once an interactive model is created, you can copy and run the model in a new notebook. You do not need to copy the code.

We can indicate the domain of n in different ways (each way will generate a different type of control button, slider, popup menu, checkbox, etc.)

```
Manipulate[n^2, {n, 1, 4, 0.25}]
```

```
Manipulate[n^2, {n, 1, 4, 1}]
```

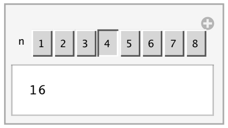
```
Manipulate[n^2, {n, {1, 2, 3, 4}}]
```

Options for the controls can also be given within the specification for the variables.
The option setting **ControlType ->** type attempts to use controls of the specified type. Possible control types include: Animator, Checkbox, CheckboxBar, ColorSetter, ColorSlider, InputField, Manipulator, PopupMenu, RadioButton or RadioButtonBar, Setter or SetterBar, Slider, Slider2D, TogglerBar, Trigger, and VerticalSlider, None. More information can be found in the Mathematica Documentation.
The option setting **ControlPlacement ->** pos specifies that controls should be placed at position pos relative to expr. Possible settings for position are Bottom, Left, Right, and Top.

In[24]:=

```
Manipulate[n^2, {n, {1, 2, 3, 4, 5, 6, 7, 8}, ControlType → SetterBar}]
```

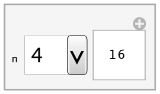
Out[24]=



In[25]:=

```
Manipulate[n^2,
 {n, {1, 2, 3, 4, 5, 6, 7, 8}, ControlType -> PopupMenu, ControlPlacement → Left}
]
```

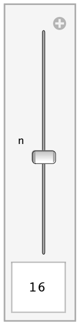
Out[25]=



In[26]:=

```
Manipulate[n^2, {n, {1, 2, 3, 4, 5, 6, 7, 8}, ControlType → VerticalSlider}]
```

Out[26]=



☹ Solution 2

```
Manipulate[
 temp = n;
 temp = temp^2;
 temp,
 {n, 1, 4}
]
```

Notice that the cell reevaluates itself continuously (the right cell bracket is contantly blinking), even when we do not change the position of the slider. You can confirm this by going to Evaluation->Find Currently Evaluating Cell. This happens because the variable temp has its value changed during the evaluation (temp =temp^2), even if the value of n has not changed.

☺ Solution 3

The problem can be solved by making the global variable “temp” be local variable inside a **Module**. Nothing you do to local **Module** variables will cause reevaluating, because it is part of the definition of Module that values of local variables do not survive from one invocation to the next.

```
Manipulate[Module[{temp},
   temp = n;
   temp = temp^3;
   temp],
  {n, 1, 4}
]
```

☹ Solution 4

```
Manipulate[
  f[x_] := x^2;
  f[n],
  {n, 1, 4}]
```

☺ Solution 5

```
Manipulate[Module[{g},
   g[x_] := x^2;
   g[n]],
  {n, 1, 4}]
```

The function g is now a local variable, so it does not cause any extra reevaluations. Notice that it is not defined outside of Module[...], so if you try to call g[3] say, somewhere below, the function g will not be recognized.

```
g[3]
```

☺ Solution 6

```
Manipulate[
  f[x_] := x^2;
  f[n],
  {n, 1, 4}, TrackedSymbols ⧴ {n}]
```

We can keep the function f global, without causing any reevaluations of Manipulate, if we explicitly indicate that the only variable whose values we should keep track of is the parameter n (by default, Manipulate[...] tracks both f and n). The example above reevaluates only when n changes its value as a result of moving the slider.

☹ Solution 7

```
h[x_] := x^2
Manipulate[
  h[n],
  {n, 1, 4}]
```

We can also define the function h(x)=x^2 globally, before Manipulate[..]. At first glance, it looks like everything works well and without causing any reevaluations of Manipulate.

The downfall is that the definition of the function h (which is called later in the body of Manipulate) is not saved together with the output of Manipulate. To see this, open a new *Mathematica* notebook, and set “Evaluation->Notebook’s Default Context” to “Unique to this notebook”. Then copy ONLY the ouput of Manipulate (that is, ONLY the interactive model, absolutely NO code) in the new notebook. Try changing the slider. What do you notice?

☺ Solution 8

```
h[x_] := x^2
Manipulate[
 h[n],
 {n, 1, 4}, SaveDefinitions → True]
```

The option “**SaveDefinitions→True**” forces any function definitions used by Manipulate to be saved with the output. The output can be copied and directly run in a new notebook. Try it!

■ Exercise 1. Make an interactive model that reads the value of two variables, total and sum, represented by two different controls, and outputs their sum, total+sum. Does the example below do that?

```
Manipulate[
  total = total + step;
 {step, total},
 {{total, 0}, -1000, 1000, 1}, {{step, 0}, -10, 10, 1},
 FrameLabel → "Good or bad?"]
```

■ Exercise 2. A way to compute the sum of the first m positive integers in *Mathematica* is to do a For loop, like the one written below. Turn the code below into an interactive model where m will be allowed to take any values in the set {10, 20, 30, 40, 50}.

```
m = 5;
s = 0;
For[i = 1, i ≤ m, i++, s = s + i];
s
```

■ Exercise 3. Interactive models with graphs. This is the exercise from last time, plus some additional tasks.

Open the *Mathematica* notebook MetroStationMap.The variable called **Metro** is a list of 5 graphs with the metro networks in Seoul, New York, Paris, Mexico City and London. The vertices of the graphs are labeled with the names of the stations.

```
NameCity = {"Seoul", "New York", "Paris", "Mexico City", "London"}
```

- **(a).** Write an interactive program using **Manipulate[...]** that has a control button with five values 1,2,3,4,5. When the user selects a given value, you display the name of the city and the corresponding metro map. Because your program uses some global variables defined outside Manipulate (like **Metro**), you should also set **SaveDefinitions→True**.
- **(b).** Expand the model from part (a) with a second control which gets populated with the vertices of the corresponding graph. When the user selects a vertex from the list, you show the corresponding metro station highlighted in green (or with increased size) on the metro network. Make the control type a **PopupMenu**.
- **(c).** Make an interactive model from part (b) with nodes v an u as parameters, that highlights the shortest path from v to u, if there exists such a path in the graph, otherwise it only highlights the two vertices. Use the function **FindShortestPath[...].**
- **(d).** Make an interactive model that highlights a vertex v together with its parents and children (i.e. if there is an edge v->u, then v is called a parent node and u a child node). The function NeighborhoodGraph[...] might be of help.
- **(e).** A Metro hub is a metro station which has direct connections to five or more other metro stations. Make an interactive model where the metro hubs are highlighted with VertexSize->**1.2**
  Use for example **VertexInDegree[..]** to count the number of incoming edges for each node. You may also need to use the function **Sort[..]** (check also the Mathematica documentation on Sort).
- **(f). Optional: Combined Interactive Model:** Parameters: nodes v and u, and an extra parameter “option” with values {b,c,d,e}. If “option” b is selected, then the model outputs the same thing as part (b), If “option” c is selected, then the model outputs the same thing as part (c) etc. If you skiped some part, say (d), then you should allow only two values for “option”, {b,c,e}. You may need to use some conditional statements like **If** or **Switch**.

MAT 331: Project 1

Project description:

The PageRank algorithm provides the basis for Google's web search tools. As we described in the lecture, PageRank is a recursive algorithm, and it works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites and therefore, the importance of any website can be judged by looking at the number and quality of the pages that link to it. Components of the PageRank vector serve as "importance" weights for web pages, independent of their textual content, solely based on the hyperlink structure of the web.

In this project we are interested in the theoretical foundations of the PageRank algorithm, in computing the Page Rank of small-scale graphs and in the effects of particular aspects of web graph structure on the Page Rank vector.

Tasks:

1. Browse the web and read a little bit about the importance of the PageRank algorithm for web searches. Ranking algorithms such as PageRank have also found applications in other areas (population models, article citations, hotel ranking systems, etc.). Give examples of the applications of Page Rank to search engines and to other real life problems. Please include all relevant references.
2. Describe how the Page Rank algorithm works. Investigate possible ways to compute the Page Rank vector of a graph (that is, computing eigenvalues, or rather applying the Power Method) and describe their advantages and drawbacks as the size of the graph gets bigger and bigger.
3. Explain the mathematics behind PageRank. The algorithm can be explained using ideas from several fields of Mathematics, like probability, linear algebra, dynamical systems or analysis. You can choose the mathematical language that you feel more confortable with. Feel free to use external sources and articles and include references for any relevant theorems and proofs that you may decide to use. If you decide to follow the steps explained in the lecture notes, then take a look at the lemmas listed at the end and fill in the proofs.
4. You will next build an interactive model in *Mathematica* for calculating the Page Rank vector of small scale web graphs, with nodes representing websites and edges representing hyperlinks. The number of nodes is assumed reasonably small, so that your algorithm works relatively fast. Your model should work with random graphs, graphs with dangling nodes, as well as some fixed test graphs, given below. The basic model should display the graph with the first five most important nodes highlighted in different colors and the (modified?) transition matrix A of the graph. The extended model should include a step-by step computation of the page-rank vector (at each step, one computes $A^k v$ and highlights the most important 5 vertices according to the page-rank approximation $A^k v$). There are several aspects that your interactive model should take into account:
   - You should include controls that enable the user to select the desired type of graph and the number of nodes.
   - The damping factor p should be given as a parameter, with values between 0.01 and 0.5.
   - The power k should also be given as a parameter, with values between 1 and 20, and with step size 1.
5. Give some brief instructions on how your model should be tested (how to use the controls, etc.).

6. Conclusions. Formulate your findings after testing your interactive model on a couple of examples. Is the Page Rank vector in agreement with your natural understanting of the relative importance of nodes? What values of the damping factor seem to give more accurate rankings? You can also copy paste in here any interesting examples that you may have discovered when testing your model.
7. References. Include all relevant references here.

General considerations:

You wil be graded on the quality and maturity of your mathematical explanations, as well as on the efficiency of the *Mathematica* code that you develop. Your interactive model should reevaluate itself only when the controls are changed (i.e. we change the position of a slider, etc.). When possible, try to avoid performing unnecessary computations (for example, do not use Eigensystem[A] if you are only interested in the eigenspace corresponding to the eigenvalue 1; you may choose not to use MatrixPower[A,k] if you have to compute A^k for many succesive values of k; etc.) Try to have a nice coding style and use functions when appropriate. Use different names for your variables! Do not include all intermediate output that you may have obtained by testing small parts of your code. All your code, including functions/variable definitions and Manipulate[..] should be contained in one single *Mathematica* cell bracket. Before you start working on your project, look also through the Wolfram Demonstration Projects, to get an idea of how a project should look like.

Mathematics:

When explaining the Mathematics behind the Page Rank Algorithm, if you decide to follow the plan outlined in the lectures, don’t forget to give justifications for the following statements:

0.1. If A is a column stochastic nxn matrix and v$\in\mathbb{R}^n$ is a probabilistic vector, then Av is a probabilistic vector.

- *Hint: By the definition of matrix multiplication, the i-th entry of the vector Av is* $(Av)_i = \sum_{j=1}^{n} a_{ij}\, v_j$.

0.2. If A is a positive nxn matrix and v$\in\mathbb{R}^n$ a probabilistic vector with non-negative entries, then Av is a vector with all entries positive.

0.3. If A is a column stochastic nxn matrix, B is the nxn matrix with all entries equal to 1 and p$\in$(0,1], then the matrix (1-p)A + p$\frac{1}{n}$B is positive and column stochastic. The number p is called the damping factor.

0.4. Let u and v be two vectors in $\mathbb{R}^n$. Define the distance between u and v to be

$d(u, v) = \frac{1}{2} \sum_{i=1}^{n} | u_i - v_i|$.

Show first that d verifies the three properties of a distance (metric):

- d (u, v) = d (v, u)
- d(u, v) ⩽ d(u, w) + d(w, v) where u,v,w are any three vectors in $\mathbb{R}^n$.
- d(u, v) ⩾ 0 for any two vectors u,v and d(u, v) = 0 if and only if u = v.

0.5. Next assume that u and v are two probabilistic vectors in $\mathbb{R}^n$. Show that the distance function is bounded

- d(u, v) ⩽ 1.

Assume that u and v are two probabilistic vectors in $\mathbb{R}^n$. Let S = $\{1\leqslant i\leqslant n,\ u_i > v_i\}$ be the set of indices i for which the i-th entry of u is stricly greater than the j-th entry of v.

- Show that S is a proper subset of {1,2,...,n}, that is, S is nonempty and S≠{1,2,...,n}.
- Show that d(u, v)= $\sum_{i\in S} u_i - v_i$ and that d(u, v)<1.

0.6. Let A be a positive column stochastic nxn matrix. Show that there exists a real number r, with 0<r<1, such that d(Au, Av) ⩽ (1-r) d(u, v), for any two probabilistic vectors u and v in $\mathbb{R}^n$.

- *Hint: Consider the sets S={1⩽i⩽n, u_i> v_i} and Q={1⩽i⩽n, $(Au)_i$> $(Av)_i$}. Use **Lemmas 0.1** and **0.4** to show that these are proper subsets of {1,2,...,n}. Next show that there exists a number r with 0<r<1 such that for any fixed column j of the matrix A, we have the inequality $\sum_{i\in Q} a_{ij}$ < 1-r. The number r can be chosen so that 0<r<min $_{0\leqslant i,j\leqslant n}\{a_{ij}\}$. Then use the definition of matrix multiplication and **Lemma 0.4**.*

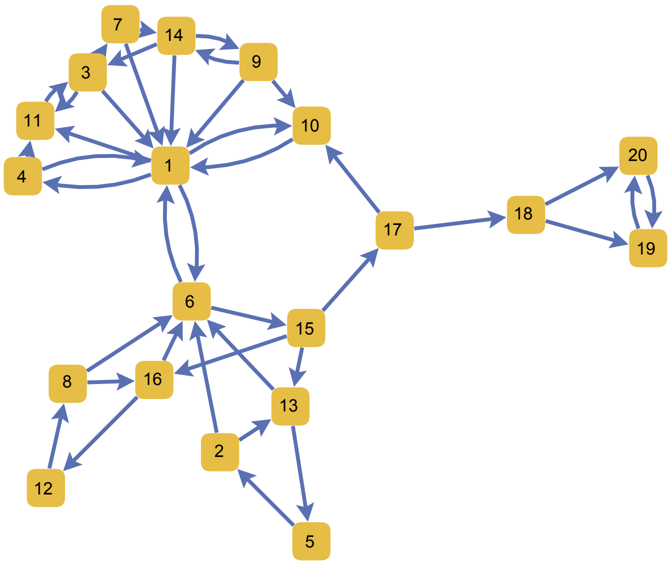
0.7. Use **Lemma 0.5** to show that if A is a positive column stochastic matrix, then there exists a unique probabilistic eigenvector corresponding to the eigenvalue 1.

0.8. Let A be a positive column stochastic nxn matrix. Let v be a probabilistic vector of size n. Use **Lemma 0.5** and **Lemma 0.6** and the ideas discussed in the lecture to show that the sequence *v*, Av, A^2 *v*... , A^k *v* converges to the unique probabilistic eigenvector corresponding to the eigenvalue 1.

Some test graphs:

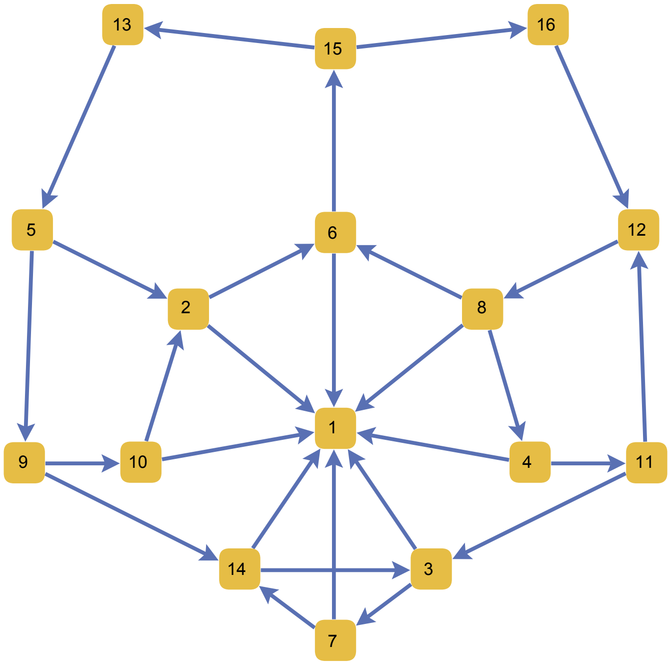
In[1]:=
```
G1 = Graph[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20},
   {1 → 4,  1 → 6,  1 → 10, 1 → 11,  2 → 6,  2 → 13,  3 → 1, 3 → 7, 3 → 11,  4 → 1,  4 → 11,
    5 → 2,  6 → 1, 6 → 15, 7 → 1,  7 → 14,   8 → 6, 8 → 16,  9 → 1, 9 → 10, 9 → 14,  10 → 1,
    11 → 3,  12 → 8,  13 → 5, 13 →  6,  14 → 1, 14 → 3,  14 → 9, 15 → 13,  15 → 16,
    16 → 12,  16 → 6,  15 → 17, 17 → 10, 17 → 18, 18 → 19, 18 → 20, 19 → 20, 20 → 19},
  GraphStyle → "SmallNetwork", GraphLayout → "SpringEmbedding"]
```

Out[1]=



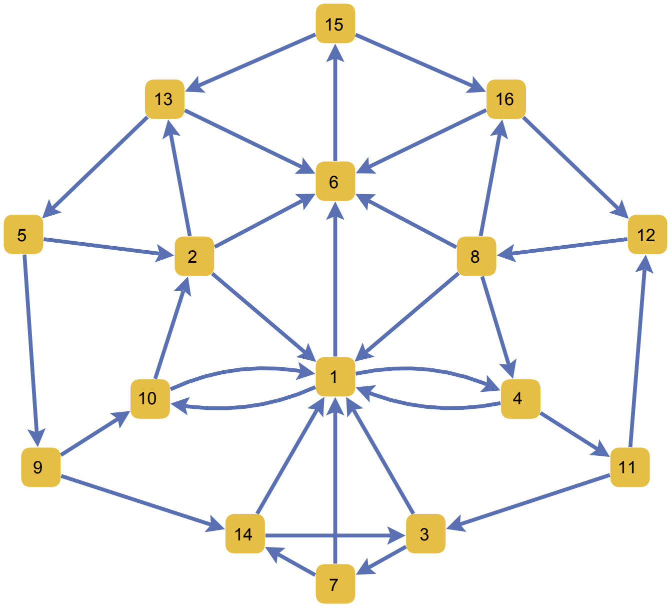
```
In[2]:= G2 = Graph[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16},
          {2 → 1,  2 → 6,  3 → 1, 3 → 7,  4 → 1,  4 → 11,  5 → 2,  5 → 9,  6 → 1, 6 → 15,
           7 → 1,  7 → 14,  8 → 1,  8 → 4,  8 → 6,  9 → 10, 9 → 14,  10 → 2, 10 → 1, 11 → 12,
           11 → 3,  12 → 8,  13 → 5, 14 → 1, 14 → 3,  15 → 13,  15 → 16, 16 → 12},
          GraphStyle → "SmallNetwork", GraphLayout → "SpringEmbedding"]
```

Out[2]=

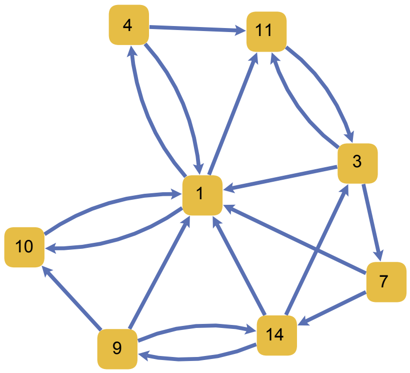


In[3]:=
```
G3 = Graph[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16},
  {1 → 4, 1 → 10, 1 → 6, 2 → 1, 2 → 6, 2 → 13, 3 → 1, 3 → 7, 4 → 1,
   4 → 11, 5 → 2, 5 → 9, 6 → 15, 7 → 1, 7 → 14, 8 → 1, 8 → 4, 8 → 6,
   8 → 16, 9 → 10, 9 → 14, 10 → 2, 10 → 1, 11 → 12, 11 → 3, 12 → 8,
   13 → 5, 13 → 6, 14 → 1, 14 → 3, 15 → 13, 15 → 16, 16 → 12, 16 → 6 },
  GraphStyle → "SmallNetwork", GraphLayout → "SpringEmbedding"]
```

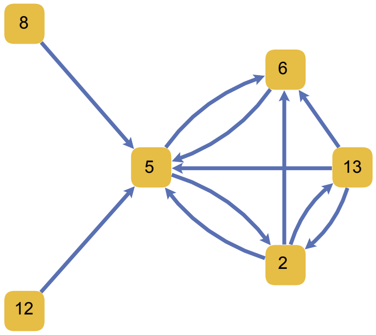
Out[3]=



In[4]:=
```
G4 = Graph[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14},
   {1 → 4, 1 → 10, 1 → 11, 2 → 6, 2 → 5, 2 → 13, 3 → 1, 3 → 7, 3 → 11, 4 → 1,
    4 → 11, 5 → 2, 5 → 6, 6 → 5, 7 → 1, 7 → 14,  8 → 5, 9 → 1, 9 → 10, 9 → 14,
    10 → 1, 11 → 3, 12 → 5, 13 → 5, 13 → 2, 13 → 6, 14 → 1, 14 → 3,  14 → 9},
   GraphStyle → "SmallNetwork", GraphLayout → "SpringEmbedding"]
```



Out[4]=



Project1 (discussion)

The project is based on the Perron-Frobenius Theorem

Theorem: If M is a positive, column stochastic nxn matrix then the following statements are true:

- The number 1 is an eigenvalue of M, of multiplicity one
- The eigenvalue 1 is the largest eigenvalue of M; all the other eigenvalues of M are strictly less than 1 in absolute value.
- There exists a unique probabilistic eigenvector w corresponding to the eigenvalue 1.
- **(Power Method Convergence)** Let v be a probabilistic eigenvector. Then the sequence v, Mv, ... , M^kv converges to w as k goes to ∞.

Page Rank Algorithm

We use the Perron - Frobenius Theorem to give a rigorous mathematical justification for the Page Rank Algorithm.

The PageRank algorithm provides the basis for Google' s web search tools. As we described in the lecture, PageRank is a recursive algorithm, and it works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites and therefore, the importance of any website can be judged by looking at the number and quality of the pages that link to it. Components of the PageRank vector serve as "importance" weights for web pages, independent of their textual content, solely based on the hyperlink structure of the web.

Back to the Web Graph

The Web Graph is not necessarily strongly connected, so its transition matrix need not be positive or column stochastic! The Web graph could have disconnected components or dangling nodes (nodes with no outgoing edges)

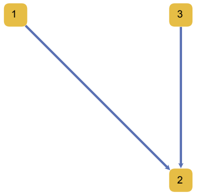
which would correspond to 0 column vectors in the transition matrix.

Some of the graphs in the project might also have these problems, especially the randomly generated ones. Then we cannot simply find the “unique” probabilistic eigenvector corresponding to the eigenvalue 1 and say that this is the Page Rank vector of our graph. We first need to adjust our algorithm slightly, to make it work also for graphs with dangling nodes and disconnected components

Dangling nodes

☹ Consider the graph depicted below. Its transition matrix is given on the right, and we can see that it is not column stochastic or positive either!

```
H = Graph[{1 → 2,  3 → 2},
   GraphStyle → "SmallNetwork",  VertexSize → 0.15];
TransitionH = Transpose[AdjacencyMatrix[H]];
GraphicsRow[{H, MatrixForm[TransitionH]}]
```



$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

☹ Perron-Frobenius Theorem cannot be directly applied! Still, let us see if the Power Method converges to something useful.

```
u = {1 / 3, 1 / 3, 1 / 3}; MatrixForm[u]
For[k = 1,  k ≤ 3,  k++,
 Print[MatrixForm[MatrixPower[TransitionH, k].u]]]
```

$$\begin{pmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ \frac{2}{3} \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

We notice that the sequence of vectors u, Au, ..., $A^k u$ converges to the 0-vector. Therefore, in this case, the rank of every page is 0. This is counterintuitive, as page 2 has 2 incoming links, so it must have some importance!

☺ A solution to the Dangling Nodes Problem

Change the transition matrix of the graph so that it becomes **column stochastic**. More specifically, change any 0-column in the transition matrix with a column where all elements are equal to 1/n.

This models the behavior of a random surfer that after viewing a web page with no outgoing edges, will simply choose another page from the Web with equal probability 1/n and go there.

```
For[i = 1, i ≤ 3, i++, TransitionH[[i, 2]] = 1 / 3];
MatrixForm[TransitionH]
```

$$\begin{pmatrix} 0 & \frac{1}{3} & 0 \\ 1 & \frac{1}{3} & 1 \\ 0 & \frac{1}{3} & 0 \end{pmatrix}$$

```
u = {1 / 3, 1 / 3, 1 / 3}
For[k = 1, k ≤ 40, k++, Print[N[MatrixPower[TransitionH, k].u]]]
```

$$\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$$

```
{0.111111, 0.777778, 0.111111}
{0.259259, 0.481481, 0.259259}
{0.160494, 0.679012, 0.160494}
{0.226337, 0.547325, 0.226337}
{0.182442, 0.635117, 0.182442}
{0.211706, 0.576589, 0.211706}
```

```
{0.192196, 0.615607, 0.192196}
{0.205202, 0.589595, 0.205202}
{0.196532, 0.606937, 0.196532}
{0.202312, 0.595376, 0.202312}
{0.198459, 0.603083, 0.198459}
{0.201028, 0.597945, 0.201028}
{0.199315, 0.60137, 0.199315}
{0.200457, 0.599087, 0.200457}
{0.199696, 0.600609, 0.199696}
{0.200203, 0.599594, 0.200203}
{0.199865, 0.600271, 0.199865}
{0.20009, 0.59982, 0.20009}
{0.19994, 0.60012, 0.19994}
{0.20004, 0.59992, 0.20004}
{0.199973, 0.600053, 0.199973}
{0.200018, 0.599964, 0.200018}
{0.199988, 0.600024, 0.199988}
{0.200008, 0.599984, 0.200008}
{0.199995, 0.600011, 0.199995}
{0.200004, 0.599993, 0.200004}
{0.199998, 0.600005, 0.199998}
{0.200002, 0.599997, 0.200002}
{0.199999, 0.600002, 0.199999}
{0.200001, 0.599999, 0.200001}
{0.2, 0.600001, 0.2}
{0.2, 0.599999, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
{0.2, 0.6, 0.2}
```

Disconnected components

☺ Solution: Let T be the transition matrix of the Web graph. T is an nxn matrix, where n is huge. Fix a positive constant p between 0 and 1, called the damping factor (a typical value for p is 0.15). Define the Page Rank matrix (also known as the Google matrix) of the graph to be

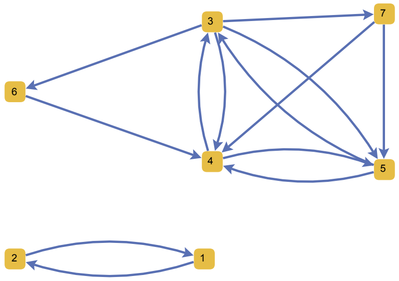
$$M=(1-p)T+p\begin{pmatrix} \frac{1}{n} & \frac{1}{n} & .. & \frac{1}{n} \\ .. & .. & .. & .. \\ \frac{1}{n} & \frac{1}{n} & .. & \frac{1}{n} \end{pmatrix}$$

Theorem: If T is a column stochastic matrix, then M is a positive and column stochastic matrix.

The matrix M models the behaviour of the random surfer as follows: most of the time, a surfer will follow links from the current page he/she is viewing. From a page i, the surfer will follow the outgoing links and move on to one of the neighbors of page i. A smaller, but positive percentage of the time, the surfer will dump the current page and choose arbitrarily a different page from the web and "teleport" there. The damping factor p reflects the probability that the surfer quits the current page and goes to a new one. Since he/she can teleport to any web page, each page has equal probability $\frac{1}{n}$ to be chosen. So the Internet graph becomes strongly connected.

An example of a graph with disconnected components

```
G2 = Graph[{1, 2, 3, 4, 5, 6, 7},
  {1→2, 2→1, 3→4, 4→3, 4→5, 5→4, 5→3, 3→5, 3→6,
   6→4, 7→5, 7→4, 3→7}, GraphStyle→"SmallNetwork"]
```



Find the transition matrix first and compute its eigenvalues:

```
Lout = VertexOutDegree[G2];
TA = Transpose[AdjacencyMatrix[G2]];
f[i_, j_] := If[Lout[[j]] != 0, TA[[i]][[j]] / Lout[[j]], 0];
T = Table[f[i, j], {i, Length[TA]}, {j, Length[TA]}];
MatrixForm[T]
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
Eigenvalues[T]
```

$$\left\{-1, 1, 1, -\frac{1}{2}, \frac{1}{4}\left(-1+i\sqrt{3}\right), \frac{1}{4}\left(-1-i\sqrt{3}\right), 0\right\}$$

```
Eigensystem[T]
```

$$\left\{\left\{-1, 1, 1, -\frac{1}{2}, \frac{1}{4}\left(-1+i\sqrt{3}\right), \frac{1}{4}\left(-1-i\sqrt{3}\right), 0\right\},\right.$$
$$\left\{\{-1, 1, 0, 0, 0, 0, 0\}, \left\{0, 0, 4, \frac{13}{3}, \frac{11}{3}, 1, 1\right\}, \{1, 1, 0, 0, 0, 0, 0\},\right.$$
$$\{0, 0, 0, -1, 1, 0, 0\}, \left\{0, 0, -1+i\sqrt{3}, -i\sqrt{3}, -1, 1, 1\right\},$$
$$\left.\left.\left\{0, 0, -1-i\sqrt{3}, i\sqrt{3}, -1, 1, 1\right\}, \{0, 0, 0, -1, 1, -1, 1\}\right\}\right\}$$

The eigenspace corresponding to the eigenvalue 1 is now two-dimensional! So there are many probabilistic eigenvectors corresponding to the eigenvalue 1. So for different choices of initial probabilistic vectors u_2 and u_3, the sequences u_2, Tu_2, ... T^ku_2.... and u_3, Tu_3, ... T^ku_3 can converge to different limits, as one can see below:

```
n = Length[T];
u2 = Table[1 / n, {n}];
For[k = 1, k ≤ 10, k++, Print[N[MatrixPower[T, k].u2]]]
```

```
{0.142857, 0.142857, 0.142857, 0.321429, 0.178571, 0.0357143, 0.0357143}

{0.142857, 0.142857, 0.25, 0.178571, 0.214286, 0.0357143, 0.0357143}

{0.142857, 0.142857, 0.196429, 0.223214, 0.169643, 0.0625, 0.0625}

{0.142857, 0.142857, 0.196429, 0.227679, 0.191964, 0.0491071, 0.0491071}

{0.142857, 0.142857, 0.209821, 0.21875, 0.1875, 0.0491071, 0.0491071}

{0.142857, 0.142857, 0.203125, 0.219866, 0.186384, 0.0524554, 0.0524554}

{0.142857, 0.142857, 0.203125, 0.222656, 0.186942, 0.0507813, 0.0507813}

{0.142857, 0.142857, 0.204799, 0.220424, 0.1875, 0.0507813, 0.0507813}

{0.142857, 0.142857, 0.203962, 0.221122, 0.186802, 0.0511998, 0.0511998}

{0.142857, 0.142857, 0.203962, 0.221191, 0.187151, 0.0509905, 0.0509905}
```

```
u3 = {1 / 2, 1 / 2, 0, 0, 0, 0, 0};
For[k = 1, k ≤ 10, k++, Print[N[MatrixPower[T, k].u3]]]
```

```
{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}

{0.5, 0.5, 0., 0., 0., 0., 0.}
```

Let us now choose a damping factor p different from 0 and redo the previous computations.

```
B = Table[1 / n, {n}, {n}];
p = 0.2;
M = (1 - p) * T + p * B; MatrixForm[M]
For[k = 1, k ≤ 10, k++, Print[MatrixPower[M, k].u2]]
```

```
( 0.02857  0.82857  0.02857  0.02857  0.02857  0.02857  0.02857 )
( 0.82857  0.02857  0.02857  0.02857  0.02857  0.02857  0.02857 )
( 0.02857  0.02857  0.02857  0.42857  0.42857  0.02857  0.02857 )
( 0.02857  0.02857  0.22857  0.02857  0.42857  0.82857  0.42857 )
( 0.02857  0.02857  0.22857  0.42857  0.02857  0.02857  0.42857 )
( 0.02857  0.02857  0.22857  0.02857  0.02857  0.02857  0.02857 )
( 0.02857  0.02857  0.22857  0.02857  0.02857  0.02857  0.02857 )
```

```
{0.142857, 0.142857, 0.142857, 0.285714, 0.171429, 0.0571429, 0.0571429}

{0.142857, 0.142857, 0.211429, 0.194286, 0.194286, 0.0571429, 0.0571429}

{0.142857, 0.142857, 0.184, 0.217143, 0.171429, 0.0708571, 0.0708571}

{0.142857, 0.142857, 0.184, 0.218971, 0.180571, 0.0653714, 0.0653714}

{0.142857, 0.142857, 0.188389, 0.216046, 0.179109, 0.0653714, 0.0653714}

{0.142857, 0.142857, 0.186633, 0.216338, 0.178816, 0.0662491, 0.0662491}

{0.142857, 0.142857, 0.186633, 0.216923, 0.178933, 0.0658981, 0.0658981}

{0.142857, 0.142857, 0.186914, 0.216549, 0.179027, 0.0658981, 0.0658981}

{0.142857, 0.142857, 0.186802, 0.216643, 0.178933, 0.0659542, 0.0659542}

{0.142857, 0.142857, 0.186802, 0.21665, 0.17897, 0.0659318, 0.0659318}
```

```
For[k = 1, k ≤ 20, k++, Print[MatrixPower[M, k].u3]]
```

```
{0.428571, 0.428571, 0.0285714, 0.0285714, 0.0285714, 0.0285714, 0.0285714}
{0.371429, 0.371429, 0.0514286, 0.08, 0.0571429, 0.0342857, 0.0342857}
{0.325714, 0.325714, 0.0834286, 0.102857, 0.0845714, 0.0388571, 0.0388571}
{0.289143, 0.289143, 0.103543, 0.125714, 0.101943, 0.0452571, 0.0452571}
{0.259886, 0.259886, 0.119634, 0.144366, 0.117669, 0.04928, 0.04928}
{0.23648, 0.23648, 0.133385, 0.158702, 0.129957, 0.0524983, 0.0524983}
{0.217755, 0.217755, 0.144035, 0.170229, 0.139728, 0.0552485, 0.0552485}
{0.202776, 0.202776, 0.152554, 0.179568, 0.147569, 0.0573784, 0.0573784}
{0.190792, 0.190792, 0.159426, 0.186964, 0.153861, 0.0590823, 0.0590823}
{0.181205, 0.181205, 0.164901, 0.1929, 0.158875, 0.0604567, 0.0604567}
{0.173535, 0.173535, 0.169281, 0.19765, 0.162894, 0.0615517, 0.0615517}
{0.1674, 0.1674, 0.172789, 0.201447, 0.166108, 0.0624277, 0.0624277}
{0.162491, 0.162491, 0.175594, 0.204486, 0.168679, 0.0631292, 0.0631292}
{0.158564, 0.158564, 0.177838, 0.206917, 0.170736, 0.0636902, 0.0636902}
{0.155423, 0.155423, 0.179633, 0.208862, 0.172382, 0.0641389, 0.0641389}
{0.15291, 0.15291, 0.181069, 0.210417, 0.173698, 0.064498, 0.064498}
{0.150899, 0.150899, 0.182218, 0.211662, 0.174751, 0.0647852, 0.0647852}
{0.149291, 0.149291, 0.183137, 0.212658, 0.175594, 0.065015, 0.065015}
{0.148004, 0.148004, 0.183872, 0.213454, 0.176268, 0.0651988, 0.0651988}
{0.146975, 0.146975, 0.18446, 0.214092, 0.176807, 0.0653458, 0.0653458}
```

Tips for the project:

- Create a graph (random or taken from a list). Use the command **Graph[{List of vertices}, {List of Edges}]** to create a graph starting from a set of vertices and edges. You can also create a graph starting from the adjacency matrix if you use the command **AdjacencyGraph[Matrix]**. Do not use **GraphPlot[...]** to create graphs, use it only for display, because it creates a Graphics object, and not a Graph object. So you will end up with an object that can be nicely plotted, but most of the *Mathematica* algorithms for graphs will not work with Graphics objects.
- Find the Transition Matrix of the graph, using the function that you wrote for HW2. Change the Transition Matrix to get rid of the Dangling Nodes (that is, change the 0-columns, if any).
- If n is the number of vertices of your graph, then generate an nxn matrix with all elements equal to 1/n.

- Find the unique probabilistic eigenvector corresponding to the eigenvalue 1. Since you have to higlight the most important 5 vertices, you may need to sort your page-rank vector. **Sort[List]** will give the elements in increasing order. **Sort[List, Greater]** sorts the list in decreasing order, as you can see by executing the one-line code below. You can then use the top 5 values to highlight the most important nodes in the graph.

```
Sort[{0.2, 0.55, 0.1, 0.8}, Greater]
```

- To show a step-by-step computation of the page-rank vector, one needs to start with a probabilistic eigenvector u and multiply u by succesive powers of the (modified) Transition Matrix. At first run, you can assume that k<=10 and color the vertices according to the ranks given by u, Tu, T^2u, ... ,T^ku. The sequence should converge exponentially fast to the page-rank vector, so you should see the relative rankings stabilizing after a couple of steps. If you discover than a bigger value of k is needed, you can adjust k afterwards.
- Make sure first that your project works on the test graphs (and other fixed graphs). Then make it work for random graphs.
- Working with random graphs inside Manipulate[...] can be a little bit tricky if you have a couple of sliders. In particular, you may discover that a new graph is generated each time you move a slider, even if the slider has no connection with the graph. The examples below should help see why.

Tips for the project:

- Create a graph (random or taken from a list). Use the command **Graph[{List of vertices}, {List of Edges}]** to create a graph starting from a set of vertices and edges. You can also create a graph starting from the adjacency matrix if you use the command **AdjacencyGraph[Matrix]**. Do not use **GraphPlot[...]** to create graphs, use it only for display, because it creates a Graphics object, and not a Graph object. So you will end up with an object that can be nicely plotted, but most of the *Mathematica* algorithms for graphs will not work with Graphics objects.
- Find the Transition Matrix of the graph, using the function that you wrote for HW2. Change the Transition Matrix to get rid of the Dangling Nodes (that is, change the 0-columns, if any, to columns with all entries 1/n).
- If n is the number of vertices of your graph, then generate an nxn matrix with all elements equal to 1/n.
- Find the unique probabilistic eigenvector corresponding to the eigenvalue 1. Since you have to higlight the most important 5 vertices, you may need to sort your page-rank vector. **Sort[List]** will give the elements in increasing order. **Sort[List, Greater]** sorts the list in decreasing order, as you can see by executing the one-line code below. You can then use the top 5 values to highlight the most important nodes in the graph.

```
Sort[{0.2, 0.55, 0.1, 0.8}, Greater]
```

- To show a step-by-step computation of the page-rank vector, one needs to start with a probabilistic vector u and multiply u by succesive powers of the (modified) Transition Matrix. At first run, you can assume that k<=10 and color the vertices according to the ranks given by u, Tu, T^2u, ... ,T^ku. The sequence should converge exponentially fast to the page-rank vector, so you should see the relative rankings stabilizing after a couple of steps. If you discover than a bigger value of k is needed, you can adjust k afterwards.
- Make sure first that your project works on the test graphs (and other fixed graphs). Then make it work for random graphs.
- Working with random graphs inside **Manipulate[...]** can be a little bit tricky if you have a couple of sliders. In particular, you may discover that a new graph is generated each time you move a slider, even if the slider has no connection with the graph. The examples below should help see why.

Manipulate[...]

In the practice lab we saw that certain global variable assignment can cause Manipulate[...] to reevaluate itself more than needed. The first two examples below are relevant for this problem.

```
Manipulate[
 z = 10;
 z = z + n,
 {n, 1, 10}]
```

```
z = 10;
Manipulate[
 z = z + n,
 {n, 1, 10}]
```

So we need to make z a local variable inside Module[{Variables}, Body-of-Module] to prevent reevaluations. If z is the only local variable, then we can write: **Module[{z}, z=10; z=z+n]**

```
Manipulate[Module[{z},
  z = 10;
  z = z + n],
 {n, 1, 10}]
```

Functions with local variables

Sometimes, we need to do more computations in the body of a function, that span several lines of code. Perhaps we also need auxiliary variables to do those computations. We will declare these variables as local variables inside **Module[...]**.

Suppose we want to construct a function that takes as input a list, sorts the list in decreasing order, and checks whether the sum of the biggest 3 elements minus the sum of the smallest 3 elements is bigger than 10.

```
f6[L_] := Module[{aux, sum},
  aux = Sort[L, Greater];
  Print[aux];
  aux2 = Sort[L];
  Print[aux2];
  sum = (aux[[1]] + aux[[2]] + aux[[3]] ) -
    (aux2[[1]] + aux2[[2]] + aux2[[3]]);
  Return[sum]
 ]
```

```
f6[{1, 2, 6, 3, 7, 8}]
```

```
{8, 7, 6, 3, 2, 1}

{1, 2, 3, 6, 7, 8}

15
```

If we test it on a list with less than 2 elements, we get an error.

```
f6[{1, 2}]
```

So, we redefine this function to cover this case.

```
f6[L_] :=
 "Error! The given list has less than 3 elements" /; Length[L] < 3
```

```
f6[{1, 2}]
```

```
Error;The given list has less than 3 elements
```

Working with random elements inside Manipulate - Easier or Harder?

How about generating random elements inside the body of **Manipulate[...]**? Should the code below reevaluate itself as fast as possible, each time generating a new random integer?

```
Manipulate[
 x = RandomInteger[n],
 {n, {10, 100}}
]
```

Every time you evaluate **RandomReal[...]** or **RandomInteger[...]** or **RandomGraph[...]**, you get a different answer, and you might think that an assignment like **x=RandomReal[...]** inside **Manipulate[...]** should therefore constantly update itself as fast as

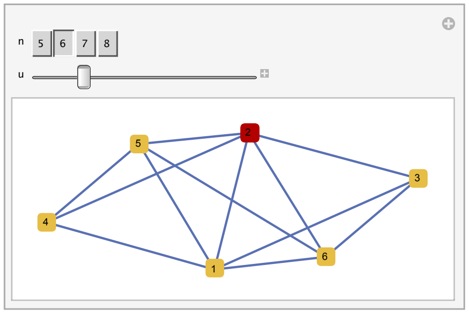
possible. But this would normally not be useful, and would in fact have negative consequences for a number of algorithms that use randomness internally. For this reason, these functions are not "**ticklish**", in the sense that they do not trigger updates. If you want to see actually new random numbers, you have to use Refresh to specify how frequently you want the output updated.

Harder because ...

In the next example, we want to build an interactive model with two controls, n and u, that generates a random graph called K with n vertices and 2n edges and highlights vertex u of graph K.

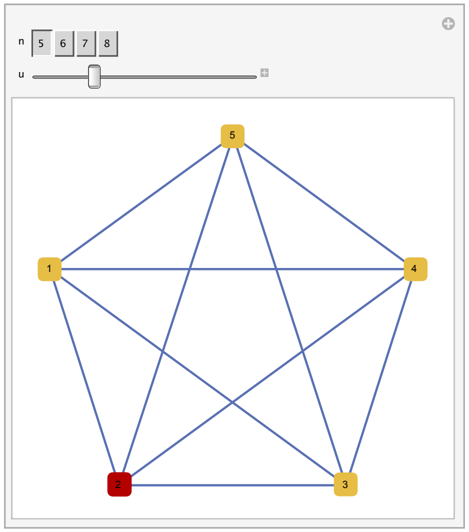
- **Problem:** A new graph should be generated only when we change n, the number of nodes. However, when the vertex u is changed, the graph changes as well! So each time we change u, we highlight vertex u of a different graph.

```
Manipulate[
 K = RandomGraph[{n, 2 * n}];
 HighlightGraph[K, {u}, GraphStyle → "SmallNetwork"],
 {n, {5, 6, 7, 8}},
 {u, 1, n, 1}
]
```



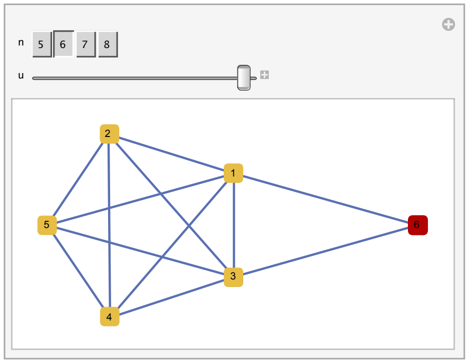
Making K a local variable inside Module[...] does not solve the problem, because the line K=RandomGraph[...] did not trigger by itself any reevaluations!

```
Manipulate[Module[{K},
   K = RandomGraph[{n, 2 * n}];
   HighlightGraph[K, {u}, GraphStyle → "SmallNetwork"]],
 {n, {5, 6, 7, 8}},
 {u, 1, n, 1}
]
```



Making K a local variable inside a DynamicModule[...] solves the problem.

```
Manipulate[DynamicModule[{K},
   K = RandomGraph[{n, 2 * n}];
   Dynamic[HighlightGraph[K, {u}, GraphStyle → "SmallNetwork"]]],
 {n, {5, 6, 7, 8}},
 {u, 1, n, 1}
]
```



DynamicModule[{x,y,…},expr]
represents an object which maintains the same local instance of the symbols x, y, … in the course of all evaluations of Dynamic objects in expr.

Variables specified in a DynamicModule will by default have their values maintained from one dynamic update to the next.

Conclusion

For the project, you may consider using the following syntax:

Define functions & global variables

```
Manipulate[
    DynamicModule[{ graph g},
         Define graph g etc. according to the user choice;
```

```
                Dynamic[GraphicsRow[{
                    Module[{all other local variables},
                      Your page - rank computations here;
                      HighlightGraph[ ...]
                    ]
                 }]]
        ],
 controls for sliders etc.,
 SaveDefinitions->True
 ]
```

Using Dynamic[...]

Let us define the variable t and then perform a simple computation, like t^2+5.

```
t = 10;
```

This is a static output:

```
t^2 + 5
```

This is a dynamic output:

```
Dynamic[t^2 + 5]
```

At first glance, the static output and the dynamic output look the same. However, notice what happens to the output of Dynamic[t^2+5] when we change the value of t.

```
t = 6;
```

The output of Dynamic[t^2+5] has changed to reflect the current value of t. The output of t^2+5 remains the same, until we compile it again (Shift+Enter). Run also the following lines to understand the difference between static and dynamic output.

```
MousePosition[]
```

```
Dynamic[MousePosition[]]
```

Dynamic[...] can give infinite loops :

```
x = 0;
Dynamic[x = x + 1]
```

However, an assignment like t = RandomReal[...] inside Dynamic[...] does not constantly update itself as fast as possible, because this function is not "ticklish", in the sense that it does not trigger updates.

```
t = 0;
Dynamic[t = RandomReal[10]]
```

Using DynamicModule[...] and Dynamic[...]

Suppose now that we want to evaluate a polynomial expression like ax^2+bx+c such that the output always reflects the current value of x. The coefficients a, b and c are assumed fixed, so any changes to these coefficients later on in the notebook should not trigger any retroactive dynamic updates of the expression ax^2+bx+c.

```
a = 1;
b = 2;
c = 3;
x = 0;
Dynamic[a*x^2+b*x+c]
```

Try now to change the value of x to 1, then to change the value of c to 1000. What do you notice?

```
x = 1;
```

```
c = 1000;
```

Let us now clear our variables and try again, this time using a DynamicModule[...].

```
Clear[a, b, c, x]
```

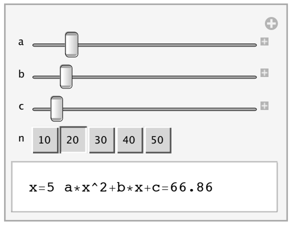
```
DynamicModule[{a, b, c},
 a = 1;
 b = 2;
 c = 3;
 x = 0;
 Dynamic[a*x^2 + b*x + c]]
```

Try again to change the value of x to 1, then to change the value of c to 1000. What do you notice?

Exercise

Now suppose we want to use Manipulate[...] to display the value of ax^2+bx+c, where a,b,c are parameters with values between 1 and 10, and x is a randomly generated number. We first try the following code:

```
Manipulate[
 x = RandomInteger[n];
 Row[{"x=", x, " a*x^2+b*x+c=", a * x^2 + b * x + c}],
 {a, 1, 10}, {b, 1, 10}, {c, 1, 10}, {n, {10, 20, 30, 40, 50}}]
```



```
x=5 a*x^2+b*x+c=66.86
```

Exercise: Rewrite the code above, so that the value of x gets updated ONLY when the parameter n is changed.

Solvers

Problem 1.5 (Homework I)

One of the applications of matrix algebra is to solve systems of linear equations, usually in a large number of variables. A system of m such equations in the n variables x_1, x_2, ..., x_n, can be written explicitly:

$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1\,n}\,x_n = b_1$
$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2\,n}\,x_n = b_2$
....
$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}\,x_n = b_m$

The same system can be written in the more convenient matrix notation as Ax=b, where A is the m × n coefficient matrix, x is the (column) vector of length n containing the variables, and b is the (column) vector of length m of the coefficients on the right hand sides of the equalities. By general theory, a system of linear equations has no solution, exactly one solution or infinitely many solutions. In *Mathematica*, we can use the command **Solve[expression,variables]**, to solve the system Ax=b for the variable x={x_1,x_2, ... , x_n}.

Problem 1.5 (Homework I)

◆ Find the inverse of a random 3x3 matrix A with integer coefficients (if it exists!), without using the *Mathematica* function **Inverse[...]**. *Recall that the matrix A is called invertible if there exists a (unique) 3x3 matrix B such that AB=BA=I_3, where I_3 is the identity matrix. If A is invertible, then the matrix B is called the inverse of the matrix A, and it is denoted by A^{-1}.*

Solution:

To find the inverse using the function Solve, we can solve three systems of equations

$AX = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $AY = \begin{pmatrix}0\\1\\0\end{pmatrix}$ and $AZ = \begin{pmatrix}0\\0\\1\end{pmatrix}$. If all three systems have unique solutions,

then the inverse matrix is the 3 x3 matrix with columns X, Y and Z.

```
A = {{1, 2, 3}, {1, 5, 6}, {7, 8, 9}}; Print["A=" MatrixForm[A]];
b1 = {1, 0, 0}; b2 = {0, 1, 0}; b3 = {0, 0, 1};
X = Table[x[j], {j, 3}];
Y = Table[y[j], {j, 3}];
Z = Table[z[j], {j, 3}];
(* Solve three systems *)
S1 = Solve[A.X == b1, X]
S2 = Solve[A.Y == b2, Y]
S3 = Solve[A.Z == b3, Z]
n1 = 0; n2 = 0; n3 = 0;
(* Test if all three systems have unique solutions *)
If[S1 ≠ {}, n1 = Dimensions[S1][[2]]];
If[S2 ≠ {}, n2 = Dimensions[S2][[2]]];
If[S3 ≠ {}, n3 = Dimensions[S3][[2]]];
If [n1 == n2 == n3 == 3,
   (* invertible! *)
   Print["Matrix A is invertible"];
   soln = {Values[S1][[1]], Values[S2][[1]], Values[S3][[1]]};
   inv = Transpose[soln];
   Print ["and the inverse matrix is A^-1=", MatrixForm[inv], "."
          "  Inverse test:  ",
  MatrixForm[A], MatrixForm[inv], "=", MatrixForm[A.inv]],
   (* not invertible *)
   Print["Matrix A is not invertible"]
]
```

Transformation rules

The solution returned by Solve[...] is a list of lists of transformation rules.

```
S1 = Solve[A.X == b1, X]
```

$$\left\{\left\{x[1]\to\frac{1}{6},\ x[2]\to-\frac{11}{6},\ x[3]\to\frac{3}{2}\right\}\right\}$$

The association $x[1]\to\frac{1}{6}$ **is called a transformation rule.**

How can we extract the useful value “1/6” from the transformation rule?

```
"Naive Idea"
```

```
S1[[1]]
```

```
S1[[1]][[1]]
```

```
S1[[1]][[1]][[2]]
```

```
"Better Idea"
```

```
Values[S1]
```

```
Keys[S1]
```

We can get rid of one set of curly brackets by using the command **Flatten[...]**

```
Flatten[Values[S1]]
```

However, we lost the connection between the name of the variables and the numeric solutions obtained.

Transformation rules & Replacement Operator “/.”

expr /. rule lhs->rhs   Applies a transformation rule lhs->rhs to expression expr
The replacement operator (pronounced “slash-dot”) applies rules to expressions.

In[1]:=
```
Sin[x] /. x → 4
```

Out[1]= Sin[4]

If you give a list of rules, you get a list of results. Each rule will be tried once on each part of the expression.

In[2]:=
```
Sin[x + y] /. {x → 4, y → 3}
```

Out[2]= Sin[7]

In[3]:=
```
Sin[x + y] /. {x → Pi, y → Pi / 2}
```

Out[3]= -1

In[4]:=
```
x + x^2 + y * x + x^3 /. {x^3 → x, y → x, x → 20}
```

Out[4]= 420 + 21 x

If you give a list of lists of rules, you get a list of results.

```
Sin[x] /. {{x → 0}, {x → 1}, {x -> 2}, {x → 3}, {x → 4}}
```

In[5]:=
```
Sin[x + y] /. {{x → Pi, y → Pi / 2}, {x → 4, y → 3}}
```

Out[5]= {-1, Sin[7]}

Short exercise : Use the transformation rule to change the 0 - rows of a matrix like {{1,2,3}, {0,0,0}, {0,1,0}, {0,0,0}, {0,2,3}} with 1 - rows.

Back to the homework exercise

In[6]:=
```
Clear[X, x];
A = {{1, 2, 3}, {1, 5, 6}, {7, 8, 9}};
X = Table[x[i][j], {i, 3}, {j, 3}];
solution = Solve[A.X == IdentityMatrix[3], Flatten[X]]
```

Out[9]= $\{\{x[1][1] \to \frac{1}{6}, x[1][2] \to -\frac{1}{3}, x[1][3] \to \frac{1}{6}, x[2][1] \to -\frac{11}{6}, x[2][2] \to \frac{2}{3}, x[2][3] \to \frac{1}{6}, x[3][1] \to \frac{3}{2}, x[3][2] \to -\frac{1}{3}, x[3][3] \to -\frac{1}{6}\}\}$

In[10]:=
```
X /. Flatten[solution]
```

Out[10]= $\{\{\frac{1}{6}, -\frac{1}{3}, \frac{1}{6}\}, \{-\frac{11}{6}, \frac{2}{3}, \frac{1}{6}\}, \{\frac{3}{2}, -\frac{1}{3}, -\frac{1}{6}\}\}$

In[11]:=
```
MatrixForm[X /. Flatten[solution]]
```

Out[11]//MatrixForm=

$$\begin{pmatrix} \frac{1}{6} & -\frac{1}{3} & \frac{1}{6} \\ -\frac{11}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{3}{2} & -\frac{1}{3} & -\frac{1}{6} \end{pmatrix}$$

Solving equations

```
Clear[x]
```

Mathematica can solve various kinds of equations. For example set

In[12]:=
```
eqn = x^2 + 2 a x + b == 0
```

Out[12]= $b + 2\,a\,x + x^2 == 0$

Note that the `Set` operator (=) has lower precedence than the `Equal` operator (==). To solve this equation for x you use the command

In[13]:=
```
sol = Solve[eqn, x]
```

Out[13]= $\{\{x \to -a - \sqrt{a^2 - b}\}, \{x \to -a + \sqrt{a^2 - b}\}\}$

Note that the result is given as a list with two lists of rules. Each sublist represents a distinct solution. To verify that this is indeed a solution of the quadratic equation, we can substitute the solution back into the equation and simplify the result.

```
In[14]:= eqn /. sol
```

$$\text{Out[14]= } \left\{2\,a\left(-a-\sqrt{a^2-b}\right)+\left(-a-\sqrt{a^2-b}\right)^2+b==0,\ 2\,a\left(-a+\sqrt{a^2-b}\right)+\left(-a+\sqrt{a^2-b}\right)^2+b==0\right\}$$

```
In[15]:= Simplify[eqn /. sol]

Out[15]= {True, True}
```

You can now also use the solutions for other computations. Suppose you want to evaluate some function f1 on the solution set:

```
f1[x] /. sol
```

Short exercise: We know that sol contains the roots of the quadratic polynomial (given as a list of transformation rules) and we want to check (using transformation rules!) that the sum of the two roots is -2a and the product of the two roots is b.

Numerical Solvers

NSolve[expr, vars] -- attempts to find numerical approximations to the solutions of the system expr of equations or inequalities for the variables vars.
NSolve[expr,vars,Reals] -- finds solutions over the domain of real numbers.

Example: Find the solutions of a polynomial equation of degree 5

```
Solve[x^5 - 2 x + 3 == 0, x]
```

```
{{x → Root[3 - 2 #1 + #1^5 &, 1]},
 {x → Root[3 - 2 #1 + #1^5 &, 2]}, {x → Root[3 - 2 #1 + #1^5 &, 3]},
 {x → Root[3 - 2 #1 + #1^5 &, 4]}, {x → Root[3 - 2 #1 + #1^5 &, 5]}}
```

```
NSolve[x^5 - 2 x + 3 == 0, x]
```

```
{{x → -1.42361}, {x → -0.246729 - 1.32082 i}, {x → -0.246729 + 1.32082 i},
 {x → 0.958532 - 0.498428 i}, {x → 0.958532 + 0.498428 i}}
```

```
NSolve[x^5 - 2 x + 3 == 0, x, Reals]
```

```
{{x → -1.42361}}
```

Functions with no name (pure functions #)

#^5 &

When solving equations where solutions do not have any closed form, you may see these kind of functions returned by Mathematica. These are functions with no name, or pure functions.

represents the first variable (first argument) in a pure function
#n represents the n^th variable (argument) in a pure function
& the ampersand marks the end of the pure function

The name of a function is irrelevant if you do not intend to refer to the function again, so the Wolfram language lets you define functions with no names. Pure functions in the Wolfram Language can take any number of arguments. The Wolfram Language allows you to avoid using explicit names for the arguments of pure functions, and instead to specify the arguments by giving “slot numbers” . In a Wolfram Language pure function, **#n** stands for the nth argument you supply. **#** stands for the first argument.

#^5 & is simply the function that takes any number and raises it to power 5. We can use this function to compute 2^5 for example.
#1+#2 & is a function that computes the sum of two numbers.

```
#^5 &[2]
```

```
32
```

```
#1 + #2 &[1, 2]
```

```
3
```

Example : Sort the list **L = {{1, 2}, {2, 5}, {1, 1}, {4, 8}, {3, 4}, {9,0}, {7,6}}** in ascending order, according to the following criterion : **{a, b} < {c, d} iff b < d**

```
L = {{1, 2}, {2, 5}, {1, 1}, {4, 8}, {3, 4}, {9, 0}, {7, 6}};
Sort[L]   (* the first two sort commands don't help,
as they sort by the first term of the pair *)
Sort[L, Greater]
Sort[L, #1[[2]] < #2[[2]] &]
```

Short Exercise: Sort a list of the form S = {{1, 1 -> 2}, {2, 2 -> 5}, {1, 1 -> 1}, {4, 4 -> 8}, {3, 3 -> 4}, {9, 9 -> 0}, {7, 7 -> 6}} according to the following criterion: {a, a->b} < {c, c->d} iff b<d

Newton' s Method

Numeric solvers that are used for finding real roots usually implement Newton' s Method.

Newton' s Method is a method for finding successively better approximations to the roots (or zeroes) of a real - valued differentiable function f : [a,b] -> R. Suppose that f has a single root in the interval [a,b]. The idea of the method is as follows: one starts with an initial guess which is reasonably close to the true root, then the function is approximated by its tangent line (which can be computed using the tools of calculus), and one computes the x-intercept of this tangent line (which is easily done with elementary algebra). This x-intercept will typically be a better approximation to the function's root than the original guess, and the method can be iterated.

Suppose we have some current approximation x_n. The equation of the tangent line to the curve y = f(x) at the point (x_n, $f(x_n)$) is
$y = f\,'(x_n)(x-x_n) + f(x_n)$.
To find the intersection point of the tangent line with the x-axis we set y=0 and solve for x. We get
$x_{n+1} = x_n - \frac{f(x_n)}{f\,'(x_n)}$. We get a sequence of succesive approximations $x_1, \ldots x_n, x_{n+1,\ldots}$ of the root of the function f, in the interval [a,b].
The method will usually converge, provided this initial guess x_1 is close enough to the unknown zero, and that $f'(x_1) \neq 0$ (i.e. the tangent line is not horizontal).

MAT 331: Homework 3

Problem 3.1

The replacement operator /. (slash-dot) applies a transformation rule to an expression. If you give a list of rules, each rule will be tried once on each part of the expression. If you give a list of lists of rules, you get a list of results; each sublist is treated like an independent set of rules.

```
x^2 + x^3 + y^2 + z /. {x_ -> 2}
```

```
x^2 + x^3 + y^2 + z /. {x -> 2, y → 3, z → 1}
```

```
x^2 + x^3 + y^2 + z /. {{x → 2}, {x → 2, y → 3, z → 1}, {x → 0, z → 0}}
```

```
x^2 + x^3 + y^2 + z /. {x_^n_ → a}
```

1. Write a replacement rule that when applied to the expression f[x] + g[x] outputs Sin[x] + Cos[3].
2. Write a transformation rule that replaces any expression of the form Function[variable] with Cos[3].

Problem 3.2

Mathematica can solve various kinds of equations, symbolically or numerically. The result will be displayed as a list of transformation rules. We can then use the replacement operator /. (slash-dot) to apply these rules to any gven expression.

Consider now the cubic polynomial x^3+ax^2+bx+c. Use *Mathematica* to find the roots x_1, x_2 and x_3. Then use transformation rules to compute the three symmetric expressions $x_1+x_2+x_3$, $x_1x_2+x_2x_3+x_3x_1$ and $x_1x_2x_3$. Use Simplify[...] to simplify the computations. What can you conclude about the relation between the three expressions and the coefficients of the polynomial?

Problem 3.3

Consider the differential equation y' (x) = 1 + y (x).

1. Use VectorPlot[...] and StreamPlot[...] to plot the vector field of the differential equation. Try several display options (i.e. change the size of the arrows, make the picture larger, change the colors, etc.) to see which one gives the most accurate picture.
2. Next use DSolve[...] to solve the differential equation y'(x) = 1 + y(x), with initial condition y(0)=1. The result will be a list of transformation rules. Define a function YSol[x_] that returns the solution found by DSolve[...]. Evaluate YSol numerically at x=0 and x=0.1. Then plot the function YSol.
3. Check that YSol is indeed a solution of the differential equation y'(x) = 1 + y(x) by using *Mathematica* to compute the derivative of the function YSol.

4. Now go back to the picture that you have obtained in part **1**, using StreamPlot[..]. Color the solution of the differential equation y'(x) = 1+y(x) corresponding to the initial condition y(0)=1 on top of the stream plot picture.

October 13, 2014

Differential equations - an introduction

A differential equation (DE) is an equation involving a function and its derivatives. The order of a DE is the highest order of its derivatives occuring. The expression

$$F(t, y, y', \dots, y^{(n)}) = 0$$

is an ordinary differential eq. (ODE) of order n. The function $y = \phi(t)$ is a solution to this differential equation. Sometimes $y = \phi(t)$ is called an integral curve.

Example: $y''' + 2e^t y'' + yy' = t^4$ is a 3rd order ODE.

We say an ODE is linear if F is a linear function of the variables $y, y', \dots, y^{(n)}$. Otherwise it is nonlinear.

Example: $\dfrac{d^2\theta}{dt^2} + mg\sin\theta = 0$ is nonlinear 2nd order.

(1) Linear 1st order ODEs

If $g(t) = 0$ then we say the eq. is homogeneous, otherwise it is nonhomogeneous.

$$y' + p(t)\,y = g(t), \quad p(t), g(t) \text{ functions on } t.$$

There are several methods to solve these equations: separation of variables, integrating factor, exact equations, Euler method, etc.

Example: Solve the initial value problem (IVP)

$$y' = \frac{dy}{dx} = \frac{x^2}{1-y^2}, \quad y(0) = -1.1$$

and sketch its solution.

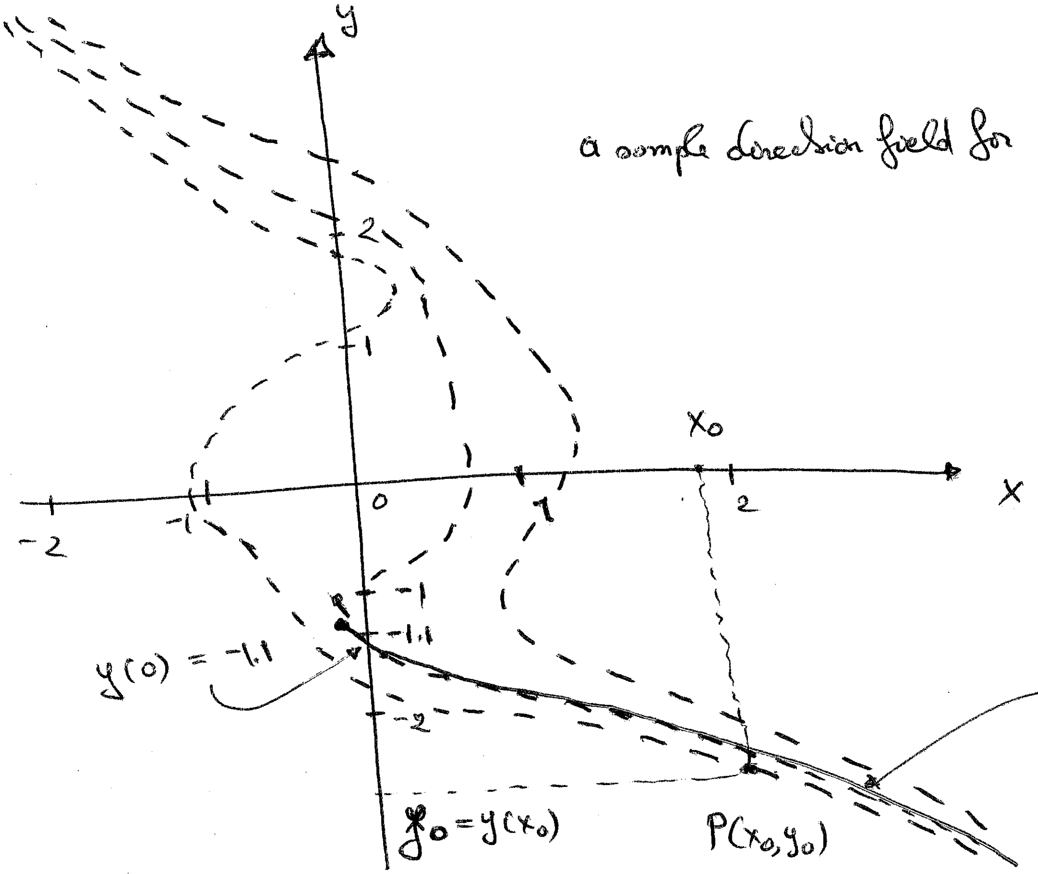
$\dfrac{dy}{dx} = \dfrac{x^2}{1-y^2}$ gives by separating variables $dy(1-y^2) = dx\,x^2$ so

$$\int (1-y^2)\,dy = \int x^2\,dx \quad \text{which gives}$$

$$y - \frac{y^3}{3} = \frac{x^3}{3} + C,$$

C is a constant that we find from $y(0) = -1.1$ so $C = -\frac{1969}{3000} \simeq -\frac{2}{3}$

So the solution is given by $3y - y^3 + 1.969 - x^3 = 0$. It is given as an implicit solution. How do we plot it? What happens with the solution as $x \to \infty$? (long time behavior) An approach would be to sketch the slope field (a direction field).



a sample direction field for $y' = \frac{x^2}{1-y^2}$,

our solution

Direction field: pick a point $P(x_0, y_0)$. Compute the slope of the tangent line at the solution passing through (x_0, y_0) $\frac{dy}{dx}\Big|_P = \frac{x_0^2}{1-y_0^2}$ and show a small vector or line segment with this slope.

Mathematica does this for us.

Example: Solve the DE

$$\frac{dy}{dt} - 2y = 4 - t \Big/ \cdot e^{-2t}$$

← multiply by the integrating factor

$$\underbrace{e^{-2t}\frac{dy}{dt} - 2e^{-2t}y}_{} = e^{-2t}(4-t)$$

product
and chain
ule

$$(e^{-2t}y)' = (4-t)e^{-2t} \quad \text{so}$$

$$\int (e^{-2t}y)' dt = \int (4-t)e^{-2t} dt$$

$$\| $$

$$e^{-2t}y = -2e^{-2t} + \frac{1}{2}te^{-2t} + \frac{1}{4}e^{-2t} + C$$

so $y = -\frac{7}{4} + \frac{1}{2}t + Ce^{2t}$ is the solution

Method of integrating factor:

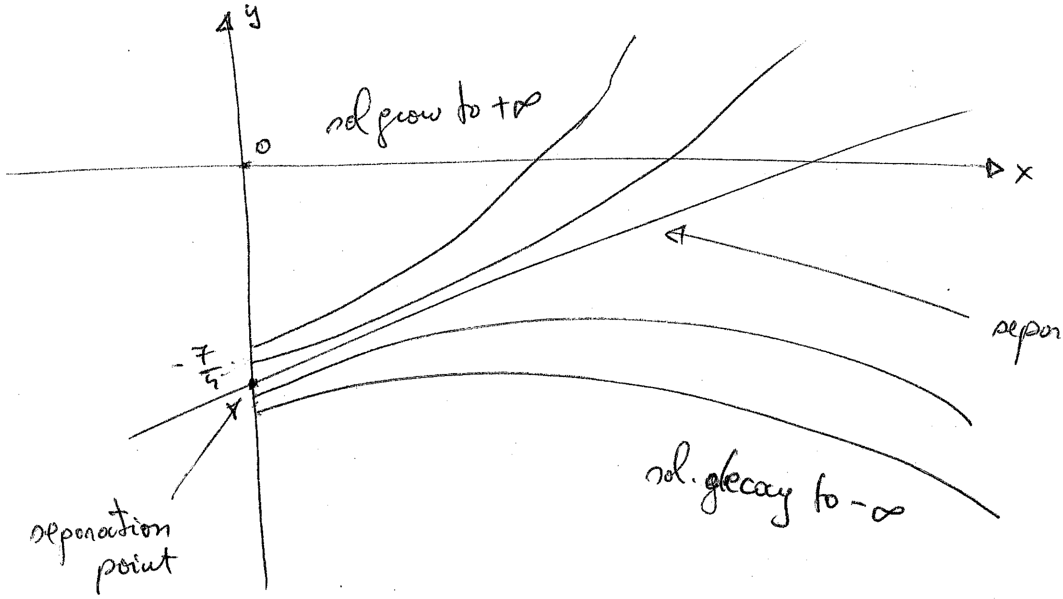
$$\frac{dy}{dt} + p(t)y = g(t)$$

multiply by $h(t) = e^{\int p(t)dt}$

$$\underbrace{h(t)\frac{dy}{dt} + h(t)p(t)y}_{} = g(t)h(t)$$

$$(y\,h(t))' = g(t)h(t)$$

so $y = \frac{1}{h(t)}\left(\int g(t)h(t)dt + C\right)$



if $y(0) = -\frac{7}{4}$ then $C = 0$ and $y = -\frac{7}{4} + \frac{1}{2}t$

Example : Solve $\underbrace{(4+t^2)y' + 2ty}_{} = 4t$

$\left((4+t^2)y\right)' = 4t$ so $(4+t^2)y = 2t^2 + C$ $\quad y = \frac{2t^2+C}{4+t^2}$

by chain rule and product rule.

② Second order linear equations

$$y'' + p(t)y' + q(t)y = g(t).$$

the initial conditions are: $y(t_0) = y_0$, $y'(t_0) = y_0'$

There are several methods to solve these: characteristic equation (if p and q are constant), reduction of order, undetermined coefficients, variation of parameters, etc.

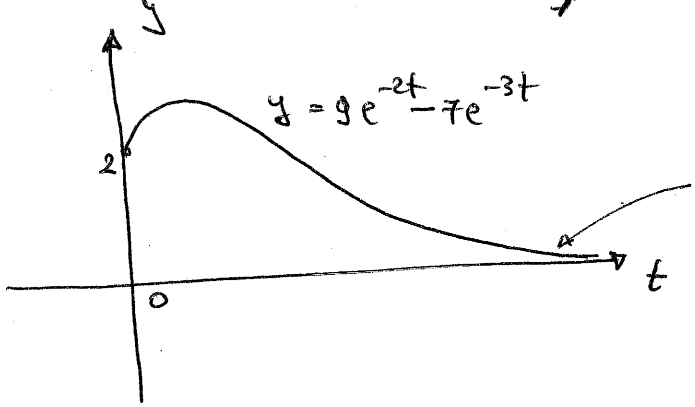
Example: Solve $y'' + 5y' + 6y = 0$ $\quad y(0) = 2$, $y'(0) = 3$

the characteristic eq. associated to this ODE is $r^2 + 5r + 6 = 0$ and $(r+2)(r+3) = 0$ has roots $r_1 = -2$, $r_2 = -3$. So the general solution to this ODE is $y = C_1e^{-2t} + C_2e^{-3t}$. We find C_1, C_2 from the initial data $y(0) = 2$, $y'(0) = 3$. and get $C_1 = 9$, $C_2 = -7$.

we get a system $\begin{cases} C_1 + C_2 = 2 \\ -2C_1 - 3C_2 = 3 \end{cases}$.

This can be solved by eliminating variables or $\underbrace{\begin{pmatrix} 1 & 1 \\ -2 & -3 \end{pmatrix}}_{A}\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$

so $\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = A^{-1}\begin{pmatrix} 2 \\ 3 \end{pmatrix}$. It is generally difficult to do by hand this way. (much easier to do with Mathematica)



we see that $y(t) \to 0$ as $t \to \infty$ so the sol. dies out.

The general theory: $y'' + ay' + by = 0$ (homogeneous)
characteristic eq is $r^2 + ar + b = 0$ has roots r_1 and r_2

$$r_{1_2} = \frac{-a \pm \sqrt{a^2-4b}}{2}.$$

There are three cases:

① $r_1 \neq r_2$ are real then $y = c_1 e^{r_1 t} + c_2 e^{r_2 t}$

② $r_1 = r_2$ real then $y = c_1 e^{r_1 t} + c_2 t e^{r_1 t}$

③ r_1, r_2 complex then $y = c_1 e^{\lambda t} \cos \mu t + c_2 e^{\lambda t} \sin \mu t$

$$\begin{aligned} r_1 &= \lambda + i\mu \\ r_2 &= \underset{\text{real part}}{\lambda} - i\underset{\text{imaginary part}}{\mu} \end{aligned}, \quad \mu \neq 0$$

③ Systems of 1st order linear equations

$$X' = AX, \quad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \text{ a } 2\times 2 \text{ matrix}$$

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ a vector}$$

Example: $x' = \begin{pmatrix} 2 & 0 \\ 0 & -3 \end{pmatrix} x$ gives $\begin{cases} x_1' = 2x_1 \\ x_2' = -3x_2 \end{cases}$ so $\begin{cases} x_1 = c_1 e^{2t} \\ x_2 = c_2 e^{-3t} \end{cases}$

so $X = \begin{pmatrix} c_1 e^{2t} \\ c_2 e^{-3t} \end{pmatrix}$.

$= c_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} e^{2t} + c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} e^{-3t}$.

The general theory is that we seek solutions of the form $X = v e^{\lambda t}$, $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ is an eigenvector and λ is an eigenvalue for the matrix A.

To find the eigenvalues we need to solve the equation (quadratic)

$$\det(A - xI_2) = x^2 - \mathrm{tr}A\, x + \det A = 0$$

If $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ then $\mathrm{tr}A = a+d$ and $\det A = ad-bc$

the solutions are $\lambda_{12} = \dfrac{\mathrm{tr}A \pm \sqrt{(\mathrm{tr}A)^2 - 4\det A}}{2}$

To find eigenvectors v corresponding to an eigenvalue λ we need to solve the system:

$$(A - \lambda I_2)\, v = 0.$$

There are three cases to consider

① if the eigenvalues λ_1, λ_2 are real and different then the solution is

$$X = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t}$$

v_1, v_2 eigenvectors for λ_1, λ_2.

② if the eigenvalues are complex $\lambda_1 = a + ib$, $\lambda_2 = a - ib$, $b \neq 0$

Let the corresponding eigenvectors be $v_1 = u + iv$, $v_2 = u - iv$, u, v are vectors too

the general sol is

$$X = c_1 e^{at}(u\cos(bt) - v\sin(bt)) + c_2 e^{at}(u\sin(bt) + v\cos(bt))$$

Remark: So even if the problem can be solved, it is more convenient to use Mathematica to plot and understand the solution.

③ eigenvalues are real and equal. $\lambda_1 = \lambda_2 = \lambda$.

(then $\mathrm{tr}A^2 = 4\det A$)

we find an eigenvector v_1 by solving $(A - \lambda I_2)v_1 = 0$ and another vector v_2 by solving $(A - \lambda I_2)v_2 = v_1$. ($v_2$ is called a generalized eigenvector)

The general solution is then

$$X = c_1 v_1 e^{\lambda t} + c_2(v_2 e^{\lambda t} + v_1 t e^{\lambda t})$$

$$= e^{\lambda t}(c_1 v_1 + c_2 v_1 t + c_2 v_2) = e^{\lambda t}(v_1(c_1 + c_2 t) + c_2 v_2)$$

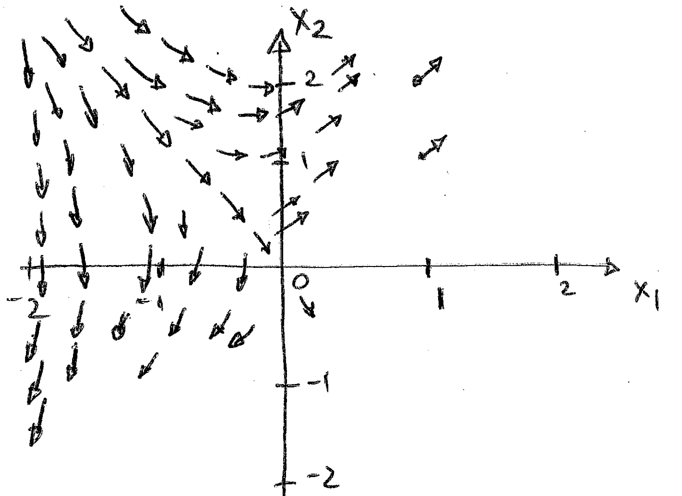
<u>Example</u>: Consider the system $X' = AX$, where $A = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix}$

Plot a direction field and determine the qualitative behaviour of solutions. Find the general solution and draw a phase portrait showing several trajectories.

Let $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$. A <u>direction field</u> (or slope field) is obtained by plotting tangent vectors to solutions in the x_1x_2-plane. We can use the chain rule

$$AX = \begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ 4x_1 + x_2 \end{pmatrix} = \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} \qquad \frac{dx_2}{dx_1} = \frac{\frac{dx_2}{dt}}{\frac{dx_1}{dt}} = \left(\frac{x_1+x_2}{4x_1+x_2}\right)^{-1} = \frac{4x_1+x_2}{x_1+x_2}$$

So the slope of the tangent at $x_1 = 1$, $x_2 = 1$ is $\left(\frac{2}{5}\right)^{-1} = \frac{5}{2}$.



Direction field sample.

By following the arrows you can see that a typical solution in the second quadrant eventually moves into the first or third quadrant.

To find the general solution, we need to find the eigenvalues and eigenvectors for the matrix A.

The x_1x_2-plane is called the <u>phase plane</u>. A qualitative understanding of the behavior of solutions can usually be gained from a direction field. A plot that includes a representative sample of trajectories of a given system is called a <u>phase portrait</u>. A qualitatively accurate phase portrait requires computer assistance.

$\operatorname{tr} A = 2$

$\det A = 1 - 4 = -3$ so the eigenvalues are solutions to

$$\lambda^2 - 2\lambda + (-3) = 0$$

$\lambda_{12} = \dfrac{2 \pm \sqrt{4 + 3\cdot 4}}{2}$ gives

$\lambda_1 = 3$, $\lambda_2 = -1$

The eigenvector v_1 corresponding to eigenvalue $\lambda_1 = 3$ satisfies

$$(A - \lambda_1 I_2)v_1 = 0 \quad \text{so} \quad \left(\begin{pmatrix} 1 & 1 \\ 4 & 1 \end{pmatrix} - \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}\right)\begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 4 & -2 \end{pmatrix}\begin{pmatrix} v_{11} \\ v_{12} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

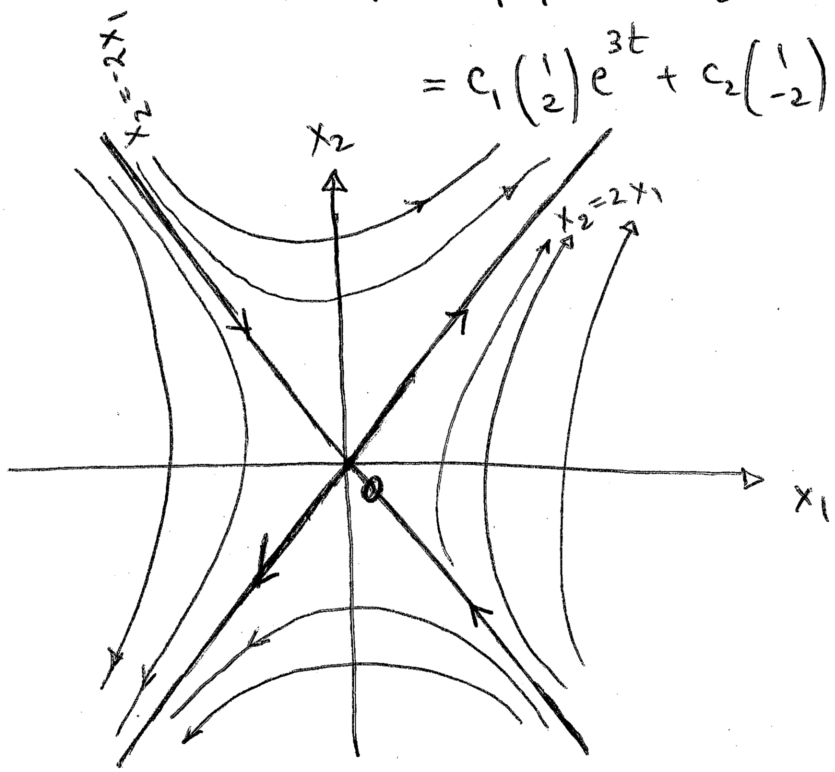
so $\left\{\begin{array}{l} -2v_{11} + v_{12} = 0 \\ 4v_{11} - 2v_{12} = 0 \end{array}\right.$ so $v_{12} = 2v_{11}$ and we can take $v_{11} = 1$, $v_{12} = 2$, which gives $v_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

Similarly the eigenvalue $\lambda_2 = -1$ has $v_2 = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$ as eigenvector.

The general solution of this system is

$$X = c_1 v_1 e^{\lambda_1 t} + c_2 v_2 e^{\lambda_2 t}$$

$$= c_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix} e^{3t} + c_2 \begin{pmatrix} 1 \\ -2 \end{pmatrix} e^{-t} = \begin{pmatrix} c_1 e^{3t} + c_2 e^{-t} \\ 2c_1 e^{3t} - 2c_2 e^{-t} \end{pmatrix} = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$



A phase portrait including the graphs of several trajectories.

If $c_2 = 0$ then
$$x_1(t) = c_1 e^{3t}$$
$$x_2(t) = 2c_1 e^{3t} = 2x_1(t)$$

So $x_2 = 2x_1$ is a special line: It is the line through the origin in the direction of the eigenvector v_1.

If $c_1 = 0$ then
$$x_1(t) = c_2 e^{-t}$$
$$x_2(t) = -2c_2 e^{-t} = -2x_1(t)$$

So $x_2 = -2x_1$ is again a special line: it is the line through the origin in the direction of v_2.

Observation: For large t the term $c_1 \begin{pmatrix} 1 \\ 2 \end{pmatrix} e^{3t}$ is dominant and $c_2 \begin{pmatrix} 1 \\ -2 \end{pmatrix} e^{-t}$ is negligible. So all solutions for which $c_1 \neq 0$ are asymptotic to the line $x_2 = 2x_1$ as $t \to \infty$. All solutions for which $c_2 \neq 0$ are asymptotic to the line $x_2 = -2x_1$ as $t \to \infty$.

The origin is called a saddle point (eigenvalues of A are real and have opposite signs) and is unstable because almost all trajectories depart from 0 as t increases (see the graph)

Example: Do a similar analysis for $X' = AX$ where

a) $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$, b) $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$, c) $A = \begin{pmatrix} -3 & \sqrt{2} \\ \sqrt{2} & -2 \end{pmatrix}$.

(4) Non-linear systems of first order equations

Example: Predator - prey equations (2D)

$$x' = x(1-y)$$
$$y' = y(x-1)$$

Mathematica can draw direction fields and solution curves.

Example: Lorenz equations (3-dimensional nonlinear)

$$\frac{dx}{dt} = \sigma(-x+y)$$
$$\frac{dy}{dt} = rx - y - xz$$
$$\frac{dz}{dt} = -bz + xy$$

$\sigma = -10$
$r = 28$
$b = -\frac{8}{3}$

we get the famous Lorenz attractor - like a butterfly. Solutions spiral towards the attractor.

Conclusions: - Linear algebra is a necessary tool to understand higher order differential equations a systems of differential equations

- Mathematica is an important tool that will help us visualize the solutions/understand their behavior easier and compute various numerical data

- many differential equations cannot be solved explicitly, so we need numerical methods to approximate them.

Solving Differential Equations

Example 1: Solve the ODE $\frac{dy}{dx} = \frac{x^2}{1-y^2}$

This ODE was solved last week using separation of variables:

$$(1 - y^2)dy = x^2 dx$$

and the general solution was found (by integration) to be

$y - \frac{y^3}{3} + c = \frac{x^3}{3}$, where c is a constant parameter.

The implicit curves

$y - \frac{y^3}{3} + c = \frac{x^3}{3}$ are called the integral curves of the differential equation.

Example 2 (Initial value problem): Solve the ODE $\frac{dy}{dx} = \frac{x^2}{1-y^2}$, with initial condition y(0)=-1.1

Otherwise said, we need to find the unique solution of the ODE that passes through the point (x0,y0)=(0,-1.1)

Given a point (x0,y0), to find the solution that passes through the point (x0,y0) means to solve the equation $y_0 - \frac{{y_0}^3}{3} + c = \frac{{x_0}^3}{3}$ and find the value of the parameter c.

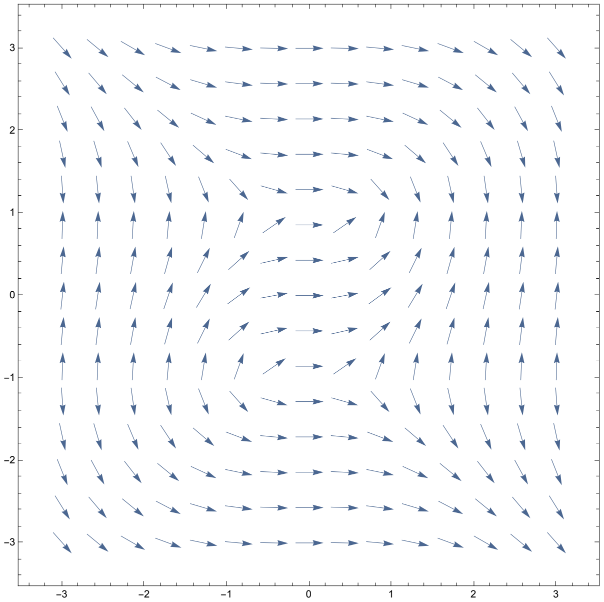
Vector field plot

Pick any point P=(x_0, y_0) and compute the tangent line at P to the solution curve passing through the point P. The slope of the tangent line at P is $m = \frac{dy}{dx}\Big|_P = \frac{{x_0}^2}{1-{y_0}^2}$, so the equation of the tangent line at P is $y - y_0 = m(x - x_0)$.

VectorPlot[{vx, vy}, {x, xmin, xmax}, {y, ymin, ymax}]

generates a vector plot of the vector field as a function of x and y.

```
VectorPlot[{1, x^2 / (1 - y^2)}, {x, -3, 3},
  {y, -3, 3}, VectorScale → {Tiny, Small, None}]
```

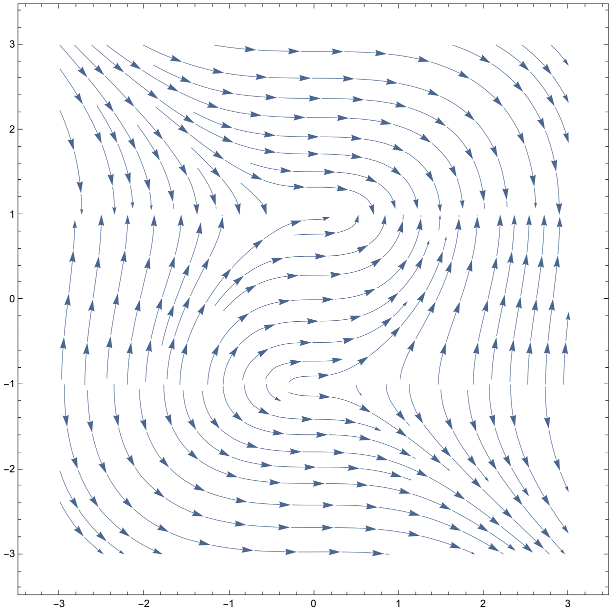


The option VectorScale determines the length and arrowhead size of field vectors that are drawn.

Stream Plot

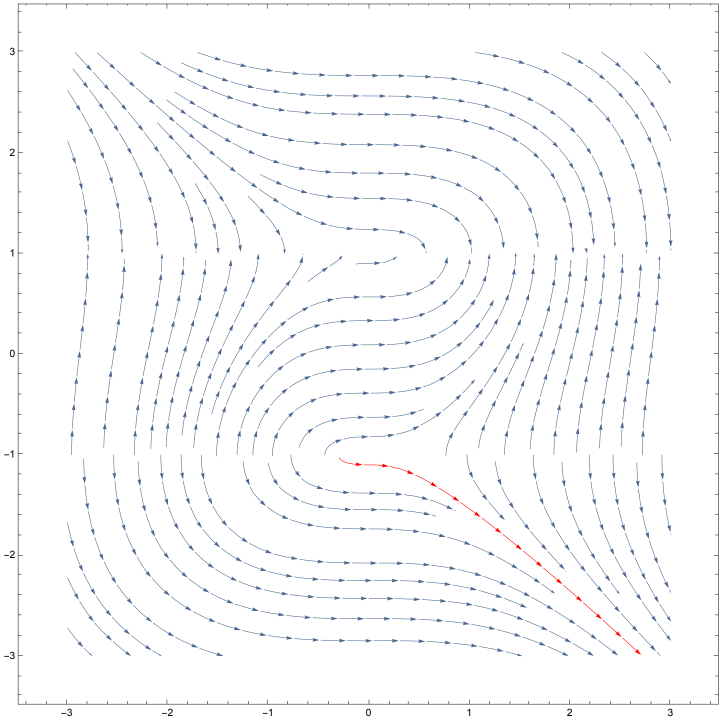
StreamPlot[{vx, vy}, {x, xmin, xmax}, {y, ymin, ymax}]
generates a stream plot of the vector field as a function of x and y.

```
StreamPlot[{1, x^2 / (1 - y^2)}, {x, -3, 3}, {y, -3, 3}]
```



Plot the integral curves of the ODE $\frac{dy}{dx} = \frac{x^2}{1-y^2}$, with the integral curve that passes through the point (0, -1.1) plotted in red.

```
StreamPlot[
 {1, x^2 / (1 - y^2)}, {x, -3, 3}, {y, -3, 3},
 StreamScale → Tiny,
 StreamPoints → {{{{0, -1.1}, Red}, Automatic}}]
```



Exact Solutions

DSolve[eqn,y,{x,xmin,xmax}]

solves a differential equation for the function y, with the independent variable x between xmin and xmax.

DSolve returns results as lists of rules. This makes it possible to return multiple solutions to an equation. For a system of equations, possibly multiple solution sets are grouped together inside curly brackets { }. You can use the rules to substitute the solutions into other calculations.

It may not always be possible to compute exact solutions of an ODE.

```
DSolve[{y'[x] == x^2 / (1 - y[x]^2), y[0] == -1.1}, y, {x, 0, 5}]
```

```
DSolve::bvnul: For some branches of the general solution, the given boundary conditions lead to an empty solution. >>
DSolve::bvnul: For some branches of the general solution, the given boundary conditions lead to an empty solution. >>
```

$$\Big\{\Big\{y \to \text{Function}\Big[\{x\},\ \Big(i\ \Big(18\, i - 18\sqrt{3} + i\, 2^{1/3}\left(53.163 - 27.\,x^3 + \sqrt{-89.6954 - 2870.8\,x^3 + 729\,x^6}\right)^{2/3} + 2^{1/3}\sqrt{3}\left(53.163 - 27.\,x^3 + \sqrt{-89.6954 - 2870.8\,x^3 + 729\,x^6}\right)^{2/3}\Big)\Big) \Big/ \Big(6 \times 2^{2/3}\left(53.163 - 27.\,x^3 + \sqrt{-89.6954 - 2870.8\,x^3 + 729\,x^6}\right)^{1/3}\Big)\Big]\Big\}\Big\}$$

Numerical Solutions

NDSolve[eqns, y, {x, xmin, xmax}]
finds a numerical solution to the ordinary differential equations eqns for the function y with the independent variable x in the range xmin to xmax. The output of NDSolve is a list of transformation rules.

```
sol1 = NDSolve[
  {y'[x] == x^2 / (1 - y[x]^2), y[0] == -1.1}, y, {x, 0, 5}]
```

```
{{y → InterpolatingFunction[ ⊞  Domain {{0., 5.}}
                                Output scalar ]}}
```

The solution to the initial value problem from Example 2 is unique. In this case, the output of NDSolve is a nested list with one transformation rule. The transformation rule describes y as a (numerically computed) function of x. Conceptually, think of the transformation rule as something like y → f[x_]

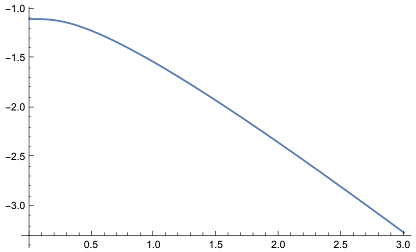
Recall : To apply a transformation rule (or a set of transformation rules) to an expression, we use the Replacement operator "/." (slash-dot).

```
y[2] /. sol1
```

```
{-2.35757}
```

We can also plot the function returned by NDSolve[..], just as we plot any other function in the Wolfram language, using the command Plot[..]. Compare the graph below to the stream line that we had plotted in red in the last StreamPlot[..].

```
Plot[y[x] /. sol1, {x, 0, 3}, PlotRange → All]
```



The command Flatten[..] removes one set of curly brackets.

```
Flatten[sol1]
y[2] /. Flatten[sol1]
```

```
{y → InterpolatingFunction[ ⊞  Domain {{0., 5.}}
                               Output scalar ]}
```

```
-2.35757
```

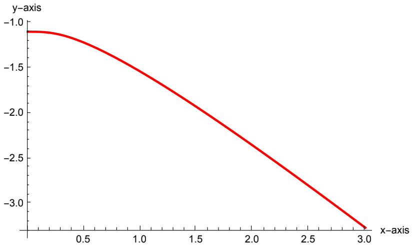
If we plan to use the function returned by NDSolve several times, it makes sense to give it a name, so we do not have to apply the replacement operator "/." (slash - dot) every time.

```
f[z_] := Module[{sol0, x, y},
  sol0 =
   NDSolve[{y'[x] ⩵ x^2 / (1 - y[x]^2), y[0] ⩵ -1.1}, y, {x, 0, 5}];
  y[z] /. Flatten[sol0]
 ]
```

```
f[2]
```

```
-2.35757
```

```
Plot[f[z], {z, 0, 3},
 AxesLabel → {"x-axis", "y-axis"}, PlotStyle → {Red, Thick}]
```



Exercise: Study the ODE $\frac{dy}{dx} = \frac{4x - x^3}{4 + y^3}$

1. Plot the vector field using **StreamPlot[...]** in the region x ∈ [-3.5, 3.5], y ∈ [-3.3, 3.3]. On the same plot, plot the tranjectory starting at the point (0, 1) in Red, and the tranjectory starting at the point (1, -1.5) in Green.
2. Find the discontinuities of the vector field using **NSolve[expr,vars].** This command attempts to find numerical approximations to the solutions of the system expr of equations or inequalities for the variables vars.
3. Use **NDSolve[...]** to solve the ODE from example 2, $\frac{dy}{dx} = \frac{4x - x^3}{4 + y^3}$, with initial condition **y(0)=1**. The range for the variable x is assumed to be [-3,3]. Plot the function that NDSolve returns, on the domain [-3,3].

MAT 331: Project 2

Project description

In this project we will use *Mathematica* to solve several problems involving differential equations. Differential equations are currently used to model a wide range of phenomena from biology, physics, chemistry, computer science, economic analysis, etc. The theory of differential equations has become an essential tool in all areas of science, particularly since computers became commonly available.

Problem 1

The fish and game department in a certain state is planning to issue hunting permits to control the deer population (one deer per permit). It is known that if the deer population falls below a certain level m, the deer will become extinct. It is also known that if the deer population rises above the carrying capacity M, the population will decrease back to M through disease and malnutrition. Consider the following model for the growth rate of the deer population P as a function of time t:

$$\frac{dP}{dt} = r\,P(M-P)\,(P-m)$$

where P is the deer population and r=0.00003 is a constant of proportionality. The values of the other parameters are M=100 and m=55.

1. Plot the vector field of the differential equation using VectorPlot[..]. Then plot the vector field using StreamPlot[..]; identify the constant solutions and color them in red using StreamPlot[..]. Try a window size of 100x130 (in the txP plane). What happens to the solutions as time t increases, t→∞? Color a couple of trajectories to illustrate the possible behaviors, then explain the plot.
2. Solve the system of differential equations for the initial condition P(20)=110. If DSolve[...] does not work, then a numeric approximation may be the next best thing to hope for, so use *Mathematica* to find a numerical approximation of the true solution, then plot it.
3. If the initial deer population is 140, about how many hunting permits should be issued so that the deer population does not become extinct?
4. Write a small interactive model using Manipulate[..] and Locator[..], that initially plots the StreamPlot[..] from part **1**, and then on click, colors the solution curve that passes through the point where the user clicked.

Problem 2

In this problem, we will visualize a famous phenomenon in dynamics of nonlinear differential equations, called the Poincaré-Andronov-Hopf bifurcation. It exhibits the birth of a limit cycle through a change in the stability of the equilibrium point. Consider the nonlinear system $\mathcal{S}$ of differential equations given below, where α is a real parameter:

$$\begin{cases} x' = y - x(x{}^\wedge 2 + y{}^\wedge 2 - \alpha) & (1) \\ y' = -x - y(x{}^\wedge 2 + y{}^\wedge 2 - \alpha) & (2) \end{cases}.$$ Its linearization

at (0, 0) is denoted by $\mathcal{L}$ and given by $\begin{cases} x' = \alpha x + y & (1) \\ y' = -x + \alpha y & (2) \end{cases}$

1. Find the equilibrium points of the nonlinear system. Find the eigenvalues of the coeffient matrix of the linearized system at the equilibrium points in terms of the parameter α.

2. Look at the linear system $\mathcal{L}$ and sssume that the value of the parameter α is -1/4.

2.0.1. Assume that the system satisfies the initial condition x(0) = -10 and y(0) = 20. Use DSolve[..] to find the solution of the corresponding initial value problem, then plot the correspoding trajectory in the xy plane.

2.0.2. For your trajectory in part **2.0.1**, draw the graphs of x versus t and y versus t on the same graph.

2.0.3. For your trajectory in part **2.0.1**, draw the corresponding graph in the three-dimensional txy-space.

*In parts **2.0.1** - **2.0.3**, it is important that you choose appropriate appropriate ranges for x,y,t and approriate scales for your plot.*

3. By using Manipulate[...] and StreamPlot[..], do an interactive model of the vector field of the system, with the parameter α taking values in the closed interval [-2, 2]. The interactive model will show the Phase Portrait of the nonlinear system $\mathcal{S}$, the Phase Portrait of the linear system $\mathcal{L}$ and the eigenvalues at the equilibrium points.

4. Analyze the type and stability of the equilibrium points of the nonlinear system $\mathcal{S}$ for all values of the parameter α. Use the interactive model to find the values of the parameter α where the qualitative nature of the solutions for the system changes.

5. Use the Interactive model to find the values of the parameter α for which the system develops limit cycles. Then prove mathematically the existence of the limit cycle by changing the system into polar coordinates and solving it.

General considerations:

No previous knowledge of differential equations is assumed is this project, except for the theory developed in the lecture notes posted on Blackboard. Please read the lab notes on Blackboard to find what *Mathematica* commands are used for solving differential equations and visualizing Phase Portraits. More examples and tutorials about solving differential equations and plotting vector fields can be found in the *Mathematica* documentation (Help → Wolfram Documentation).

Systems of Differential Equations

Solving systems of first order differential equations

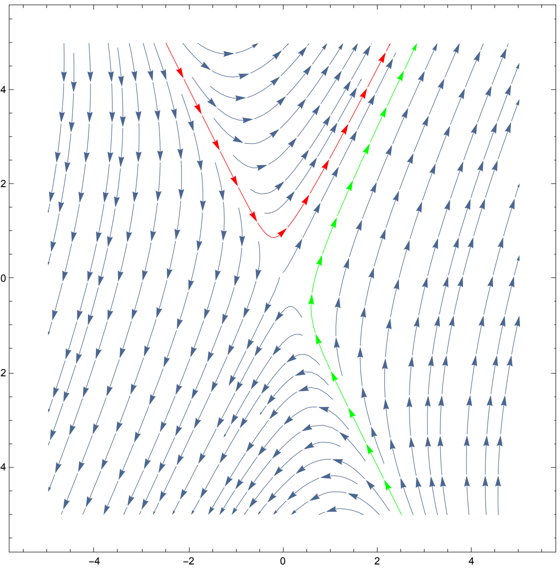
Example: Solve the system $\begin{cases} x' = x + y & (1) \\ y' = 4\,x + y & (2) \end{cases}$

The functions x and y are functions of time t and the derivatives are also taken with respect to the variable t. $\begin{cases} x'\,(t) = x(t) + y(t) & (1) \\ y'\,(t) = 4\,x(t) + y(t) & (2) \end{cases}$.

Vector field plot

The vector field plot gives valuable information about the asymptotic behaviour of solutions:

```
StreamPlot[{x + y, 4 x + y}, {x, -5, 5},
 {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{0, 1}, Red}, {{1, 1}, Green}, Automatic}}]
```



```
A = {{1, 1}, {4, 1}}; (* coefficient matrix of the system *)
Eigensystem[A]
```

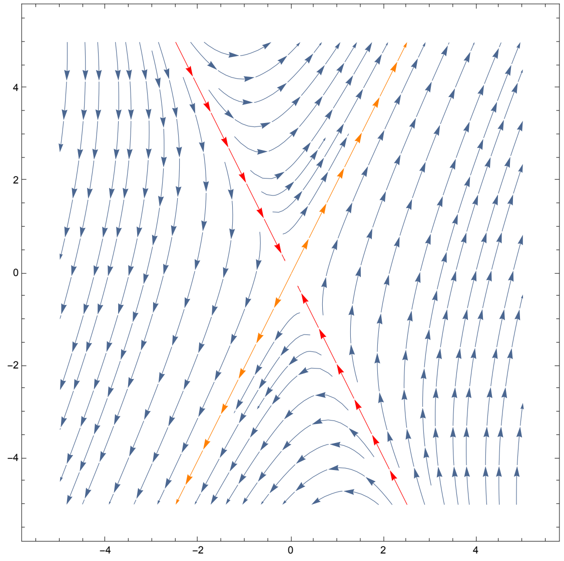
```
{{3, -1}, {{1, 2}, {-1, 2}}}
```

The eigenvalues of the coefficient matrix A are real and distinct (3 and -1), so the general solution of the system of differential equations is
$\begin{pmatrix}x(t)\\ y(t)\end{pmatrix} = c_1\, e^{3\,t}\begin{pmatrix}1\\ 2\end{pmatrix} + c_2\, e^{-t}\begin{pmatrix}-1\\ 2\end{pmatrix}$, where c_1 and c_2 are constant parameters.
There are two special solutions when c_1=0, and respectively when $c_2 = 0$. We will color these special solutions in red, and respectively in orange.

```
L = Eigenvectors[A];
Rc1  = {L[[1]], Orange}
Lc1 = {-L[[1]], Orange}
Rc2  = {L[[2]],  Red}
Lc2 = {-L[[2]], Red}
```

```
{{1, 2}, 🟧}

{{-1, -2}, 🟧}

{{-1, 2}, 🟥}

{{1, -2}, 🟥}
```

```
StreamPlot[{x + y, 4 x + y}, {x, -5, 5},
 {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{Rc1, Rc2, Lc1, Lc2, Automatic}}]
```



DSolve

DSolve[eqn,y,x]
solves a differential equation for the function y, with independent variable x.
DSolve[eqn,y,{x,xmin,xmax}]
solves a differential equation for the function y, with the independent variable x between xmin and xmax.
DSolve[{eqn1,eqn2,...},{y1,y2,...},...]
solves a list of differential equations for y1, y2, ...

DSolve returns results as lists of rules. This makes it possible to return multiple solutions to an equation. For a system of equations, possibly multiple solution sets are grouped together inside curly brackets { }. You can use the rules to substitute the solutions into other calculations.

How to use the output of DSolve[...]

```
sol1 = DSolve[
   {x'[t] == x[t] + y[t], y'[t] == 4 x[t] + y[t]}, {x, y}, t]
```

$$\left\{\left\{x \to \text{Function}\left[\{t\}, \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[1] + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) C[2]\right],\right.\right.$$
$$\left.\left. y \to \text{Function}\left[\{t\}, e^{-t} \left(-1 + e^{4 t}\right) C[1] + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[2]\right]\right\}\right\}$$

When the second argument of DSolve is specified as y, the solution is returned as a transformation rule y->Function[...], where the right hand side of the rule is a pure function (a function with no name). A general solution contains arbitrary parameters labeled C[i] that can be varied to produce particular solutions for the equation.

```
f = Function[{t}, 1/2 e^-t (1 + e^4t) C[1] + 1/4 e^-t (-1 + e^4t) C[2]]
 (*copy paste*)
```

$$\text{Function}\left[\{t\}, \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[1] + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) C[2]\right]$$

How to use the solution returned by DSolve[...]

```
g = x /. Flatten[sol1]
h = y /. Flatten[sol1]
```

The command Flatten[...] will get rid of the additional curly brackets. Instead of using two command lines, we can use only one:

```
{g, h} = {x, y} /. Flatten[sol1]
```

$$\left\{\text{Function}\left[\{t\}, \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[1] + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) C[2]\right],\right.$$
$$\left. \text{Function}\left[\{t\}, e^{-t} \left(-1 + e^{4 t}\right) C[1] + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[2]\right]\right\}$$

To obtain a particular solution, we can give values to the constants C[1] and C[2].

```
? C
```

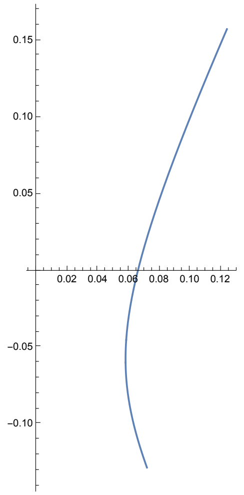
C[*i*] is the default form for the i^{th} parameter or constant generated in representing the results of various symbolic computations >>

```
{g0, h0} = {x, y} /. Flatten[sol1] /. {C[1] → 0.1, C[2] → 0.1}
```

$$\left\{\text{Function}\left[\{t\}, \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) 0.1 + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) 0.1\right],\right.$$
$$\left. \text{Function}\left[\{t\}, e^{-t} \left(-1 + e^{4 t}\right) 0.1 + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) 0.1\right]\right\}$$

We can plot the solution that we have obtained using ParametricPlot[...].

```
ParametricPlot[{g0[t], h0[t]}, {t, -1, 0.1}, AxesOrigin → {0, 0}]
```



Several ways to call DSolve[...]

```
sol2 =
 DSolve[{x'[t] == x[t] + y[t], y'[t] == 4 x[t] + y[t]}, {x[t], y[t]}, t]
```

$$\left\{\left\{x[t] \to \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[1] + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) C[2],\right.\right.$$
$$\left.\left. y[t] \to e^{-t} \left(-1 + e^{4 t}\right) C[1] + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[2]\right\}\right\}$$

We can also relabel the parameters generated by DSolve[...], see the example below and the *Mathematica* documentation on Generated Parameters.

```
sol3 = DSolve[{x'[t] == x[t] + y[t], y'[t] == 4 x[t] + y[t]},
   {x[t], y[t]}, t, GeneratedParameters → (Subscript[d, #] &)]
```

$$\left\{\left\{x[t] \to \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) d_1 + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) d_2, y[t] \to e^{-t} \left(-1 + e^{4 t}\right) d_1 + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) d_2\right\}\right\}$$

We can no longer use the same command as before to extract the solutions, {x,y} /. Flatten[sol2], because the replacement rule is now of the form x[t]->Rule, and not of the form x->Rule. Notice also that we defined the functions g2 and h2 using “=” and not “:=”.

```
Clear[t];
g2[t_] = x[t] /. Flatten[sol2]
h2[t_] = y[t] /. Flatten[sol2]
```

$$\frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[1] + \frac{1}{4} e^{-t} \left(-1 + e^{4 t}\right) C[2]$$

$$e^{-t} \left(-1 + e^{4 t}\right) C[1] + \frac{1}{2} e^{-t} \left(1 + e^{4 t}\right) C[2]$$

When an adequate number of initial conditions are specified, DSolve[...] returns particular solutions to the given equations.

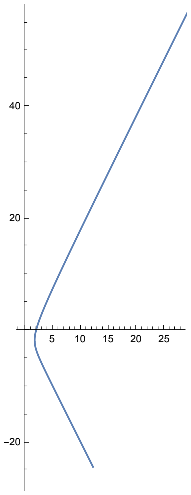
```
sol4 = DSolve[{x'[t] == x[t] + y[t],
    y'[t] == 4 x[t] + y[t], x[0] == 2, y[0] == 0}, {x[t], y[t]}, t]
```

$$\left\{\left\{x[t] \to e^{-t} \left(1 + e^{4 t}\right), y[t] \to 2 e^{-t} \left(-1 + e^{4 t}\right)\right\}\right\}$$

```
{x1[t_], y1[t_]} = {x[t] , y[t]} /. Flatten[sol4]
```

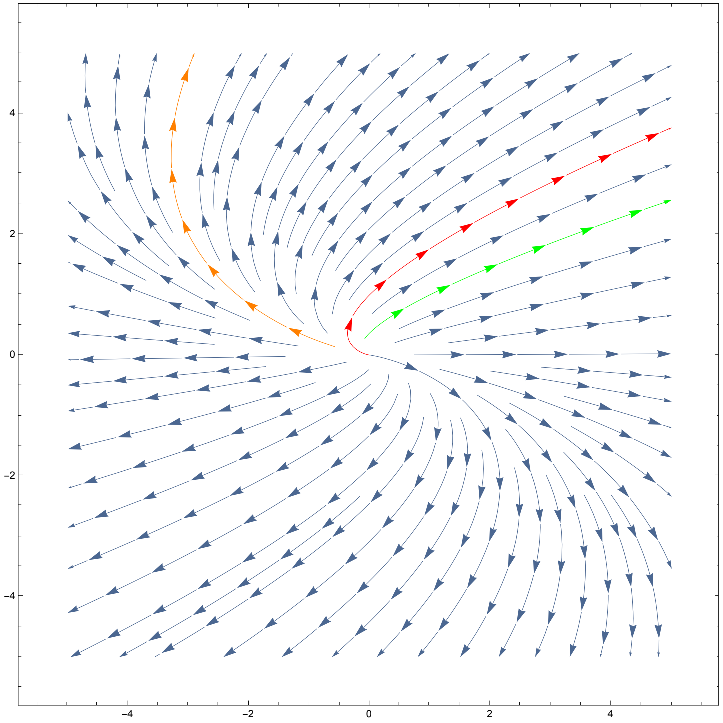
$$\left\{e^{-t} \left(1 + e^{4 t}\right), 2 e^{-t} \left(-1 + e^{4 t}\right)\right\}$$

```
ParametricPlot[{x1[t], y1[t]}, {t, -2.5, 1.5}]
```

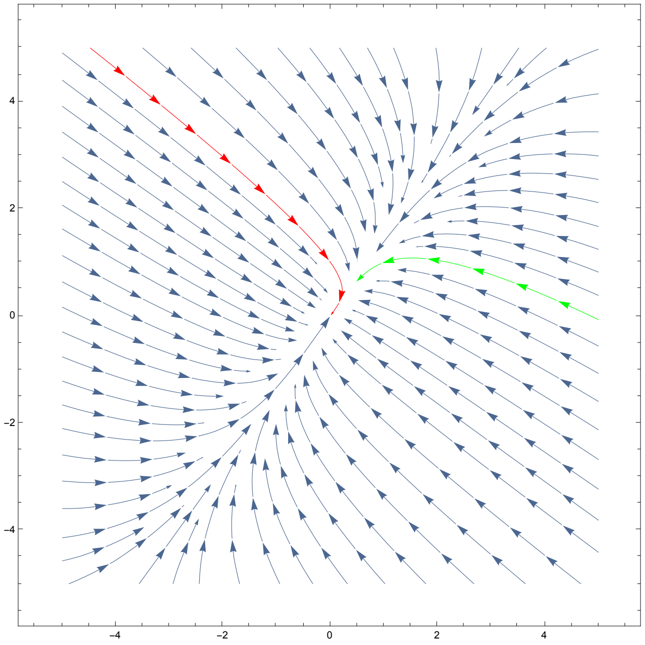


Other examples of systems of first order differential equations

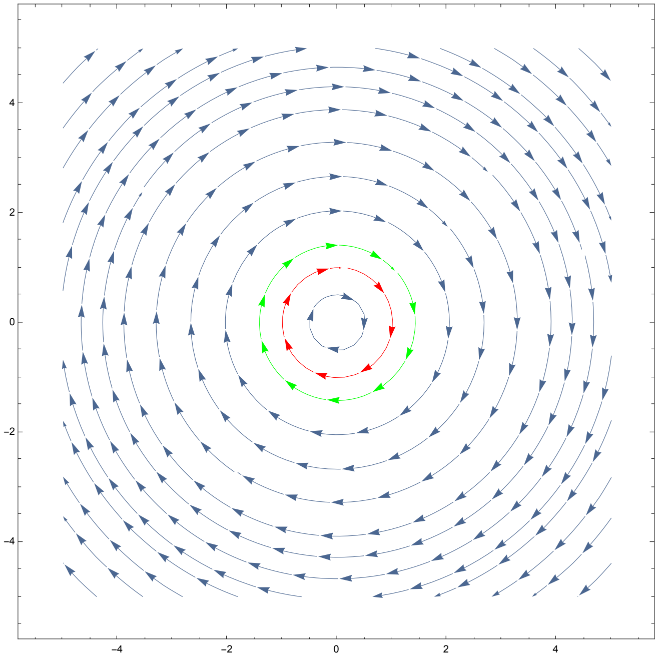
```
StreamPlot[{x + y, y}, {x, -5, 5},
 {y, -5, 5}, StreamScale → Automatic, StreamPoints →
  {{{{0, 1}, Red}, {{1, 1}, Green}, {{-3, 2}, Orange}, Automatic}}]
```



```
StreamPlot[{-3 x + √2 y, √2 x - 2 y},
 {x, -5, 5}, {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{0, 1}, Red}, {{1, 1}, Green}, Automatic}}]
```



```
StreamPlot[{y, -x}, {x, -5, 5}, {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{0, 1}, Red}, {{1, 1}, Green}, Automatic}}]
```



Exercise 1 (from last class):

Study the ODE $\dfrac{dy}{dx} = \dfrac{4x-x^3}{4+y^3}$.

1. Plot the vector field using **StreamPlot[...]** in the region $x \in [-3.5, 3.5]$, $y \in [-3.3, 3.3]$. On the same plot, plot the tranjectory starting at the point (0, 1) in Red, and the tranjectory starting at the point (1, -1.5) in Green.

2. Find the discontinuities of the vector field using **NSolve[expr,vars].** This command attempts to find numerical approximations to the solutions of the system expr of equations or inequalities for the variables vars.
3. Use **NDSolve[...]** to solve the ODE from example 2, $\dfrac{dy}{dx} = \dfrac{4x-x^3}{4+y^3}$, with initial condition **y(0)=1**. The range for the variable x is assumed to be [-3,3]. Plot the function that NDSolve returns, on the domain [-3,3].

Exercise 2 (predator-prey system):

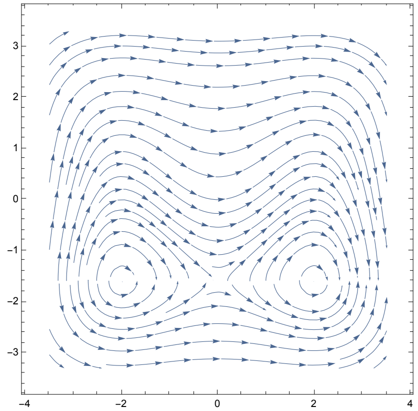
Consider the system usually described as a predator - prey model (Voltera, 1931):

$$\frac{dx}{dt} = 0.2\,x - 0.7\,xy, \quad \frac{dy}{dt} = -0.4\,y + 0.5\,xy.$$

where you can think of the function x(t) as representing a population of rabbits that naturally grow at a rate proportional to their population (that is, exponential growth in the absence of predators) and the function y(t) as representing a population of foxes that naturally decline (that is, exponential decay in the absence of prey). The term xy in both equations is proportional to the number of likely encounters of the two population in a certain environment. Encounters are detrimental to the rabbits and beneficial to the foxes.

1. Plot the vector field using **StreamPlot[...]** in the region that makes sense for the given problem. On the same plot, plot the tranjectory starting at the point (0, 1) in Red, the tranjectory starting at the point (1, 0) in Green, the trajectory starting at (1,1) in Orange, and the trajectory starting at (5, 2) in Purple.
2. What do you notice? How do the solutions look like?
3. Assume that initially there are 5 rabbits and 2 foxes. Use **NDSolve[...]** or **DSolve[...]** to find the solution corresponding to this initial condition. Try plotting the solution that NDSolve returns.

```
StreamPlot[{1, (4 x - x^3)/(4 + y^3)}, {x, -3.5`, 3.5`}, {y, -3.3`, 3.3`}]
```

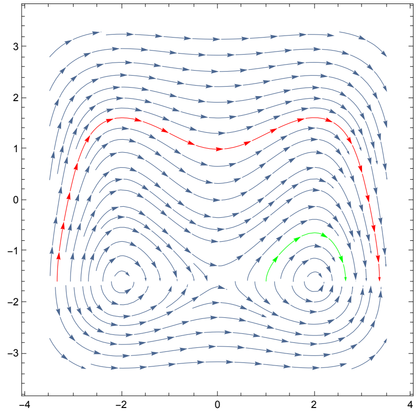


```
NSolve[4 + y^3 == 0, y]
```

```
{{y → -1.5874}, {y → 0.793701 - 1.37473 i}, {y → 0.793701 + 1.37473 i}}
```

The line y = -1.5874 on the StreamPlot represents a line of discontinuity of the vector field. Notice how the direction of the arrows changes, and tha the slopes of the integral curves are ∞ or -∞.

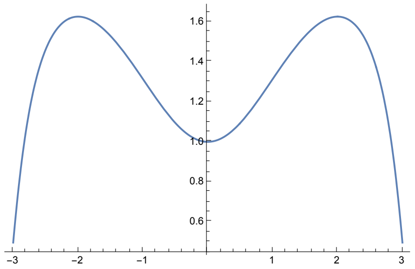
```
StreamPlot[{1, (4 x - x^3)/(4 + y^3)}, {x, -3.5`, 3.5`}, {y, -3.3`, 3.3`},
 StreamPoints → {{{{0, 1}, Red}, {{1, -1.5}, Green}, Automatic}}]
```



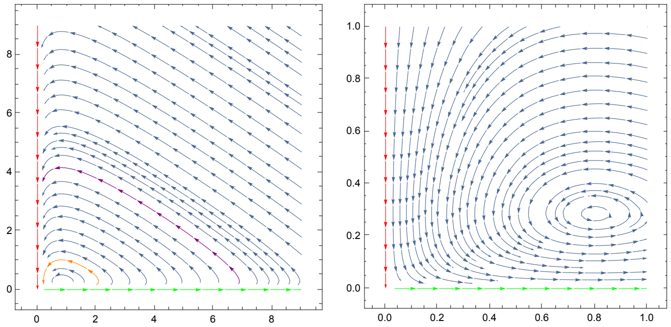
```
NDSolve[{y'[x] == (4 x - x^3) / (4 + y[x]^3), y[0] == 1}, y, {x, -3, 3}]
```

```
{{y → InterpolatingFunction[ Domain {{-3., 3.}} Output scalar ]}}
```

```
Plot[Evaluate[y[x] /. NDSolve[
    {y'[x] == (4 x - x^3) / (4 + y[x]^3), y[0] == 1}, y, {x, -3, 3}]], {x, -3, 3}]
```



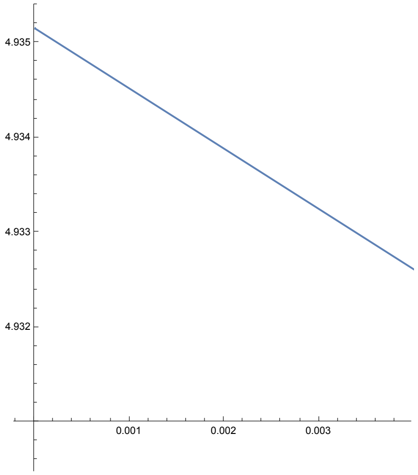
```
s1 = StreamPlot[{0.2 x - 0.7 x y, -0.4 y + 0.5 x y},
   {x, 0, 1}, {y, 0, 1}, StreamScale → Automatic,
   StreamPoints → {{{{0, 1}, Red}, {{1, 0}, Green},
      {{1, 1}, Orange}, {{5, 2}, Purple}, Automatic}}];
s2 = StreamPlot[{0.2 x - 0.7 x y, -0.4 y + 0.5 x y},
   {x, 0, 9}, {y, 0, 9}, StreamScale → Automatic,
   StreamPoints → {{{{0, 1}, Red}, {{1, 0}, Green},
      {{1, 1}, Orange}, {{5, 2}, Purple}, Automatic}}];
GraphicsRow[{s2, s1}, ImageSize → {900, 400}]
```



```
● Clear[x, y, x1, y1, t];
  sol5 = NDSolve[{x'[t] == 0.2 * x[t] - 0.7 * x[t] * y[t],
     y'[t] == -0.4 * x[t] + 0.5 * x[t] * y[t],
     x[0] == 5, y[0] == 2}, {x[t], y[t]}, {t, 1, 10}]
```

```
{{x[t] → InterpolatingFunction[ Domain {{1., 10.}} Output scalar ][t],
  y[t] → InterpolatingFunction[ Domain {{1., 10.}} Output scalar ][t]}}
```

```
{x1[t_], y1[t_]} = {x[t] , y[t]} /. Flatten[sol5];
ParametricPlot[{x1[t], y1[t]}, {t, 1, 10}]
```



Stability of Equilibrium Solutions

Equilibrium (Critical point, Steady State Solution)

Types of critical points

Stability for systems of first order linear differential equations

Exercise (predator-prey system):

Consider the system usually described as a predator - prey model (Voltera, 1931):

$$\frac{dx}{dt} = 0.2\,x - 0.7\,xy, \quad \frac{dy}{dt} = -0.4\,y + 0.5\,xy.$$

where you can think of the function x(t) as representing a population of rabbits that naturally grow at a rate proportional to their population (that is, exponential growth in the absence of predators) and the function y(t) as representing a population of foxes that naturally decline (that is, exponential decay in the absence of prey). The term xy in both equations is proportional to the number of likely encounters of the two population in a certain environment. Encounters are detrimental to the rabbits and beneficial to the foxes.

1. Plot the vector field using **StreamPlot[...]** in the region that makes sense for the given problem. On the same plot, plot the tranjectory starting at the point (0, 1) in Red, the tranjectory starting at the point (1, 0) in Green, the trajectory starting at (1,1) in Orange, and the trajectory starting at (5, 2) in Purple.
2. What do you notice? How do the solutions look like?
3. Assume that initially there are 5 rabbits and 2 foxes. Use **NDSolve[...]** or **DSolve[...]** to find the solution corresponding to this initial condition. Try plotting the solution that NDSolve returns.

Systems of first order linear differential equations

Consider a system S of linear differential equations $\begin{cases} x' = ax + by & (1) \\ y' = cx + dy & (2) \end{cases}$

The functions x and y are functions of time t and the derivatives are also taken with respect to the variable t. $\begin{cases} x'(t) = ax(t) + by(t) & (1) \\ y'(t) = cx(t) + dy(t) & (2) \end{cases}$.

Definition: The phase portrait (obtained in *Mathematica* using the command StreamPlot) of a systems of linear differential equations is a representative set of its solutions (trajectories, integral curves), plotted as parametric curves (with t as the parameter) on the Cartesian plane xy, tracing the path of each particular solution (x, y) = (x(t), y(t)), $-\infty < t < \infty$.

Similar to a vector field (obtained in *Mathematica* using the command VectorPlot), a phase portrait is a graphical tool to visualize how the solutions of a given system of differential equations would behave in the long run. We can classify the behavior of solutions and the type and stability of the equilibrium solution of a given linear system by looking at the phase portrait and analyzing the properties of the coefficient matrix A = $\left(\begin{array}{c|c} a & b \\ c & d \end{array}\right)$.

Equilibrium Solution

Definition: An equilibrium solution of the system S is a point (x,y) where x'=0 and y'=0. An equilibrium solution is a constant solution of the system, and is sometimes also called a critical point.

For our linear system of differential equations, an equilibrium solution occurs at each solution of the system (of homogeneous algebraic equations) ax+by=0 and cx+dy=0. Equivalently, we need to solve the matrix equation $A\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. As we have seen, such a system has exactly one solution, located at the origin $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$, if and only if the matrix A is invertible. If A is not invertible, then there are infinitely many solutions. Recall also that A is invertible if and only if the determinant of the matrix det(A) is different from 0 if and only if 0 is an eigenvalue of A. If det (A) = 0, then there are infinitely many solutions. The condition det(A) is equivalent to the fact that 0 is an eigenvalue of A.

Recall also that the following statements are equivalent:
A is invertible ⇔ the determinant of the matrix det (A) is different from 0 ⇔ 0 is an eigenvalue of A.

Classification and Stability of Critical Points

Assume that det(A) ≠ 0. Let λ and μ be the two eigenvalues of A.

Case I: The eigenvalues λ and μ are real and different.

As discussed in the lecture notes, the general solution of the system of linear differential euations is

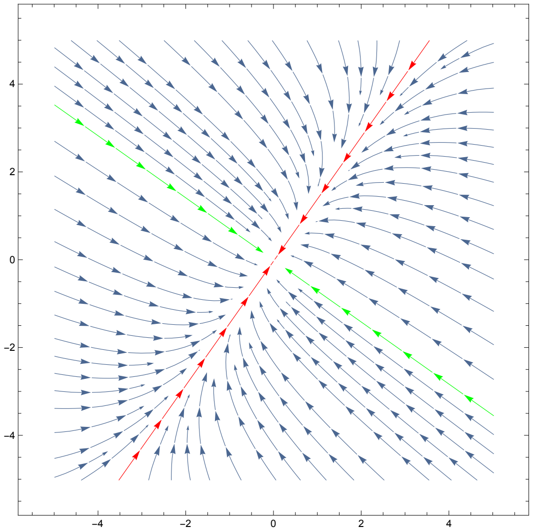
$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = c_1 \; e^{\lambda t} v + c_2 \; e^{\mu t} w,$$

where c_1 and c_2 are constant parameters, and v and w are eigenvectors corresponding to the eigenvalues λ and μ.

a) The eigenvalues λ and μ are both negative.
The origin is called a sink (attracting point, stable node, stable point, nodal sink).
All trajectories converge to 0 when t→∞.

```
A = {{-3, √2}, {√2, -2}};
Print["The eigenvalues are: ", Eigenvalues[A]]
Print["The eigenvectors are: ", Eigenvectors[A]]

StreamPlot[{-3 x + √2 y, √2 x - 2 y},
 {x, -5, 5}, {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{1/√2, 1}, Red}, {{-1/√2, -1}, Red},
    {{-√2, 1}, Green}, {{√2, -1}, Green}, , Automatic}}]
```

```
The eigenvalues are: {-4, -1}

The eigenvectors are: {{-√2, 1}, {1/√2, 1}}
```

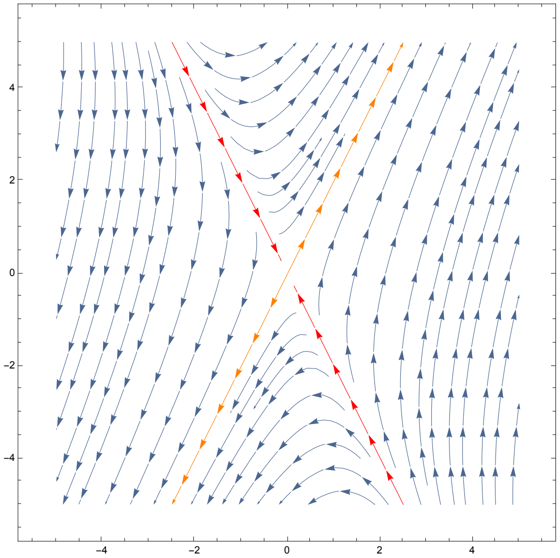


b) The eigenvalues λ and μ are both positive.
The origin is called a source (repelling point, unstable node, unstable point, nodal source). All trajectories diverge away from 0 when t→∞.

c) The eigenvalues λ and μ have different signs, say λ>0 and μ<0. The origin is called a saddle.

The trajectories given by the eigenvectors of the negative eigenvalue μ<0 initially start at infinite distance away, but converge to the origin exponentially fast (x(t),y(t)) = $c_2 \; e^{\mu t}$w. The trajectories given by the eigenvectors of the positive eigenvalue λ>0 diverge from the origin exponentially fast (x(t),y(t)) = $c_1 \; e^{\lambda t}$v. Every other trajectory moves towards the origin following the attracting direction w, but never converges to the origin, as it changes direction and diverges from the origin following the repelling direction v.

The example discussed during last class was a saddle, see the lecture notes for the code :



Case 2: The eigenvalues λ and μ are complex conjugate.

As discussed in the lecture notes, the general solution of the system of linear differential equations is

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = c_1\ e^{at}(u\cos(bt) - v\sin(bt)) + c_2\ e^{at}(u\sin(bt) + v\cos(bt)),$$

where c_1 and c_2 are constant parameters,
λ = a+ib, μ = a-ib,
and w $_\lambda$ = u+iv and w $_\mu$ = u-iv are eigenvectors corresponding to the eigenvalues λ and μ.

a) The eigenvalues λ and μ have real part Re(λ)=Re(μ)<0.

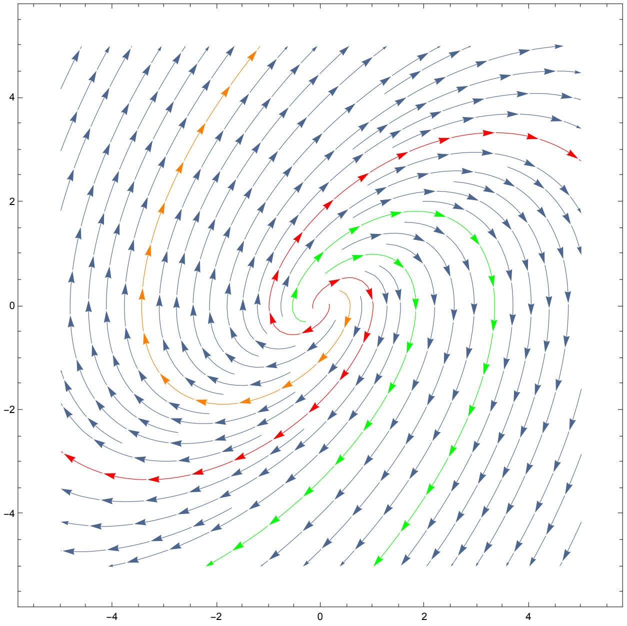
The origin is called a spiralling sink. All trajectories spiral toward 0 when t→∞.

b) The eigenvalues λ and μ have real part Re(λ)=Re(μ)>0.
The origin is called a spiralling source. All trajectories spiral away from 0 when t→∞.

```
A = {{0, 1}, {-1, 1}};
Print["The Eigenvalues are complex conjugate",
 N[Eigenvalues[A], 1], " with positive real part"]
Print["Eigenvectors: ", e = Eigenvectors[A]]

StreamPlot[{y, -x + y}, {x, -5, 5},
 {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{1, 0}, Red}, {{-1, 0}, Red}, {{0, 1}, Green},
    {{1, 1}, Green}, {{-3, 2}, Orange}, Automatic}}]
```

```
The Eigenvalues are complex conjugate
 {0.5 + 0.9 i, 0.5 - 0.9 i} with positive real part
```

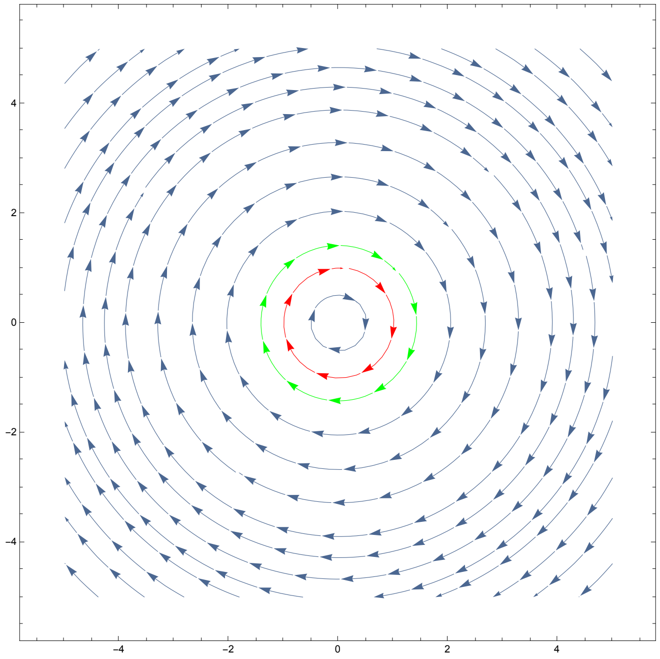
Eigenvectors: $\left\{\left\{-\frac{1}{2} i \left(i+\sqrt{3}\right), 1\right\}, \left\{\frac{1}{2} i \left(-i+\sqrt{3}\right), 1\right\}\right\}$



c) The eigenvalues λ and μ have real part Re(λ)=Re(μ)=0.
The origin is called a center. All trajectories are closed circles around the origin.

```
A = {{0, 1}, {-1, 0}};
Print["The eigenvalues are ", Eigenvalues[A]]
Eigenvectors[A]

StreamPlot[{y, -x}, {x, -5, 5}, {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{0, 1}, Red}, {{1, 1}, Green}, Automatic}}]
```

```
The eigenvalues are {i, -i}

{{-i, 1}, {i, 1}}
```



Case 3: The eigenvalues λ and μ are real and $\lambda = \mu$.

a) The eigenvalue λ has algebraic and geometric multiplicity 2 (there exists two linearly independent eigenvectors v and w of λ).

The general solution is $\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = c_1\ e^{\lambda t}\mathrm{v} + c_2\ e^{\lambda t}\mathbf{w}.$

All the theory from Case 1 holds true, and the origin is called a sink if λ<0 and respectively a source if λ>0. In the Phase Portrait, every nonzero solution looks like a straight line in the direction given by the vector $c_1\mathbf{v} + c_2\mathbf{w}$.

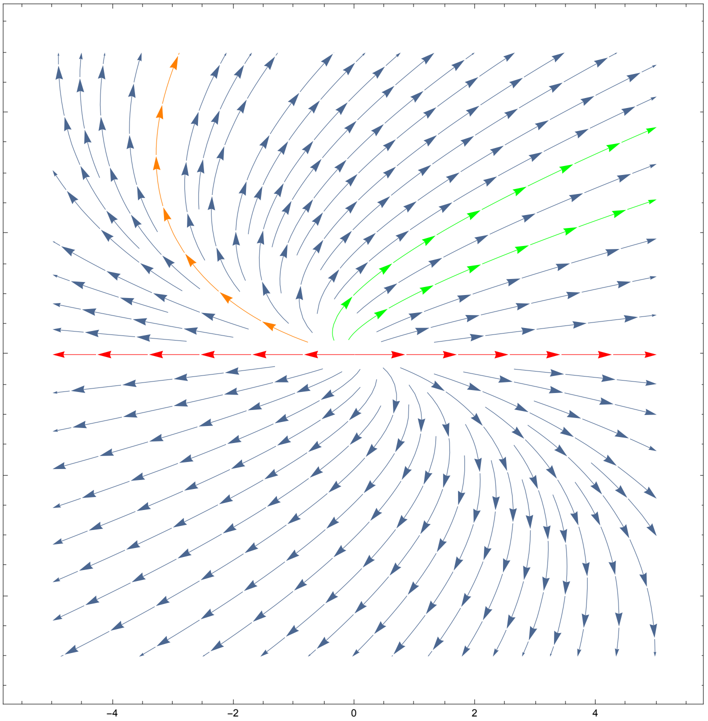
b) The eigenvalue λ has algebraic multiplicity 2 and geometric multiplicity 1.

The general solution is $\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = c_1\ e^{\lambda t}\mathrm{v} + c_2\ (e^{\lambda t}\mathrm{w} + t\ e^{\lambda t}\mathrm{v})$, where v is an eigenvector, and w is a generalized eigenvector (see the lectures notes for the definition of a generalized eigenvector).

When λ<0, the origin is called a degenerate sink. All trajectories converge to the origin. With only one linearly independent eigenvector, the Phase Portrait looks like a combination between Case 1(a) and Case 2(a). The origin is called a degenerate source when λ>0.

```
A = {{1, 1}, {0, 1}};
Print["The command Eigenvalues returns ",
 Eigenvalues[A], " so 1 is the only
   eigenvalue and it has algebraic multiplicity 2"]
Print["The command Eigenvectors returns ", e = Eigenvectors[A],
 " but the only eigenvector is ", e[[1]]]

StreamPlot[{x + y, y}, {x, -5, 5},
 {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{1, 0}, Red}, {{-1, 0}, Red}, {{0, 1}, Green},
    {{1, 1}, Green}, {{-3, 2}, Orange}, Automatic}}]
```

```
The command Eigenvalues returns {1, 1}
  so 1 is the only eigenvalue and it has algebraic multiplicity 2
The command Eigenvectors returns {{1, 0}, {0, 0}} but the only eigenvector is {1, 0}
```



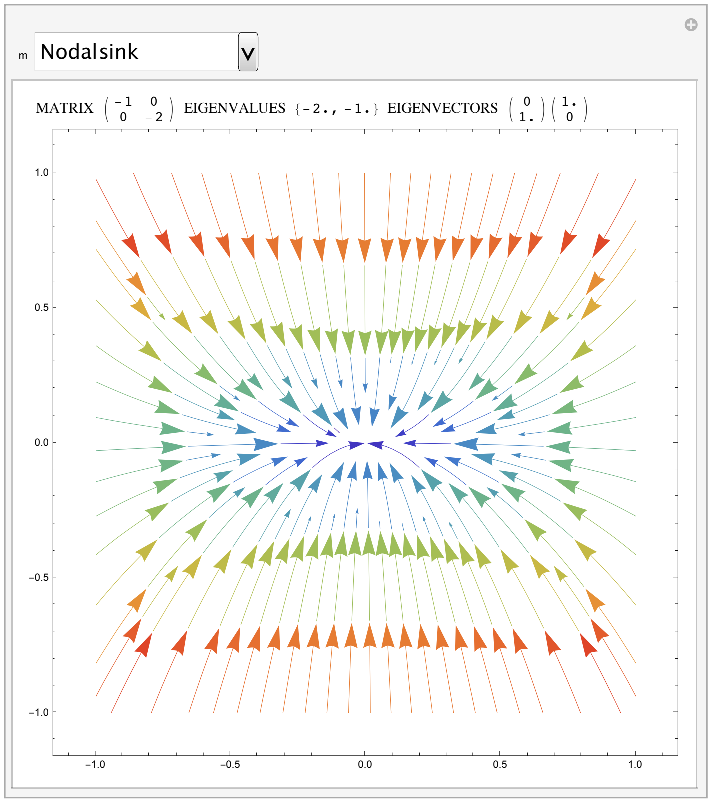
Interactive examples with StreamPlot

Interactive plot of the vector field of various systems of linear differential equations.

In[3]:=

```
Manipulate[
 Column[{
   Row[{
     Text["  MATRIX  "],
     MatrixForm[m],
     Text["  EIGENVALUES  "],
     N[Eigenvalues[m], 1],
     Text["  EIGENVECTORS  "],
     Row[{
       MatrixForm[N[Eigenvectors[m][[1]], 1]],
       MatrixForm[N[Eigenvectors[m][[2]], 1]]
      }]
     }],
   StreamPlot[m.{x, y}, {x, -1, 1}, {y, -1, 1}, StreamScale → Large,
    StreamColorFunction → "Rainbow", ImageSize → Large]
  }],
 {{m, ({{1, 0}, {0, 2}})},
  {({{1, 0}, {0, 2}}) → "Nodal source",
   ({{1, 1}, {0, 1}}) → "Degenerate source",
   ({{0, 1}, {-1, 1}}) → "Spiral source",
   ({{-1, 0}, {0, -1}}) → "Nodal sink",
   ({{-1, 0}, {0, -2}}) → "Nodal sink", ({{-1, 1}, {0, -1}}) →
    "Degenerate sink", ({{0, 1}, {-1, -1}}) → "Spiral sink",
   ({{0, 1}, {-1, 0}}) → "Center", ({{1, 0}, {0, -2}}) → "Saddle"}}
]
```

Out[3]=



MATRIX $\begin{pmatrix} -1 & 0 \\ 0 & -2 \end{pmatrix}$ EIGENVALUES {−2., −1.} EIGENVECTORS $\begin{pmatrix} 0 \\ 1. \end{pmatrix}$ $\begin{pmatrix} 1. \\ 0 \end{pmatrix}$

Phase Portraits of Nonlinear Differential Equations

Nonlinear Differential Equations:

Consider the system $\begin{cases} x' = f(x, y) & (1) \\ y' = g(x, y) & (2) \end{cases}$ where f and g are functions of two variables x and y. Assume that f and g are not both linear.

The functions x and y are functions of time t and the derivatives are also taken with respect to the variable t. $\begin{cases} x'(t) = f(x(t), y(t)) & (1) \\ y'(t) = g(x(t), y(t)) & (2) \end{cases}$.

A linear system $\begin{cases} x' = ax + by & (1) \\ y' = cx + dy & (2) \end{cases}$ with invertible coefficient matrix has only one equilibrium point (critical point, steady state solution) at (x,y)=(0,0). Analyzing the stability of this equilibrium solutions gives complete information about the Phase Portrait of the ODE (the trajectories).

Unlike a linear system, a nonlinear system could have none, one, two, three, or any number of critical points. Like a linear system, however, the critical points are found by setting x' = y' = 0, and solve the resulting system f(x,y)=0 and g(x,y)=0.

Since there might be multiple critical points present on the phase portrait, each trajectory could be influenced by more than one critical point. This results in a much more chaotic appearance of the phase portrait. Consequently, the type and stability of each critical point need to be determined locally (in a small neighborhood on the phase plane around the critical point in question) on a case-by-case basis. The trajectories will appear very differently than those of the linear systems, except very near the critical points.

General Strategy :

We approximate the nonlinear system by a linear system near the equilibirum points.
Suppose that (a, b) is an equilibrium point, that is f(a, b) = g(a, b) = 0.

$$\begin{cases} x' = f(x, y) \simeq f(a.b) + \partial_x f(a, b)\,(x - a) + \partial_y f(a, b)\,(y - b) \; = & (1) \\ \qquad \partial_x f(a, b)\,(x - a) + \partial_y f(a, b)\,(y - b) & \\ y' = g(x, y) \simeq g(a, b) + \partial_x g(a, b)\,(x - a) + \partial_y g(a, b)\,(y - b) \; = & (2) \\ \qquad \partial_x g(a, b)\,(x - a) + \partial_y g(a, b)\,(y - b) & \end{cases}$$

The critical point can be translated to 0 via the coordinate change x := x - a and y := y - b and still retains all of its properties. The approximated system becomes a linear system:

$\begin{cases} x' = \partial_x f(a, b)\, x + \partial_y f(a, b)\, y & (1) \\ y' = \partial_x g(a, b)\, x + \partial_y g(a, b)\, y & (2) \end{cases}$ with coefficient matrix $\mathbf{J}_{(a,b)} = \begin{pmatrix} \partial_x f(a, b) & \partial_y f(a, b) \\ \partial_x g(a, b) & \partial_y g(a, b) \end{pmatrix}$.

The matrix $\mathbf{J}_{(x,y)} = \begin{pmatrix} \partial_x f(x, y) & \partial_y f(x, y) \\ \partial_x g(x, y) & \partial_y g(x, y) \end{pmatrix}$ is called the Jacobian matrix of the system. It contains the first order partial derivatives of f and g evaluated at the point (x,y).

We can use the eigenvalues of the matrix Jacobian matrix to decide the type and stability of the critical point (a,b). Let λ and μ be the two eigenvalues. Assume that λ and μ are different from 0.

Eigenvalues	Linear System	Nonlinear Syste
λ, μ Real	□	□
$\lambda > \mu > 0$	Nodal Source (Unstable)	Nodal Source (Unstab
$\lambda < \mu < 0$	Nodal Sink (Stable)	Nodal Sink (Stable)
$\lambda > 0 > \mu$	Saddle point (Unstable)	Saddle point (Unstabl
$\lambda = \mu > 0$	Degenerate Source or Nodal Source (Unstable) depending of the geometric multiplicity of λ	Source (Degenerate, Nodal, Spiral Sou depending on the nonlinear terms and the
$\lambda = \mu < 0$	Degenerate Sink or Nodal Sink (Stable) depending of the geometric multiplicity of λ	Sink (Degenerate, Nodal, Spiral Sink) (S depending on the nonlinear terms and the

λ, μ Complex	□	□
Re (λ) > 0	Spiral Source (Unstable)	Spiral Source (Unstab
Re (λ) < 0	Spiral Sink (Stable)	Spiral Sink (Stable)
Re (λ) = 0	Center (Stable)	Center, Spiral Sink, Spiral Source (Stability cannot be determined based

Check also the Bifurcation Diagram on Wikipedia.

Hyperbolic critical point

Definition: The equilibrium is said to be hyperbolic if all eigenvalues of the Jacobian matrix have non - zero real parts.

Hyperbolic equilibria are robust : Small perturbations do not change qualitatively the phase portrait near the equilibria. Moreover, local phase portrait of a hyperbolic equilibrium of a nonlinear system is equivalent to that of its linearization. This statement has a mathematically precise form known as the Hartman - Grobman Theorem.

Examples of nonlinear differential equations:

Example 1. Solve the system $\begin{cases} x' = x(1-y) & (1) \\ y' = y(x-1) & (2) \end{cases}$

Usually an explicit solution cannot be found, but you can still ask Mathematica to find a numeric solution using the NDSolve command.

```
DSolve[{x'[t] == x[t] (1 - y[t]),
  y'[t] == y[t] (x[t] - 1), x[0] == 1, y[0] == 1}, {x[t], y[t]}, t]
```

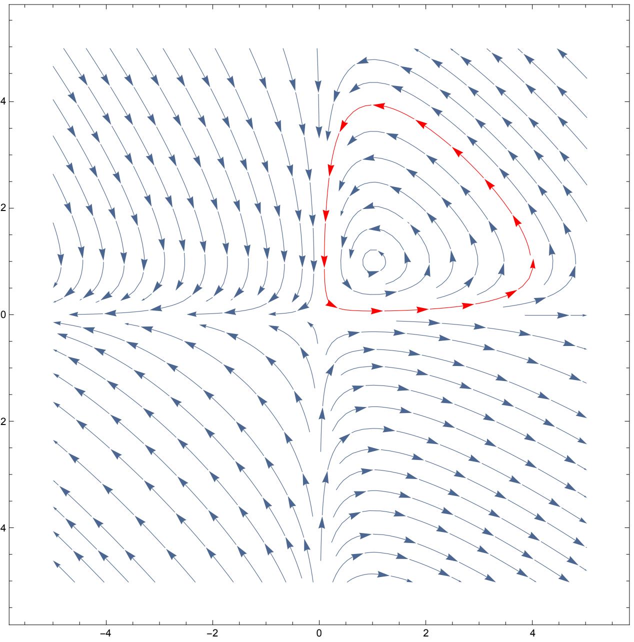
```
Solve::ifun:
  Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information. >>
Solve::ifun:
  Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information. >>
Solve::ifun:
  Inverse functions are being used by Solve, so some solutions may not be found; use Reduce for complete solution information. >>
General::stop: Further output of Solve::ifun will be suppressed during this calculation. >>
DSolve::bvnul: For some branches of the general solution, the given boundary conditions lead to an empty solution. >>
{ }
```

```
NDSolve[{x'[t] == x[t] (1 - y[t]), y'[t] == y[t] (x[t] - 1),
  x[0] == 2, y[0] == 1}, {x[t], y[t]}, {t, 0, 10}]
```

```
{{x[t] → InterpolatingFunction[ ⊞  Domain {{0., 10.}}
                                    Output scalar  ][t],
  y[t] → InterpolatingFunction[ ⊞  Domain {{0., 10.}}
                                    Output scalar  ][t]}}
```

The most reliable information here comes from the Vector Field Plot.

```
StreamPlot[{x (1 - y), y (x - 1)},
 {x, -5, 5}, {y, -5, 5}, StreamScale → Automatic,
 StreamPoints → {{{{4, 1}, Red}, Automatic}}]
```



What is the nature of the equilibrium solutions?

1. Set x' = 0 and y' = 0. In this case, it is easy to solve the equation by hand: x(1-y)=0 and y(x-1)=0 implies (x,y)=(0,0) or (x,y)=(1,1). For more complicated cases, we will use Solve[..] or NSolve[..]

```
EqPoints = Solve[{x (1 - y) == 0, y (x - 1) == 0}, {x, y}]
```

```
{{x → 0, y → 0}, {x → 1, y → 1}}
```

2. Next we find the Jacobian matrix of the system. In this case, it is easy to compute it by hand:

$$J_{(x,y)} = \left(\begin{array}{c|c} \partial_x\,(x\,(1-y)) & \partial_y\,(x\,(1-y)) \\ \partial_x\,(y\,(x-1)) & \partial_y\,(y\,(x-1)) \end{array}\right) = \left(\begin{array}{c|c} 1-y & -x \\ y & x-1 \end{array}\right)$$

In more complicated cases, we will ask Mathematica to compute the Jacobian matrix for us, as follows :

The commands **D[f[x], {x}]** or **f'[x]** give the dervative of f with respect to x. If f is a function of several variables, **D[f[x, y], {x}]** or ∂_y **(f[x, y])** gives the partial derivative of f with respect to x. Try them out!

```
∂y (x y + y^2)
```

```
D[y^2, {y}]
```

The following code computes the Jacobian matrix:

```
Clear[f, J, x, y]
f[x_, y_] := {x (1 - y), y (x - 1)}
J[f_] :=
 Module[{A = Table[0, 2, 2]},
  A[[1, 1]] = D[f[x, y][[1]], {x}];
  A[[1, 2]] = D[f[x, y][[1]], {y}];
  A[[2, 1]] = D[f[x, y][[2]], {x}];
  A[[2, 2]] = D[f[x, y][[2]], {y}];
  Return[A]]
J[f]
```

```
{{1 - y, -x}, {y, -1 + x}}
```

3. We evaluate the Jacobian matrix at the two equilibrium points and find its eigenvalues:

```
B = J[f] /. Flatten[EqPoints[[1]]];
Print["The Jacobian Matrix at the equilibrium point (0,0) is ",
 MatrixForm[B]]
Print["The eigenvalues of B are ", Eigenvalues[B]]
```

The Jacobian Matrix at the equilibrium point (0,0) is $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

```
The eigenvalues of B are {-1, 1}
```

```
T = J[f] /. Flatten[EqPoints[[2]]];
Print["The Jacobian Matrix at the equilibrium point (1,1) is ",
 MatrixForm[ T]]
Print["The eigenvalues are ", Eigenvalues[T]]
```

The Jacobian Matrix at the equilibrium point (1,1) is $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$

```
The eigenvalues are {i, -i}
```

By analyzing the Real and Imaginary Part of the Eigenvalues, we conclude that the point (0, 0) is a saddle (the Jacobian matrix has two real eigenvalues, one smaller than 0, one greater than 0) whereas the point (1, 1) is a center (the Jacobian matrix has two complex conjugate eigenvalues with Real part equal to 0, and the Phase Portrait depicts closed circular trajectories). This gives us information about the shape of trajectories in a vecinity of the equilibrium solutions.

Example 2:

Consider the system $\begin{cases} x' = y - x(x\hat{}2 + y\hat{}2) & (1) \\ y' = -x - y(x\hat{}2 + y\hat{}2) & (2) \end{cases}$. Study the nature of the critical point (x,y)=(0,0).

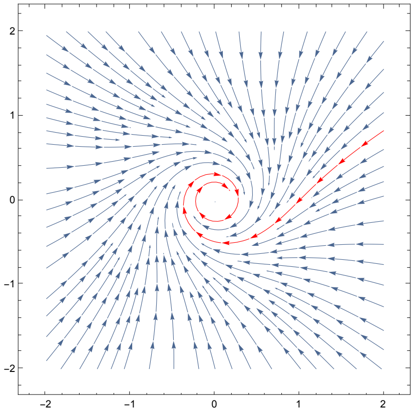
```
Clear[h]
h[x_, y_] := {y - x (x^2 + y^2), -x - y (x^2 + y^2)}
NSolve[{y - x (x^2 + y^2) == 0, -x - y (x^2 + y^2) == 0}, {x, y}]
MatrixForm[J[h]]
Eigenvalues[J[h] /. {x → 0, y → 0}]
```

```
{{x → 0., y → 0.}}
```

$$\begin{pmatrix} -3x^2 - y^2 & 1 - 2xy \\ -1 - 2xy & -x^2 - 3y^2 \end{pmatrix}$$

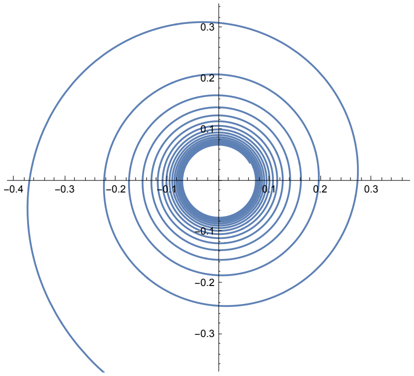
```
{i, -i}
```

```
StreamPlot[{y - x (x^2 + y^2), -x - y (x^2 + y^2) }, {x, -2, 2},
 {y, -2, 2}, StreamPoints → {{{{1, 0}, Red}, Automatic}}]
```



```
sol = NDSolve[{x'[t] == y[t] - x[t] (x[t]^2 + y[t]^2),
   y'[t] == -x[t] - y[t] (x[t]^2 + y[t]^2),
   x[0] == 1, y[0] == 0}, {x[t], y[t]}, {t, 0, 1000}]
ParametricPlot[{x[t], y[t]} /. Flatten[sol],
 {t, 0, 100}, PlotPoints → 100]
```

```
{{x[t] → InterpolatingFunction[ ⊞ Domain {{0., 1.00× 10^3}}
                                   Output scalar ] [t],

  y[t] → InterpolatingFunction[ ⊞ Domain {{0., 1.00× 10^3}}
                                   Output scalar ] [t]}}
```



```
∂x (y - x (x^2 + y^2)) + ∂y (-x - y (x^2 + y^2))
```

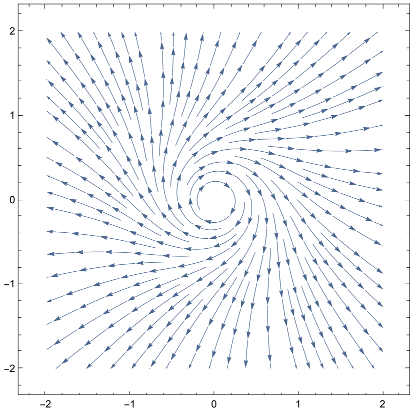
$-4\,x^2 - 4\,y^2$

```
Clear[h]
h[x_, y_] := {y + x (x^2 + y^2), -x + y (x^2 + y^2)}
MatrixForm[J[h]]
Eigenvalues[J[h] /. {x → 0, y → 0}]
```

$$\begin{pmatrix} 3\,x^2 + y^2 & 1 + 2\,x\,y \\ -1 + 2\,x\,y & x^2 + 3\,y^2 \end{pmatrix}$$

```
{i, -i}
```

```
StreamPlot[{y + x (x^2 + y^2), -x + y (x^2 + y^2) },
 {x, -2, 2}, {y, -2, 2}]
```



Conclusions : The linearized system has a center at the origin. The nonlinear system does not have a center.

Example 3

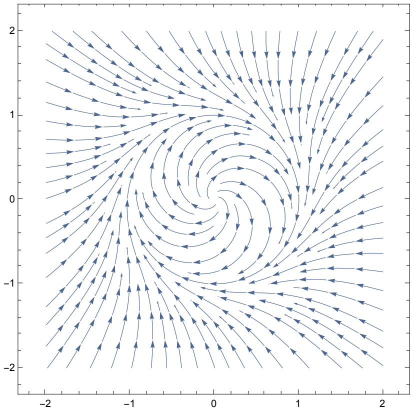
Consider the system $\begin{cases} x' = x+y-x(x\hat{}2+y\hat{}2) & (1) \\ y' = -x+y-y(x\hat{}2+y\hat{}2) & (2) \end{cases}$. Find all critical points and study their nature.

```
Clear[h]
h[x_, y_] := {x + y - x (x^2 + y^2), -x + y - y (x^2 + y^2)}
NSolve[{y - x (x^2 + y^2) == 0, -x - y (x^2 + y^2) == 0}, {x, y}]
MatrixForm[J[h]]
Eigenvalues[J[h] /. {x → 0, y → 0}]
StreamPlot[h[x, y], {x, -2, 2}, {y, -2, 2}]
```

```
{{x → 0., y → 0.}}
```

$$\begin{pmatrix} 1-3x^2-y^2 & 1-2xy \\ -1-2xy & 1-x^2-3y^2 \end{pmatrix}$$

```
{1 + i, 1 - i}
```



Conclusions : The origin is an spiral source (unstable spiral point) for both the linear system and the nonlinear system. One may therefore think that all trajectories different from 0 should spiral out to infinity. This is clearly not the case, as far away from the origin, the arrows are pointing inward. Moreover, there is a closed trajectory (a circle) and all trajectories different from (0,0) spiral towards this circle (limit cycle). You can find the equation of this limit cycle by switching to polar coordinates x=rcosθ, y=rsinθ.

The literature on the existence and properties of limits cycles is very rich:
Poincare - Bendixon Theorem
Suppose f and g have continuous partial derivatives.
a) Every closed trajectory must enclose at least one critical point. If it encloses exactly one critical point, then this cannot be a saddle point.
b) If $\partial_x f + \partial_y g$ has the same sign in a disk D, then there can be no closed trajectory of the system lying entirely inside D.
c) Suppose that U is a connected open subset that contains no critical point of the system. Let $\overline{U}$ denote the set U together with its boundary. If the vector field in $\overline{U}$ points towards the interior of U, then U contains a closed trajectory.
d) Suppose that U is a connected open subset that contains no critical point of the system. Let $\overline{U}$ denote the set U together with its boundary. If there exists some trajectory that never leaves the set U (is trapped inside U), then U must necessarily contain a closed trajectory.

Exercise :

Consider the system $\begin{cases} x' = y & (1) \\ y' = -y - 2\sin(x) & (2) \end{cases}$. Find all critical points (x,y) of the system which belong to the region -2π < x,y < 2π. Study the nature of these critical points (type, stability). Use only *Mathematica* commands, even when computing the solutions!

Phase Portraits of Nonlinear Differential Equations

Nonlinear Differential Equations:

Consider the system $\begin{cases} x' = f(x, y) & (1) \\ y' = g(x, y) & (2) \end{cases}$ where f and g are functions of two variables x and y.

The critical points (equlibrium points, steady state solutions) are found by setting x' = y' = 0,
and solving the resulting system f(x, y) = 0 and g(x, y) = 0.

We approximate the nonlinear system by a linear system near the equilibirum points.
Suppose that (a, b) is an equilibrium point, that is f(a, b) = g(a, b) = 0. The linearized system is:

$\begin{cases} x' = \partial_x f(a, b)\, x + \partial_y f(a, b)\, y & (1) \\ y' = \partial_x g(a, b)\, x + \partial_y g(a, b)\, y & (2) \end{cases}$ whose coefficient matrix is the Jacobian matrix $\mathbf{J}_{(a,b)} = \left(\begin{array}{c|c} \partial_x f(a, b) & \partial_y f(a, b) \\ \partial_x g(a, b) & \partial_y g(a, b) \end{array} \right)$.

We can use the eigenvalues of the matrix Jacobian matrix to decide the type and stability of the critical point (a,b) (sink, source, saddle, etc.).

Jacobian Matrix

The following code from last class helps compute the Jacobian matrix of a function f of two variables x and y:

```
In[1]:= Clear[f, J, x, y]
        (* Jacobian *)
        J[f_] :=
         Module[{A = Table[0, 2, 2]},
          A[[1, 1]] = D[f[x, y][[1]], {x}];
          A[[1, 2]] = D[f[x, y][[1]], {y}];
          A[[2, 1]] = D[f[x, y][[2]], {x}];
          A[[2, 2]] = D[f[x, y][[2]], {y}];
          Return[A]]
        (* Example *)
        f[x_, y_] := {x (1 - y), y (x - 1)}
        J[f]

Out[4]= {{1 - y, -x}, {y, -1 + x}}
```

Example 2:

Consider the system $\begin{cases} x' = y - x(x\hat{}2 + y\hat{}2) & (1) \\ y' = -x - y(x\hat{}2 + y\hat{}2) & (2) \end{cases}$. Study the nature of the critical point (x,y)=(0,0).

```
In[5]:= Clear[h]
        (* Define the function h *)
        h[x_, y_] := {y - x (x^2 + y^2), -x - y (x^2 + y^2)}

        (* Find the critical points of the system *)
        sol = Solve[{y - x (x^2 + y^2) == 0, -x - y (x^2 + y^2) == 0}, {x, y}];
        Print["Critical point: ", sol]

        (*Find the Jacobian matrix *)
        Print["The Jacobian Matrix is ", MatrixForm[J[h]]]

        (* Evaluate the Jacobian matrix
         at the only critical point (0,0) *)
        A = J[h] /. Flatten[sol];
        Print["The Jacobian Matrix at (0,0) is ", MatrixForm[A]]
        Print["Eigenvalues: ", Eigenvalues[A]]
```

Critical point: {{x → 0, y → 0}}

The Jacobian Matrix is $\begin{pmatrix} -3x^2 - y^2 & 1 - 2xy \\ -1 - 2xy & -x^2 - 3y^2 \end{pmatrix}$

The Jacobian Matrix at (0,0) is $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

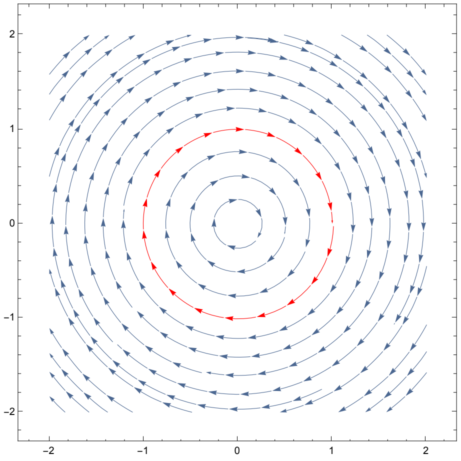
Eigenvalues: {i, -i}

The only critical point for the nonlinear system is (0, 0). The eigenvalues of the Jacobian matrix at (0, 0) are i and - i (complex conjugate with Real part equal to 0), so the origin is a center for the linearized system. Let's see if (0,0) is a center for the nonlinear system as well.

In[13]:=

```
(* Phase Portrait for the linear system *)
StreamPlot[{y, -x }, {x, -2, 2}, {y, -2, 2},
 StreamPoints → {{{{1, 0}, Red}, Automatic}}]
```

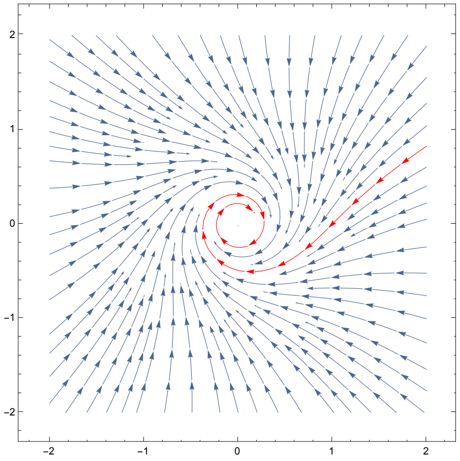
Out[13]=



In[14]:=

```
(* Phase Portrait for the nonlinear system *)
StreamPlot[{y - x (x^2 + y^2), -x - y (x^2 + y^2) }, {x, -2, 2},
 {y, -2, 2}, StreamPoints → {{{{1, 0}, Red}, Automatic}}]
```

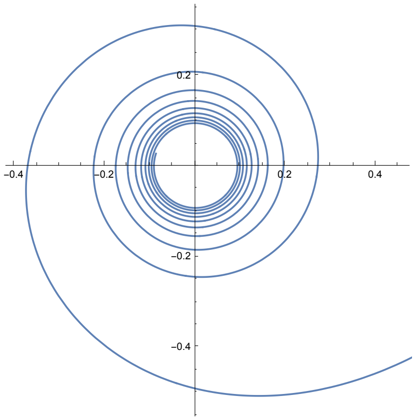
Out[14]=



```
In[19]:= (* Find the trajectory that passes through the point (1,0) *)
         sol = NDSolve[{x'[t] == y[t] - x[t] (x[t]^2 + y[t]^2),
            y'[t] == -x[t] - y[t] (x[t]^2 + y[t]^2),
            x[0] == 1, y[0] == 0}, {x[t], y[t]}, {t, 0, 1000}]
         ParametricPlot[{x[t], y[t]} /. Flatten[sol],
          {t, 0, 60}, PlotPoints → 100]
```

```
Out[19]= {{x[t] → InterpolatingFunction[ Domain {{0., 1.00×10^3}}
                                         Output scalar ][t],

           y[t] → InterpolatingFunction[ Domain {{0., 1.00×10^3}}
                                         Output scalar ][t]}}
```

Out[20]=



```
In[21]:= (* Poincare-Bendixon criterion *)
         ∂x (y - x (x^2 + y^2)) + ∂y (-x - y (x^2 + y^2))
```

Out[21]= $-4x^2 - 4y^2$

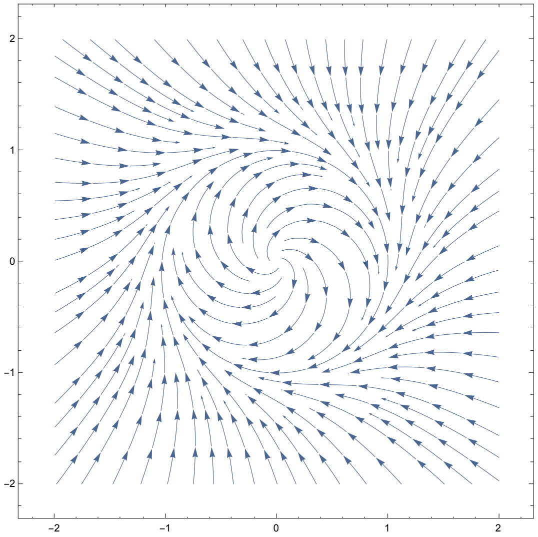
Conclusions : The linearized system has a center at the origin. The nonlinear system does not have a center! The origin is a spiral sink.

Example 3

Consider the system $\begin{cases} x' = x+y-x(x^2+y^2) & (1) \\ y' = -x+y-y(x^2+y^2) & (2) \end{cases}$. Find all critical points and study their nature.

```
In[22]:= Clear[h]
         h[x_, y_] := {x + y - x (x^2 + y^2), -x + y - y (x^2 + y^2)}

         (* Find the critical points *)
         NSolve[{y - x (x^2 + y^2) == 0, -x - y (x^2 + y^2) == 0}, {x, y}]

         MatrixForm[J[h]];
         Eigenvalues[J[h] /. {x → 0, y → 0}]
         StreamPlot[h[x, y], {x, -2, 2}, {y, -2, 2}]

Out[24]= {{x → 0., y → 0.}}

Out[26]= {1 + i, 1 - i}
```

Out[27]=



Conclusions : The origin is an spiral source (unstable spiral point) for both the linear system and the nonlinear system. One may therefore think that all trajectories different from 0 should spiral out to infinity. This is clearly not the case, as far away from the origin, the arrows are pointing inward. Moreover, there is a closed trajectory (a circle) and all trajectories different from (0,0) spiral towards this circle (limit cycle). You can find the equation of this limit cycle by switching to polar coordinates x=rcosθ, y=rsinθ.

Polar Coordinates

i. We do the coordinate change:

```
x = r Cos[θ] , y = r Sin [θ] , 0 ≤ r < ∞, 0 ≤ θ < 2 π
```

where r and θ are functions of time t.

ii. We need to compute x' and y' as functions of r, θ, r', θ'

a) We can do it by hand, using the Chain Rule. We get **x' = r'Cos[θ] - r Sin[θ]θ'** and **y' = r'Sin[θ] + r Cos[θ]θ'**
b) We use transformations rules in *Mathematica* to do the change of variables:

```
x'[t] /.  x → (r[#] Cos[θ[#]] &)
y'[t] /.  y → (r[#] Sin[θ[#]] &)
```

```
Cos[θ[t]] r′[t] - r[t] Sin[θ[t]] θ′[t]

Sin[θ[t]] r′[t] + Cos[θ[t]] r[t] θ′[t]
```

Changing the system from Example 3 to polar coordinates

We know that $x^2 + y^2 = r^2$, so can use the symmetries of the system $\begin{cases} x' = x + y - x\left(x^2 + y^2\right) & (1) \\ y' = -x + y - y\left(x^2 + y^2\right) & (2) \end{cases}$

to create more terms of the form $x^2 + y^2$ that can be easily “translated” into r^2. Therefore, we can look at the following system:-

$$\begin{cases} xx' + yy' = & (1) \\ \quad x\left(x + y - x\left(x^2 + y^2\right)\right) + y\left(-x + y - y\left(x^2 + y^2\right)\right) = x^2 + y^2 - \left(x^2 + y^2\right)\left(x^2 + y^2\right) = r^2\left(1 - r^2\right) & \\ yx' - xy' = y\left(x + y - x\left(x^2 + y^2\right)\right) - x\left(-x + y - y\left(x^2 + y^2\right)\right) = x^2 + y^2 = r^2 & (2) \end{cases}$$

Let' s see how the left hand sides xx’ + yy’ and yx’ - xy’ look like in polar coordinates:

```
In[28]:= rule = {x → (r[#] Cos[θ[#]] &),  y -> (r[#] Sin[θ[#]] &) };
         Simplify[x[t] x'[t] + y[t] y'[t] /.  rule]
         Simplify[y[t] x'[t] - x[t] y'[t] /. rule]
```

```
Out[29]= r[t] r′[t]

Out[30]= -r[t]^2 θ′[t]
```

In polar coordinates, the system becomes:

$\begin{cases} r\,r' = r^2\left(1 - r^2\right) & (1) \\ -r^2\,\theta' = r^2 & (2) \end{cases}$ which can be simplified to $\begin{cases} r' = r\left(1 - r^2\right) & (1) \\ \theta' = -1 & (2) \end{cases}$

Analyze the system written in polar coordinates :

The equation $\boldsymbol{\theta' = -1}$, solves to
$\boldsymbol{\theta(t) = -t + c}$, where c is any real constant.

The critical points of $r' = r\left(1 - r^2\right)$ are:
r=0 (the origin of the old phase portrait) and
r=1 (the unit circle in the old phase portrait).

Therefore, a periodic solution of the system is
$\boldsymbol{r(t) = 1,\ \theta(t) = -t + c}$.
(As time t increases, a point moves clockwise on the unit circle)

From the equation $r' = r\left(1 - r^2\right)$, we also see that:
r’ > 0 when r<1 (thus inside the unit circle the trajectories go outwards)
r’ < 0 when r>1 (thus outside the unit circle the trajectories are directed inwards)

What about the other trajectories?
When r ≠ 1 and r ≠ 0, we can use separation of variables to solve the equation: $\dfrac{\mathbf{dr}}{\mathbf{dt}} = r\left(1 - r^2\right)$.
We write $\dfrac{\mathbf{dr}}{r\left(1 - r^2\right)} = \mathbf{dt}$ and we can integrate each term separately.

```
In[31]:= Integrate[1 / (r (1 - r^2)), r]
```

```
Out[31]= Log[r] - 1/2 Log[1 - r^2]
```

The equation becomes $\log(r) - \frac{1}{2}\log\left(1 - r^2\right) = t + k$, where k is any real constant, with solution given by $r(t) = \frac{e^{k+t}}{\sqrt{1+e^{2\,(k+t)}}}$

In[32]:=
```
solr = Simplify[Solve [Log[r] - 1/2 Log[1 - r^2]  == t + k, r]]
Print[
 "The function r is always positive, so the solution is: ", r /. solr[[2]]]
```

Out[32]= $\left\{\left\{r \to -\frac{e^{k+t}}{\sqrt{1+e^{2\,(k+t)}}}\right\}, \left\{r \to \frac{e^{k+t}}{\sqrt{1+e^{2\,(k+t)}}}\right\}\right\}$

The function r is always positive, so the solution is: $\frac{e^{k+t}}{\sqrt{1+e^{2\,(k+t)}}}$

Poincare - Bendixon Theorem

The literature on the existence and properties of limits cycles is very rich:

Poincare - Bendixon Theorem
Suppose f and g have continuous partial derivatives.
a) Every closed trajectory must enclose at least one critical point. If it encloses exactly one critical point, then this cannot be a saddle point.
b) If $\partial_x f + \partial_y g$ has the same sign in a disk D, then there can be no closed trajectory of the system lying entirely inside D.
c) Suppose that U is a connected open subset that contains no critical point of the system. Let $\overline{U}$ denote the set U together with its boundary. If the vector field in $\overline{U}$ points towards the interior of U, then U contains a closed trajectory.
d) Suppose that U is a connected open subset that contains no critical point of the system. Let $\overline{U}$ denote the set U together with its boundary. If there exists some trajectory that never leaves the set U (is trapped inside U), then U must necessarily contain a closed trajectory.
e) Every trajectory is attracted by a fixed point, or it goes to infinity, or it accumulates on a limit cycle.

This theorem has no analogue in higher dimensions. In systems with phase space of dimension strictly greater than 2, more things can happen: trajectories may be attracted to complex geometrical objects (strange attractors), which leads to chaotical behavior.

Nonlinear Equations with several variables :

In 1963, Lorenz studied a very simple model of the atmosphere. This model exhibits chaotic behavior:

Lorenz equation

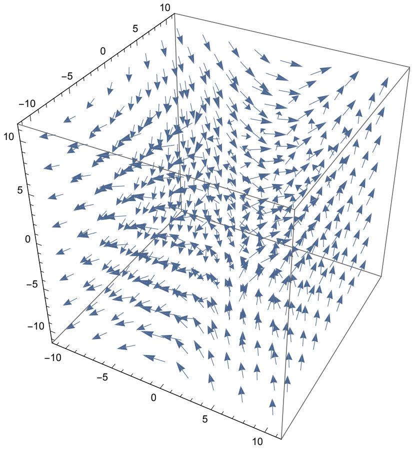
In[34]:=
```
tmax = 20;
sol = NDSolve[
   {x'[t] == -10 (x[t] - y[t]), y'[t] == -x[t] z[t] + 28 x[t] - y[t],
    z'[t] == x[t] y[t] - 8/3 z[t], x[0] == 30, y[0] == 10, z[0] == 40},
   {x, y, z}, {t, 0, tmax}, MaxSteps → 5000];
```

The 3D vector field can be plotte using the command VectorPlot3D

In[36]:=

```
VectorPlot3D[{-10 (x - y), -x * z + 28 x - y, x * y - 8/3 z},
 {x, -10, 10}, {y, -10, 10}, {z, -10, 10},
 VectorScale → {Small, Automatic, None}]
```

Out[36]=

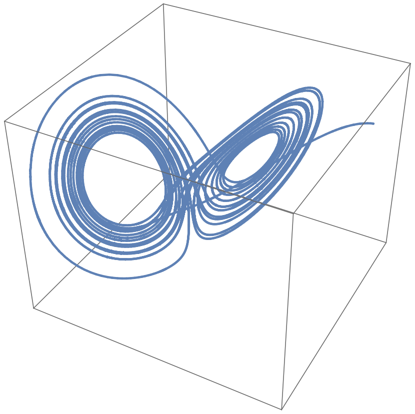


The solution can the be plotted using ParametricPlot3D as follows:

In[37]:=

```
Clear[x, y, z, t]
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. sol],
  {t, 0, tmax}, PlotPoints → 3000, Axes → False, PlotRange → All]
```

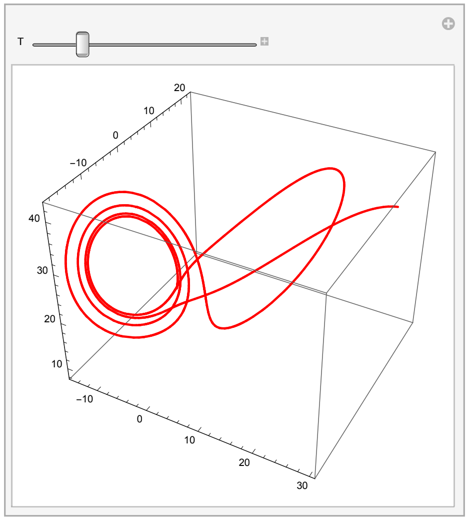
Out[38]=



In[39]:=

```
Clear[x, y, z, t]
Manipulate[
 ParametricPlot3D[
  Evaluate[{x[t], y[t], z[t]} /. NDSolve[
      {x'[t] == -10 (x[t] - y[t]), y'[t] == -x[t] z[t] + 28 x[t] - y[t],
                                  8
       z'[t] == x[t] y[t] - ─ z[t], x[0] == 30, y[0] == 10, z[0] == 40},
                                  3
      {x, y, z}, {t, 0, T}, MaxSteps → 5000]],
  {t, 0, T}, PlotStyle → Red],
 {{T, 2}, 0.1, 20}, SynchronousUpdating → False,
 SaveDefinitions → True]
```

Out[40]=



Interactive Models with Stream Plot and Locator

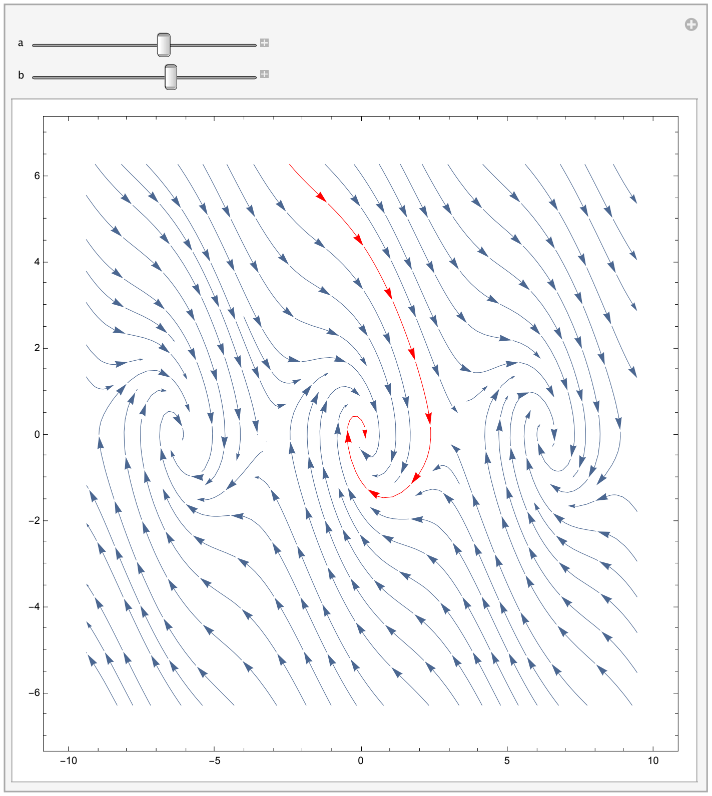
Exercise

Consider the system $\begin{cases} x' = y & (1) \\ y' = -y - 2\sin(x) & (2) \end{cases}$.

Make an interactive plot of the Phase Portrait when -3π < x < 3π , -2π < y < 2π, which can highlight the trajectory corresponding to the initial condition chosen by the user.

Solution I

```
Manipulate[
 StreamPlot[{y, -2 Sin[x] - y},
  {x, -3 π, 3 π}, {y, -2 π, 2 π}, ImageSize → Large,
  StreamPoints → {{{{a, b}, Red}, Automatic}}],
 {a, -3 π, 3 π},
 {b, -2 π, 2 π}
]
```

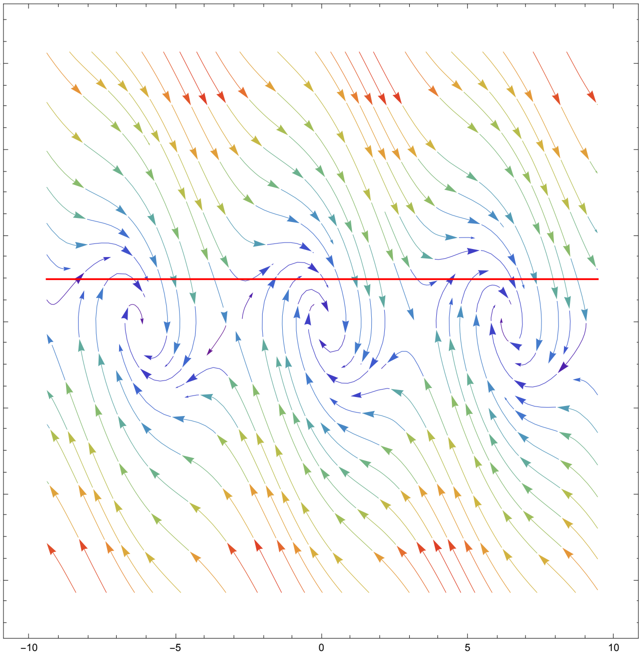


The disadvantage of solution I is that every time we drag the slider to change the initial condition, the entire phase portrait gets re - plotted, not just the trajectory selected by the user.

Solution 2

Show[g1, g2, …]
shows several graphics combined.

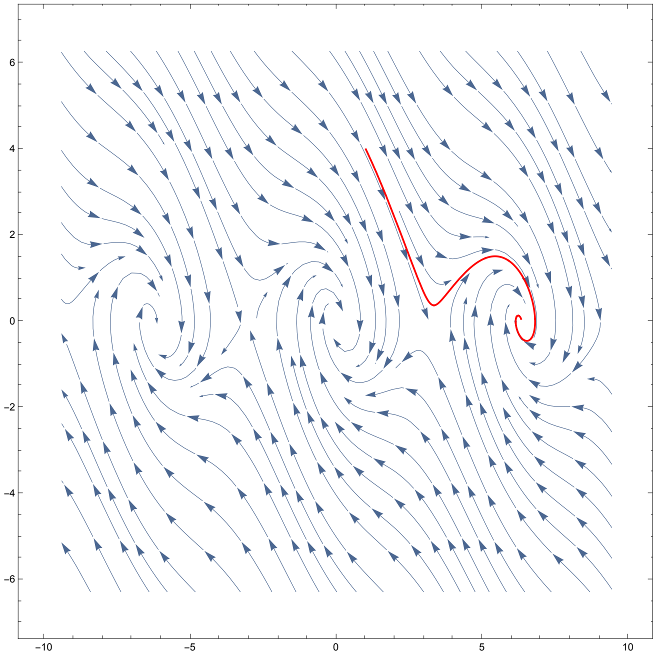
```
splot = StreamPlot[{y, -2 Sin[x] - y}, {x, -3 π, 3 π}, {y, -2 π, 2 π},
    ImageSize → Large, StreamColorFunction → "Rainbow"];
line = Plot[1, {x, -3 π, 3 π}, PlotStyle → Red];
Show[splot, line]
```



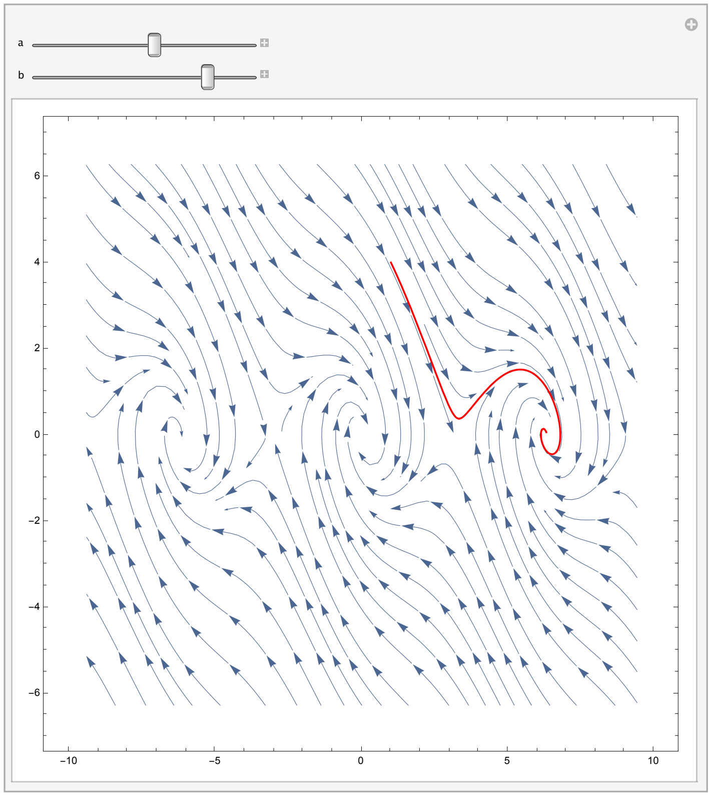
In[46]:=
```
a = 1; b = 4;
splot = StreamPlot[{y, -2 Sin[x] - y},
   {x, -3 π, 3 π}, {y, -2 π, 2 π}, ImageSize → Large];
sol = NDSolve[{x'[t] == y[t], y'[t] == -2 Sin[x[t]] - y[t],
   x[0] == a, y[0] == b}, {x, y}, {t, 0, 10}]
trajectory = ParametricPlot[Evaluate[{x[t], y[t]} /. sol],
   {t, 0, 10}, PlotStyle → Red];
Show[splot, trajectory]
```

Out[48]=
```
{{x → InterpolatingFunction[ ⊞ Domain {{0., 10.}}
                                Output scalar ],
  y → InterpolatingFunction[ ⊞ Domain {{0., 10.}}
                                Output scalar ]}}
```

Out[50]=



```
Manipulate[
 sol = NDSolve[{x'[t] == y[t], y'[t] == -2 Sin[x[t]] - y[t],
    x[0] == a, y[0] == b}, {x, y}, {t, 0, 10}];
 trajectory = ParametricPlot[Evaluate[{x[t], y[t]} /. sol],
   {t, 0, 10}, PlotStyle → Red];
 Show[splot, trajectory],
 {a, -3 π, 3 π }, {b, -2 π, 2 π} ,
 SaveDefinitions → True]
```



Solution 3

Locator[{x,y}] represents a locator object at position (x,y) in a graphical object.

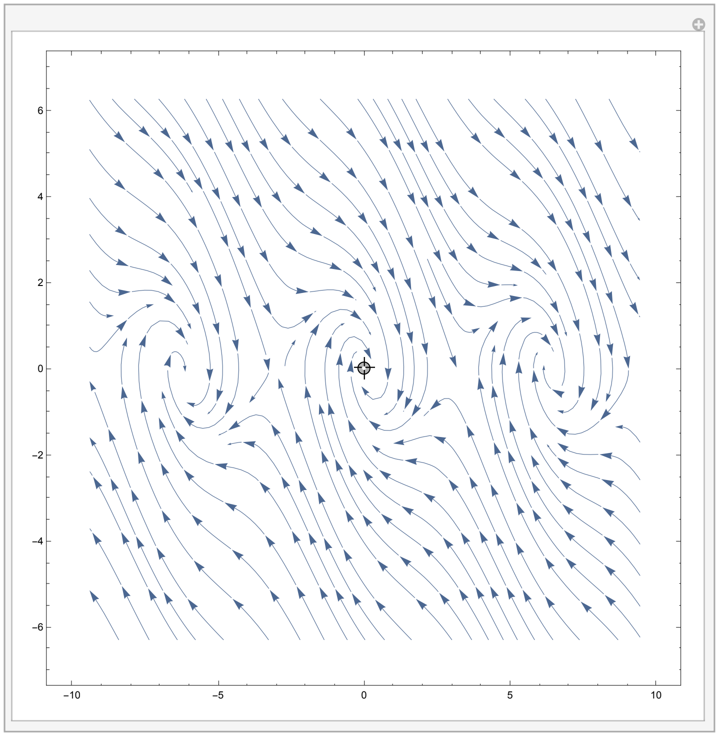
Locator[Dynamic[pos]]
takes the position to be the dynamically updated current value of pos, with this value being reset if the locator object is moved.

```
Graphics[ Text["some text", {0, 7}]]
```

Let' s write an interactive model that shows the previous StreamPlot, with a dynamic variable called point, which is linked to Locator.

```
Manipulate[
 Show[splot],
 {point, Locator},
  SaveDefinitions → True]
```

Out[51]=

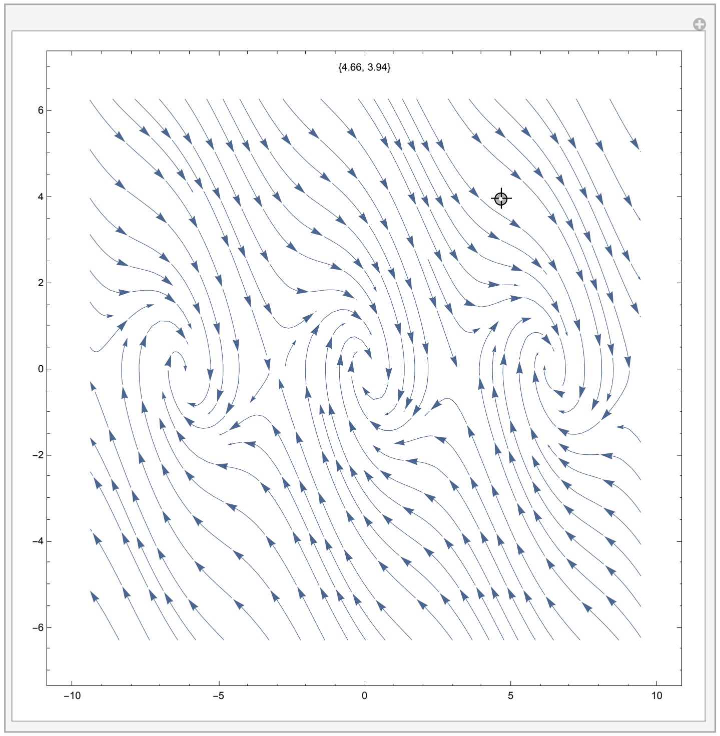


Let' s first see what Locator can do.

In[54]:=

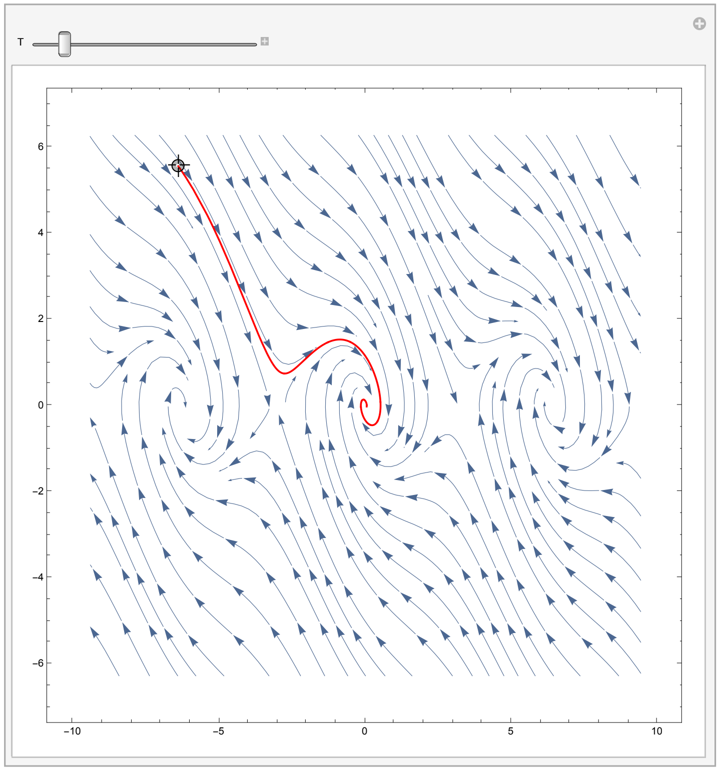
```
Manipulate[
 Show[splot, Graphics[Text[point, {0, 7}]]],
 {{point, {2, 3}}, Locator},
  SaveDefinitions → True
]
```

Out[54]=



Let's use Locator inside the interactive model. When the user clicks on a point in the Phase Portrait, we solve the differential equation corresponding to the initial condition selected by the user, and highlight the corresponding trajectory in the Phase Portrait. We can include also a slider for the time t (that controls how much of the trajectory we should plot)

```
splot = StreamPlot[{y, -2 Sin[x] - y},
   {x, -3 π, 3 π}, {y, -2 π, 2 π}, ImageSize → Large];
Manipulate[
 trajectory = ParametricPlot[
   Evaluate[{x[t], y[t]} /.
     NDSolve[{x'[t] == y[t], y'[t] == -2 Sin[x[t]] - y[t],
       x[0] == point[[1]], y[0] == point[[2]]},
      {x, y}, {t, 0, T}]], {t, 0, T}, PlotStyle → Red];
 Show[splot, trajectory],
 {{T, 10}, 0, 100},
 {{point, {1, 0}}, Locator},
 SaveDefinitions → True]
```

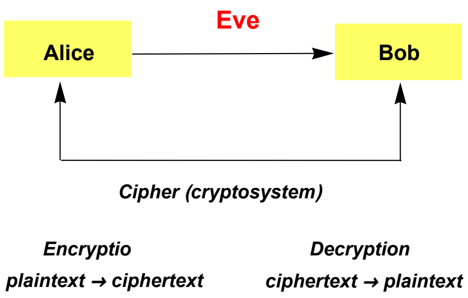

T
6
4
2
0
−2
−4
−6
−10
−5
0
5
10

Cryptography

Cryptology (Greek “kryptos” = hidden, secret, “logos”= study, science)
Cryptography (“kryptos”=secret, “graphein”=writing)

Definition:

Cryptography is the study of hiding information. Cryptography is the science of concealing the content of communication between parties, where the channel between them is controlled in some way by an unfriendly third party.



Eve

Alice

Bob

Cipher (cryptosystem)

Encryptio
plaintext → ciphertext

Decryption
ciphertext → plaintext

Cipher (Cryptosystem)

- a pair of algorithms that create the encryption from plaintext to ciphertext, and the reversing process (decryption from ciphertext to plaintext)
- the algorithms have a secret parameter called key (ideally known only to Alice and Bob, and which changes from one message exchange to the next).

Cryptanalysis

- study of methods for obtaining the meaning of encrypted information without access to the key normally required to do so (study of how to crack encryption algorithms).
- Eve (Eavesdropper, Enemy, Evil third party) intercepts the ciphertext. She may know the cipher (the algorithms for encryption, decryption, but does not know the key). Eve can have a “passive role” (attemps to recover the plaintext, or to deduce the key), or an “active role” (alter the message sent over the channel).

History of Cryptography

Classical Cryptopgraphy

- simple methods of encryption and decryption using only pen and paper an perhaps simple mechanical aids.
- non-standard hieroglyphs carved into monuments from the Old Kingdom of Egypt (1900 BC),
- clay tablets from Mesopotamia (1500 BC)
- Hebrew scholars - simple monoalphabetic substitution ciphers (500-600 BC) (replace first alphabet letter with last, second letter with next to last, ...)
- Ancient Greeks
  - transposition ciphers (500 BC) (Spartan Scytale - thin strip of parchment wrapped around a cylindrical rod called scytale; the diameter of the scytale was known only by the sender and the receiver)
  - Polybius square (205-123 BC) (pairs of numbers substitute for letters
- Romans - Caesar Cipher (50 BC, Gallic Wars) - Shift ciphers +k

Medieval Cryptography

- polyalphabetic substitution
- Alberti’s Cipher Disk
- Vigènere cipher (~16th century) correspondents agree on a keyword, which controls letter substitution depending on which letter of the key word is used.

WW1 (ADFGVX)

- correspondents knew an encoding table for letters & ciphers and a keyword; letters A, D, F, G, V, X were chosen deliberately because they sound very differently when transmitted via Morse code
- to encipher: the plaintext is coded using the letters A,D, F, G, V, X via the chart. The resulting pre-ciphertext is then enciphered using a keyword columnar transposition cipher.
- to decipher: reverse the transposition and decode using the chart.

●	A	D	F	G	V	X
A	F	L	1	A	O	2
D	J	D	W	3	G	U
F	C	I	Y	B	4	P
G	R	5	Q	8	V	E
V	6	K	7	Z	M	X
X	S	N	H	0	T	9

plaintext: WE WILL WIN
pre–ciphertext:
DFGXDFFDADADDFFDXD

A	R	M	S
1	2	3	4
D	F	G	X
D	F	F	D
A	D	A	D
D	F	F	D
X	D	□	□

keyword: ARMS
ciphertext:
DDADX GFAF FFDFD XDDD

WW2

- cable and radio transmissions were encrypted
- Enigma - cryptographic machine used by the Germans (Enigma accepted letters as keyboard input and implemented a polyalphabetic substitution cipher using an electro-mechanical rotor machine with rotating disks; it produced output by lighting electrical lamps beneath lettered windows)

Modern Cryptography

- Ciphering operations are too complex to be made by hand; computers are needed. Complex ciphering can be obtained by composition of a certain number of simpler ciphering functions.
- Claude Shanon (1949) established solid theoretical basis for cryptology and cryptanalysis (confusion/diffusion).
  Confusion - make the relation between the key and the ciphertext as complex as possible.
  Diffusion - data diffusion: changing one letter in the plaintext will change many letters in the ciphertext
  - key diffusion: changing even a tiny part of the key should change each bit (letter) in the ciphertext with a given probability.

- **Symmetric Key Cryptography**
  - the sender and the receiver share the same key; the key is communicated separately over a different channel.
  - 1970 Data Encryption Standard (DES) symmetric block cipher
  - 2001 Advanced Encryption Standard (AES)
    AES and Triple DES (with 128, 256-bit keys) are used in banking, ATM transactions, e-commerce, industry, mobile communications

- **Asymmetric (public) Key Cryptography**
  - new methods of distributing cryptographic keys (no secure channel is needed)
  - asymmetric key algorithms use a pair of mathematically related keys (one public, one private), each of which decrypts the encryption performed by the other; the private key cannot be deduced from the public key by any computational method other than trial and error. Cracking the encrypted message without the private key requires a major computational task.
  - based on number theory
  - Digital Signatures, Public Certificates, Authentication Protocols (SSL, SSH)
  - E-banking (secure bank transfers), E-commerce (cards), protection against collection of personal data by various companies, PGP email encryption, Operating Systems, Cryptographic Protection of classified documents, etc.

Example: The Caesar cipher

One of the oldest cryptosystems is due to Julius Caesar. The Caesar cipher is defined over the alphabet {A,B,C, ..., Z}, encoded numerically as integer numbers between 0 and 25, {A→0, B→1, ..., Z→25}. The key **k** can be any integer between 0 and 25. The cipher shifts each letter in the text cyclicly over k places. The encoding function is $E_k(x) = x + k \pmod{26}$ and the decoding function is $D_k(x) = x - k \pmod{26}$. The notation x mod 26 means the remainder of the division of x by 26.

Define the Caeser cipher over integer numbers {0, 1, ..., 25}

```
Clear[CaesarEncode, CaesarDecode]
```

```
CaesarEncode[x_ ?IntegerQ, k_] := Mod[x + k, 26]
CaesarDecode[x_ ?IntegerQ, k_] := Mod[x - k, 26]
```

```
SetAttributes[CaesarEncode, Listable];
SetAttributes[CaesarDecode, Listable];
```

```
Alphabet1 = Table[i, {i, 0, 25}]
CaesarEncode[Alphabet1, 3]
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25}
```

```
{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 1, 2}
```

ASCI code of alphabet letters

The Mathematica functions**ToCharacterCode[..]** and **FromCharacterCode[..]** convert symbols to their ASCI code and back (letter A has value 65, letter B has value 66, ... , letter a has value 97, letter b has value 98, etc.).

```
FromCharacterCode[65]
```

```
FromCharacterCode[{97}]
```

```
ToCharacterCode["A"]
```

```
ToCharacterCode["AaBbCc"]
```

```
ToUpperCase["AaBbCc"]
```

```
ToCharacterCode[ToUpperCase["AaBbCc"]]
```

```
Select[{4, 46, 2}, (0 <= # ≤ 25) &]
```

Define the Caesar cipher over strings

Encode and decode a given string using the simple encoding A=0, ... , Z=25 (stripping all other characters).

```
CaesarEncode[x_ ?StringQ, k_] := Module[{n, numerictext},
  numerictext = ToCharacterCode[ToUpperCase[x]] - 65;
  numerictext = Select[numerictext, (0 <= # ≤ 25) &];
  n = Mod[numerictext + k, 26];
  FromCharacterCode[n + 65]
 ]
```

```
CaesarDecode[x_ ?StringQ, k_] := Module[{n, numerictext},
  numerictext = ToCharacterCode[x] - 65;
  n = Mod[numerictext - k, 26];
  FromCharacterCode[n + 65]
 ]
```

```
? CaesarEncode
```

```
plain = "TYPE YOUR TEXT HERE";
CaesarEncode[plain, 2]
```

```
VARGAQWTVGZVJGTG
```

```
CaesarDecode["VARGAQWTVGZVJGTG", 2]
```

```
TYPEYOURTEXTHERE
```

Cryptanalysis of the Caesar cipher

An easy way to break the ciphersystem is to try out all possible keys. This method is called exhaustive key search, or brute-force approach.

Example : The text "TFEXIRKLCRKZFEJPFLALJKSIFBVPFLIWZIJKTZGYVI" has been encoded using a Caesar cipher with some unknown key k. Decrypt it.

Lab Exercises

Cryptanalysis of Caesar Ciphers:

An easy way to break the ciphersystem is to try out all possible keys. This method is called exhaustive key search, or brute-force approach.

Exercise: The text “TFEXIRKLCRKZFEJPFLALJKSIFBVPFLIWZIJKTZGYVI” has been encoded using a Caesar cipher with some unknown key k. Decrypt it.

Cryptanalysis of Affine Ciphers:

a) By exhaustive key search, or brute-force approach (try all possible keys):

The follwing text "KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX" has been encoded using an Affine Cipher. Decode the text.

b) By the method of a known ciphertext:

Exercise: Suppose that an affine cipher E(x) = ax+b (mod 26) enciphers “H” as “X” and “Q” as “Y”. Find the key. Use the key to decode the ciphertext “HXAEAEICJJOLCGPSQPVJCAPHOTHCHHCIG”, which has been encoded using the same cipher.

Hint: Solve a system of two linear equations mod 26.

c) By frequency analysis:

Exercise: Suppose that you have intercepted the following ciphertext
“QNRVKVITSTIJVLVFPUTHVONVPGNWZVTGWQTWWQVPFPWVZXPPVHDIVNGWQVHNGWITIFA
FPXZUSFHNDGWXGJWQVSVWWVICIVBDVGHXVPXGWQVHXUQVIWVYWTGOHNZUTIXGJWQVP
VRXWQWQVSVWWVICIVBDVGHXVPNCWQVVGJSXPQTSUQTAVWNGVKVIFBDXHLSFCXGOPWQ
VXZTJVPDGOVIWQVUVIZDWTWXNGUNCWQVZNPWCIVBDVGWSVWWVIPXGWQVUSTXGWVYW”
.
You know that it has been encoded using an affine cipher, but you don' t know the key. Use frequency analysis to break the cipher.

Cryptanalysis of Vigenere Ciphers:

Exercise: The following ciphertext was produced using a Vigenere cipher with a keyword of unknown length k.
“KIVQXTCQQVRPQOSGWEKVMTRYNHJVUVVKBMDPKVHKWBUWFVZFNOPQOSGWEKVMTRYNH
JGDPKOV”.

- Write a function that computes the index of coincidence of a ciphertext.

- Find an approximation of the length of the keyword, using Friedman's test. Now you know that approximatively k Caesar ciphers were used to encode the text.
- Try to break all of them. If frequency analysis does not work, then use the additional information that the cipher block “PQOSG” is an encoding for the plaintext “SOLVE”.
- Decode the ciphertext.

Cryptography with *Mathematica*

Modular Arithmetic

Definition: Let m be any positive integer. We say that two integers a and b are congruent modulo m if b-a is divisible by m. In other words, the division of a by m gives the same remainder as the division of b by m. We write a=b (mod m), or a ≡ b (mod m), or a-b ≡ 0 (mod m).

Equivalence Relation:

- **reflexive:** a ≡ a (mod m)
- **symmetric:** if a ≡ b (mod m) then b ≡ a (mod m)
- **transitive:** if a ≡ b (mod m) and b ≡ c (mod m) then a ≡ c (mod m)

Division Principle: Let m be any positive integer. Let b be any integer. Then there exist unique numbers q and r such that b=qm+r, and 0 ≤ r < m.
Smallest representative: b ≡ r (mod m) and we think of r as the simplest representative of the equivalence class of b modulo m.

The set of smallest representatives modulo m is denoted by $\mathbb{Z}_m$={0,1,2,3....,m-1}.

On $\mathbb{Z}_m$ we have two operations + (addition) and ⋆ (multiplication), with the following properties :

- If a, b and c are integers and a ≡ b (mod m), then a + c ≡ b + c (mod m)
- If a, b, c, d are integers and a ≡ b (mod m) and c ≡ d (mod m) then ac ≡ bd (mod m)
- Special case: If a and b are integers and n is a positive integer, then $a^n \equiv b^n$ (mod *m*).

Definition: A multiplicative inverse of an integer a modulo m is an integer b such that ab ≡ 1 (mod m). We write a^{-1}≡ b (mod m).

Examples :

- **2*3 ≡ 6 ≡ 1 (mod 5),** so **2^{-1}≡ 3 (mod 5)** and **3^{-1}≡ 2 (mod 5)**
- **2** is not invertible modulo **6.** Suppose, by contradiction, that there exists a number x such that 2x ≡ 1 (mod 6), then 2x-1 is divisible by 6, so in particular 2x-1 is divisible by 2, which is false, because 2x-1 is odd.

Theorem: If p⩾2 is a prime number, then any number a in {1,2,...p-1} has a multiplicative inverse in $\mathbb{Z}_p$

Theorem: Let m⩾2 be any positive number, and a be a number in {1,2,...m-1}. Then a has a multiplicative inverse in $\mathbb{Z}_m$ if and only if a and m are relatively prime.

- Suppose that a and b are relatively prime, and let us consider the set of remainders:

  a*0 (mod m), a*1 (mod m), a*2 (mod m), ... , a* (m-1) (mod m)

  Notice that remainders are all distinct. Otherwise, suppose that a*i ≡ a*j (mod m) for some i and j with 0<i<j<m. Then a(j-i) ≡ 0 (mod m), a(j-i) is divisible by m. But a and m are relatively prime, so j-i must be divisible by m. However this is not possible, because 0<j-i<m.

  In conclusion {a*0 (mod m), a*1 (mod m), a*2 (mod m), ... , a* (m-1) (mod m)} ={0,1,2,..., m-1}, so there exists a number k such that a*k ≡ 1 (mod m).

- Suppose now that a*b ≡ 1 (mod m) and let us show that a and m must be relatively prime.

Suppose by contradiction, that a and m are not relatively prime and let a= a_1*c and m=m_1*c, where 1<m_1<m.

a*b ≡ 1 (mod m) ⟹ a_1*c*b ≡ 1 (mod m) ⟹ a_1*c*b*m_1 ≡ m_1 (mod m)

⟹ (a_1*b)*(c*m_1) ≡ m_1 (mod m) ⟹ 0 ≡ m_1 (mod m), which is false because m_1 is not divisible by m.

Modular Arithmetic in *Mathematica*

PowerMod[a,b,m] - gives a^bmod m.
PowerMod[a,-1,m] - finds the inverse of a modulo m; returns an error message “a is not invertible modulo m” is the integer a is not invertible modulo m.
PowerMod[a,1/r,m] - finds a modular r root of a.

```
In[74]:=  PowerMod[3, -1, 7]

Out[74]=  5
```

```
In[75]:=  PowerMod[3, -1, 6]

          PowerMod::ninv: 3 is notinvertiblemodulo6. >>

Out[75]=  PowerMod[3, -1, 6]
```

Mathematica is able to solve modular equations using **Solve[{List of Equations}, {List of Variables}, Modulus->m]**

```
In[76]:=  Solve[{3 x == 1}, {x}, Modulus → 7 ]

Out[76]=  {{x → 5}}
```

```
In[77]:=  Solve[{2 x == 1}, {x}, Modulus → 26]

Out[77]=  {}
```

```
In[78]:=  Solve[{2 x == 2}, {x}, Modulus → 26]

Out[78]=  {{x → 1 + 13 C[1]}}
```

Affine ciphers

An affine cipher is an enchipherment scheme of the form $E_{a,b}(x) = ax + b$ **(mod 26)** and $D_{a,b}(x) = a^{-1}(x - b)$ **(mod 26)**, where a and b are integer numbers between 0 and 25, and a and 26 must be relatively prime.

These are the admissible values for the first key (key a):

```
Select[Table[i, {i, 25}], (GCD[26, #] == 1) &]
```

```
{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25}
```

An example of a good key : a = 3

```
Table[Mod[3 * i, 26], {i, 0, 25}]
```

```
{0, 3, 6, 9, 12, 15, 18, 21, 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23}
```

An example of a bad key: a = 2

```
Table[Mod[2 * i, 26], {i, 0, 25}]
```

```
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24}
```

Remark: Note that when a = 1, the Affine Cipher is just the Caesar Cipher discussed in the last lecture

Encoding and Decoding

```
Clear[AffineEncode, AffineDecode]
```

```
AffineEncode[x_ ?IntegerQ, a_, b_] := Mod[a * x + b, 26]
AffineEncode[x_ ?StringQ, a_, b_] :=
 Module[{n, numerictext},
  numerictext = ToCharacterCode[ToUpperCase[x]] - 65;
  numerictext = Select[numerictext, (0 <= # ≤ 25) &];
  n = Mod[a * numerictext + b, 26];
  FromCharacterCode[n + 65]
 ]
```

```
AffineDecode[x_ ?IntegerQ, a_, b_] := Mod[PowerMod[a, -1, 26] * (x - b), 26]
AffineDecode[x_ ?StringQ, a_, b_] :=
 Module[{n, numerictext},
  numerictext = ToCharacterCode[x] - 65;
  n = Mod[PowerMod[a, -1, 26] * (numerictext - b), 26];
  FromCharacterCode[n + 65]
 ]
```

Listable is an attribute that can be assigned to a function to indicate that the function should automatically be applied to lists that appear as its arguments. For example, Sin is listable, so Sin[{1, 2, 3, 4}] evaluates to {Sin[1], Sin[2], Sin[3], Sin[4]}

```
SetAttributes[AffineEncode, Listable];
SetAttributes[AffineDecode, Listable];
```

```
AffineEncode[3, 5, 1]
AffineDecode[16, 5, 1]
AffineEncode[{0, 1, 2, 3, 4, 5}, 5, 1]
AffineEncode["What to encode today?", 7, 11]
AffineDecode["JILOOFNYZFGNOFGLX", 7, 11]
```

```
16

3

{1, 6, 11, 16, 21, 0}

JILOOFNYZFGNOFGLX

WHATTOENCODETODAY
```

Simple substitution/Monoalphabetic substitutions

With the method of a simple substitution one chooses a fixed permutation p of the alphabet letters {a, b, …, z} and applies that permutation to all letters in the plaintext.

$E(x) = p(x)$
$D(x) = p^{-1}(x)$

```
Cipher =
  {"A" → "K", "B" → "E", "E" → "Z", "N" → "A", "Z" → "C", "C" → "B", "K" → "N"}
```

```
{A → K, B → E, E → Z, N → A, Z → C, C → B, K → N}
```

```
DecodeA = ("A" /. Cipher) → "A"
```

```
K → A
```

We can generate the English alphabet in several ways: from the ASCII codes, or by using the *Mathematica* function **Alphabet[..]**.

```
alphabet = Table[FromCharacterCode[i], {i, 65, 65 + 25}]
```

```
{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}
```

```
In[79]:=  alphabet = ToUpperCase @ Alphabet["English"]

Out[79]=  {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}
```

Now we can display the backwards correspondence between the permuted (encrypted) alphabet and the original one.

```
DecodeCipher = Table[(alphabet[[i]] /. Cipher) → alphabet[[i]], {i, 1, 26}]
```

```
{K → A, E → B, B → C, D → D, Z → E, F → F, G → G, H → H, I → I, J → J, N → K, L → L, M → M,
 A → N, O → O, P → P, Q → Q, R → R, S → S, T → T, U → U, V → V, W → W, X → X, Y → Y, C → Z}
```

Encoding and Decoding

StringReplace[“string”,s→sp] or StringReplace[“string”,{s1→sp1,s2→sp2,…}] replaces the string expressions s by sp whenever they appear as substrings of string.

```
Clear[EncodeP, DecodeP]
```

```
EncodeP[x_] := StringReplace[ToUpperCase[x], Cipher]
DecodeP[x_] := StringReplace[ToUpperCase[x], DecodeCipher]
```

```
EncodeP["ANOTHER CIPHER"]
```

```
DecodeP["KAOTHZR BIPHZR"]
```

Cryptanalysis of Affine Ciphers:

a) By exhaustive key search, or brute-force approach (try all possible keys):

The follwing text "KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX" has been encoded using an Affine Cipher. Decode the text.

b) By the method of a known ciphertext:

Exercise: Suppose that an affine cipher E(x) = ax+b (mod 26) enciphers “H” as “X” and “Q” as “Y”. Find the key. Use the key to decode the ciphertext “HXAEAEICJJOLCGPSQPVJCAPHOTHCHHCIG”, which has been encoded using the same cipher.

Hint: Solve a system of two linear equations mod 26.

c) By frequency analysis:

Exercise: Suppose that you have intercepted the following ciphertext
“QNRVKVITSTIJVLVFPUTHVONVPGNWZVTGWQTWWQVPFPWVZXPPVHDIVNGWQVHNGWITIFA
FPXZUSFHNDGWXGJWQVSVWWVICIVBDVGHXVPXGWQVHXUQVIWVYWTGOHNZUTIXGJWQVP
VRXWQWQVSVWWVICIVBDVGHXVPNCWQVVGJSXPQTSUQTAVWNGVKVIFBDXHLSFCXGOPWQ
VXZTJVPDGOVIWQVUVIZDWTWXNGUNCWQVZNPWCIVBDVGWSVWWVIPXGWQVUSTXGWVYW”
.
You know that it has been encoded using an affine cipher, but you don' t know the key. Use frequency analysis to break the cipher.

Cryptanalysis of Affine ciphers

Affine ciphers

An affine cipher is an enchipherment scheme of the form $\boldsymbol{E_{a,b}(x) = ax + b \ (\text{mod } 26)}$ and $\boldsymbol{D_{a,b}(x) = a^{-1}(x - b) \ (\text{mod } 26)}$, where a and b are integer numbers between 0 and 25, and a and 26 must be relatively prime.

These are the admissible values for the first key (key a):

```
In[38]:= GoodKeys = Select[Table[i, {i, 25}], (GCD[26, #] == 1) &]

Out[38]= {1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25}
```

Note that there are exactly 12 such numbers (the Euler characteristic of 26 is (2 - 1) (13 - 1) = 12).

Encoding and Decoding

```
Clear[AffineEncode, AffineDecode]
```

```
In[48]:= AffineEncode[x_ ?IntegerQ, a_, b_] := Mod[a * x + b, 26]
         AffineEncode[x_ ?StringQ, a_, b_] := Module[{n, numerictext},
           numerictext = ToCharacterCode[ToUpperCase[x]] - 65;
           numerictext = Select[numerictext, (0 <= # ≤ 25) &];
           n = Mod[a * numerictext + b, 26];
           FromCharacterCode[n + 65]
          ]
```

```
In[50]:= AffineDecode[x_ ?IntegerQ, a_, b_] := Mod[PowerMod[a, -1, 26] * (x - b), 26]
         AffineDecode[x_ ?StringQ, a_, b_] := Module[{n, numerictext},
           numerictext = ToCharacterCode[x] - 65;
           n = Mod[PowerMod[a, -1, 26] * (numerictext - b), 26];
           FromCharacterCode[n + 65]
          ]
```

```
In[43]:= AffineEncode["Looks like Affine chiphers are pretty easy", 17, 5]

Out[43]= KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX
```

```
In[44]:= AffineDecode["KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX", 17, 5]

Out[44]= LOOKSLIKEAFFINECHIPHERSAREPRETTYEASY
```

Associations

Association (<|....|>) - Along with lists, **associations** are fundamental constructs in the Wolfram Language. They associate keys with values, allowing highly efficient lookup and updating, even with millions of elements. Associations provide generalizations of symbolically indexed lists, associative arrays, dictionaries, hashmaps, structs, and a variety of other powerful data structures.
Association[key->val] associates keys with values
Association[{k->v, b->y}] key k is associated with value v, key b with value y, etc.
Convert a list of rules to an association using **Association[List]**. Convert an association back to list using **Normal[Association]**.

Numeric correspondents of alphabet letters:

```
In[16]:= NumAlphabet =
          Association @ Table[FromCharacterCode[i] → i - 65, {i, 65, 65 + 25}]

Out[16]= <|A → 0, B → 1, C → 2, D → 3, E → 4, F → 5, G → 6, H → 7,
          I → 8, J → 9, K → 10, L → 11, M → 12, N → 13, O → 14, P → 15, Q → 16,
          R → 17, S → 18, T → 19, U → 20, V → 21, W → 22, X → 23, Y → 24, Z → 25|>
```

If we want to obtain the frequency of letter A, we can now call TA["A"] in place of TA[[1]].

```
In[17]:= NumAlphabet["H"]

Out[17]= 7
```

```
In[18]:= NumAlphabet[[8]]

Out[18]= 7
```

Keys[..] and **Values[..]** can be used to obtain the keys or the values of an Association.

```
In[23]:= Keys[NumAlphabet]
         Values[NumAlphabet]

Out[23]= {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}

Out[24]= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25}
```

Table of frequency of letters from the English alphabet

```
In[8]:= T = {"A" → 0.0804, "B" → 0.0154, "C" → 0.0306, "D" → 0.0399,
           "E" → 0.1251, "F" → 0.0230, "G" → 0.0196, "H" → 0.0549,
           "I" → 0.0726, "J" → 0.0016, "K" → 0.0067, "L" → 0.0414,
           "M" → 0.0253, "N" → 0.0709, "O" → 0.0760, "P" → 0.0200, "Q" → 0.0011,
           "R" → 0.0612, "S" → 0.0654, "T" → 0.0925, "U" → 0.0271,
           "V" → 0.0099, "W" → 0.0192, "X" → 0.0019, "Y" → 0.0173, "Z" → 0.0009}
```

```
In[9]:=  TA = Association[T]

In[10]:= Sort[TA]

Out[10]= <|Z → 0.0009, Q → 0.0011, J → 0.0016, X → 0.0019, K → 0.0067, V → 0.0099,
          B → 0.0154, Y → 0.0173, W → 0.0192, G → 0.0196, P → 0.02, F → 0.023, M → 0.0253,
          U → 0.0271, C → 0.0306, D → 0.0399, L → 0.0414, H → 0.0549, R → 0.0612,
          S → 0.0654, N → 0.0709, I → 0.0726, O → 0.076, A → 0.0804, T → 0.0925, E → 0.1251|>
```

The letters of the English alphabet, ordered from most frequent to least frequent :

```
In[11]:= SortedLetters = Keys[Sort[TA, Greater]]

Out[11]= {E, T, A, O, I, N, S, R, H, L, D, C, U, M, F, P, G, W, Y, B, V, K, X, J, Q, Z}
```

Cryptanalysis of Monoalphabetic Ciphers

We call a monoalphabetic substitution cipher any cipher that does a permutation of the letters of the English alphabet. Unlike Caesar’s cipher or Affine ciphers, a general monoalphabetic substitution cipher does not have the drawback of a small key space. Indeed, the key space is 26! ≃4.03 10^26. However, a large key space does not guarantee that the system is secure! On the contrary, by simply counting the letter frequencies in the ciphertext and comparing these with the letter frequencies of the English alphabet, one very quickly finds the encodings of the most frequent letters in the plaintext. Indeed, the most frequent letter in the ciphertext will very likely be the encoding of one of the letters E, T, A, etc. After having found the encryptions of the most frequent letters in the plaintext, it is not difficult to fill in the rest. Of course, the longer the cipher text, the easier the cryptanalysis becomes.

Frequency analysis

The probability that letter “A” appears in an English text is approx. 0.0804. The probability that letter “E” appears in an English text is approx. 0.1251. The probability that letter “Z” appears in an English text is approx. 0.0009. Enciphering a text with a monoalphabetic cipher only permutes these frequencies around.

A function to count the number of occurences of letters in a text

We use the *Mathematica* function **StringCount[s, sub]** which counts the occurrences of the substring sub in the string s

```
In[12]:= LetterFrequencies[x_] := Module[{y, L, F},
           y = ToUpperCase[x];
           L = Table[FromCharacterCode[i], {i, 65, 65 + 25}];
           F = Table[L[[i]] -> StringCount[y, L[[i]]], {i, 1, 26}];
           Association[F]
          ]

In[13]:= LetterFrequencies["AAAABA gZZZZhADZ"]
```

```
In[14]:= Ordered = Sort[LetterFrequencies["AAAABA gZZZZhADZ"], Greater]

In[15]:= Keys[Ordered]
```

Cryptanalysis of Affine Ciphers:

a) By exhaustive key search, or brute-force approach (try all possible keys):

The follwing text "KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX" has been encoded using an Affine Cipher. Decode the text.

Solution :

What is the size of the key space of the Affine Cipher? The key of an Affine Cipher consists of two integers a and b, 0<= a,b < 25 and satisfying the additional restriction that a and 26 are relatively prime. There are 12 numbers relatively prime to 26. So the size of the key space is 12x26 = 312.

```
ciphertext2 = "KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX";
L = Select[Table[i, {i, 25}], (GCD[26, #] == 1) &];
For[i = 1, i ≤ Length[L], i++,
 For [j = 0, j ≤ 25, j++,
  Print[
   LanguageIdentify[AffineDecode[ciphertext2, L[[i]], j]],
   AffineDecode[ciphertext2, L[[i]], j]
  ]
 ]]
```

```
ciphertext2 = "KJJTZKLTVFMMLSVNULAUVIZFIVAIVQQXVFZX";
L = Select[Table[i, {i, 25}], (GCD[26, #] == 1) &];
For[i = 1, i ≤ Length[L], i++,
 For [j = 0, j ≤ 25, j++,
  If[LanguageIdentify[AffineDecode[ciphertext2, L[[i]], j]] == English (language) ,
   Print[AffineDecode[ciphertext2, L[[i]], j]]]
 ]]
```

b) By the method of a known ciphertext:

Exercise: Suppose that an affine cipher E(x) = ax+b (mod 26) enciphers “H” as “X” and “Q” as “Y”. Find the key. Use the key to decode the ciphertext “HXAEAEICJJOLCGPSQPVJCAPHOTHCHHCIG”, which has been encoded using the same cipher.

Hint: Solve a system of two linear equations modulus 26.

Solution :

```
Clear[a, b]
sol = Solve[
  {NumAlphabet["H"] * a + b == NumAlphabet["X"],
   NumAlphabet["Q"] * a + b == NumAlphabet["Y"]},
  {a, b},
  Modulus → 26
 ]
```

```
ciphertext1 = "HXAEAEICJJOLCGPSQPVJCAPHOTHCHHCIG"
AffineDecode[ciphertext1, a /. Flatten[sol], b /. Flatten[sol]]
```

c) By frequency analysis:

Exercise: Suppose that you have intercepted the following ciphertext. You know that it has been encoded using an affine cipher, but you don’ t know the key. Use frequency analysis to break the cipher.
“QNRVKVITSTIJVLVFPUTHVONVPGNWZVTGWQTWWQVPFPWVZXPPVHDIVNGWQVHNGWITIFA
FPXZUSFHNDGWXGJWQVSVWWVICIVBDVGHXVPXGWQVHXUQVIWVYWTGOHNZUTIXGJWQVP
VRXWQWQVSVWWVICIVBDVGHXVPNCWQVVGJSXPQTSUQTAVWNGVKVIFBDXHLSFCXGOPWQ
VXZTJVPDGOVIWQVUVIZDWTWXNGUNCWQVZNPWCIVBDVGWSVWWVIPXGWQVUSTXGWVYW”
.

Solution :

First we need to find out the most frequent letters in the ciphertext. Since a monoalphabetic substitution cipher was used, we expect that the most frequent letters in the ciphertext are encodings of the most frequent letters in the English alphabet. So we can use this information to break the cipher!

```
ciphertext3 =
  "QNRVKVITSTIJVLVFPUTHVONVPGNWZVTGWQTWWQVPFPWVZXPPVHDIVNGWQVHNGWITIFAFPXZUSFHN⋱
     DGWXGJWQVSVWWVICIVBDVGHXVPXGWQVHXUQVIWVYWTGOHNZUTIXGJWQVPVRXWQWQVSVWWVI⋱
     CIVBDVGHXVPNCWQVVGJSXPQTSUQTAVWNGVKVIFBDXHLSFCXGOPWQVXZTJVPDGOVIWQVUVIZ⋱
     DWTWXNGUNCWQVZNPWCIVBDVGWSVWWVIPXGWQVUSTXGWVYW";
S = Sort[LetterFrequencies[ciphertext3], Greater]
(* Letters in the ciphertext, from most frequent to least frequent *)
CipherLetters = Keys[S]
(* The letters of the English alphabet,
ordered from most frequent to least frequent *)
SortedLetters
```

```
Out[35]= <|V → 46, W → 33, G → 19, Q → 17, X → 16, I → 16, P → 15,
          T → 13, N → 12, S → 9, H → 9, U → 8, D → 8, Z → 7, F → 7, C → 6,
          J → 5, O → 4, B → 4, Y → 2, R → 2, L → 2, K → 2, A → 2, M → 0, E → 0|>

Out[36]= {V, W, G, Q, X, I, P, T, N, S, H, U, D, Z, F, C, J, O, B, Y, R, L, K, A, M, E}

Out[37]= {E, T, A, O, I, N, S, R, H, L, D, C, U, M, F, P, G, W, Y, B, V, K, X, J, Q, Z}
```

From exercise b) we know that it is enough to decipher two ciphertext letters to break the entire Affine cipher. Using frequency analysis, we infer that most likely letter V is an encoding of letter E, and letter

W is an encoding of letter T. We try to break the affine cipher using the method outlined in exercise b). If we get something that makes sense, it means that we have broken the cipher. Otherwise, we continue guessing, and assume that for instance, the ciphertext letter V is an encoding for E, and ciphertext letter W is an encoding for A.

```
Solve[{4 a + b == 21, 19 a + b == 22}, {a, b}, Modulus → 26]
```

```
{{a → 7, b → 19}}
```

```
In[70]:= Clear[a, b];
         sol2 = Solve[
            {NumAlphabet[SortedLetters[[1]]] * a + b == NumAlphabet[CipherLetters[[1]]],
             NumAlphabet[SortedLetters[[2]]] * a + b == NumAlphabet[CipherLetters[[2]]]},
            {a, b}, Modulus → 26];
         Print["We have broken the cipher ", Flatten[sol2],
          " and the decoded message is:"]
         AffineDecode[ciphertext3, a /. Flatten[sol2], b /. Flatten[sol2]]
```

```
         We have broken the cipher {a → 7, b → 19} and the decoded message is:

Out[73]= HOWEVERALARGEKEYSPACEDOESNOTMEANTHATTHESYSTEMISSECUREONTHECONTRARYBYSIMPLYCOUNTIN⋱
            GTHELETTERFREQUENCIESINTHECIPHERTEXTANDCOMPARINGTHESEWITHTHELETTERFREQUENCIESO⋱
            FTHEENGLISHALPHABETONEVERYQUICKLYFINDSTHEIMAGESUNDERTHEPERMUTATIONPOFTHEMOSTFR⋱
            EQUENTLETTERSINTHEPLAINTEXT
```

Polyalphabetic ciphers - Vigenere ciphers

The Vigenere cipher is a combination of several Caesar ciphers. Different letters in the plaintext are encrypted with different substitution alphabets.
How it works: Correspondents agree on a keyword.
To encrypt a plaintext, one writes the keyword repeatedly alongside the plaintext, converts both plaintext and keyword to their numerical equivalents (0-A, 1-B,...25-Z) and adds the corresponding letters modulo 26.
To decrypt a ciphertext, one writes the keyword repeatedly alongside the ciphertext, converts both ciphertext and keyword to their numerical equivalents (0-A, 1-B,...25-Z) and subtracts the corresponding letters modulo 26.

Keyword $k = k_0\, k_1\, k_2 \ldots k_{n-1}$
Plaintext $x = x_0\, x_1\, x_2 \ldots$
Ciphertext $y = y_0\, y_1\, y_2 \ldots$, where $y_i = (x_i + k_{(i \bmod n)}) \bmod 26$

Example: Ecrypt “GOAHEAD” using keyword “WIND”

plain	G	O	A	H	E	A	D
x	6	14	0	7	4	0	3
key	W	I	N	D	W	I	N
k	22	8	13	3	22	8	13
x + k (mod 26)	2	22	13	10	0	8	16
cipher	C	W	N	K	A	I	Q

Example : Decrypt "PWSRPULNB" using keyword "DWIN"

cipher	P	W	S	R	P	U	L	N	B
y	15	22	18	17	15	20	11	13	1
key	D	W	I	N	D	W	I	N	D
k	3	22	8	13	3	22	8	13	3
y - k (mod 26)	12	0	10	4	12	24	3	0	24
plain	M	A	K	E	M	Y	D	A	Y

Index of coincidence of a ciphertext

Experiment: Pick a pair of letters from a text. What is the probability of two letters being identical? Assume that the text has n letters, counting repetitions, distributed as follows: n_0 occurences of “A”, n_1 occurences of “B”, ... n_{25} occurences of “Z”, where $n_0 + n_1 + \ldots n_{25} = n$.

Counting Probabilities: Total number of ways in which the experiment can turn out: $C(n, 2) = \frac{n(n-1)}{2}$.

Total number of ways in which identical pairs can be obtained: C(n_0,2)+C(n_1,2)+...C(n_{25},2)=$\sum_{i=0}^{25} \frac{n_i(n_i-1)}{2}$

Definition: The number $I = \frac{2}{n(n-1)} \sum_{i=0}^{25} \frac{n_i(n_i-1)}{2} = \frac{1}{n(n-1)} \sum_{i=0}^{25} n_i(n_i - 1)$ is called the index of coincidence of the ciphertext. It represents the probability that two letters selected at random from the ciphertext are identical.

The index of coincidence of an alphabet in which each letter has the same frequency is **1/26 ≃ *0.0385***
The index of coincide of the English alphabet is about **0.065**

Monoalphabetic substitution (Caesar, Affine, Permutation): $I \simeq$ 0.065
Polyalphabetic substitution (Vigenere cipher): *I ≃ (closer to) 0.0385*

Vigenere ciphers from longer keywords have a more uniform distribution of letters, so the index of coincidence is closer to 0.0385. If the keyword is short, then the index of coincidence is closer to 0.065.

Connection between I and the keyword length

Formula (Friedman’s Test): Suppose that a ciphetext with n letters was obtained by a Vigenere encoding using a keyword with length k. Then the length of the keyword is approximatively

$$k \simeq \frac{0.0265\, n}{(0.065 - I) + n(I - 0.0385)}.$$

How it works :

Suppose that the ciphertext has n letters and the Vigenere keyword has k letters. Assume for simplicity that n is a multiple of k. We can group the letters of the ciphertext in a table with k columns and n/k rows, where the letters in each column are encoded with the same ciphertext.

Each column is a shift encipherment by an ammount corresponding to the key letter. If we choose a pair of letters at random, then they either:

1. come from the same column
2. come from different columns

Let' s analyze these two situations, and compute te probability that the chosen letters are identical.

Case 1.

We have **k** ways to choose the column and **C(n/k, 2)** ways to choose a pair of letters from the same column.
Since letters from the same column are encrypted using the same Caesar cipher, they have probabilty **0.065** to be identical. **The expected number of identical pairs of letters from the same column is:**
S = 0.065*k*C(n/k, 2)

Case 2.

We have **C(k,2)** ways to choose two distinct columns and **n/k** ways to choose a letter in the first chosen column, and likewise, also **n/k** ways to choose a letter in the second column.
Since letters from different columns are encrypted using different Caesar ciphers, they have probabilty **1/26 = 0.0385** to be identical. **The expected number of identical pairs of letters from different columns is:**

$$D = 0.0385 * C(k, 2) * \left(\frac{n}{k}\right)^2$$

Conclusion (Case 1 + Case 2)

The index of coincidence of the ciphertext will be

$$I \simeq \frac{S + D}{C(n, 2)} = \frac{0.065 * (n-k) + 0.0385 * n\,(k-1)}{k\,(n-1)}$$

If we want to guess a probable keyword length k from the index of coincidence of a ciphertext, we take the previous equation and solve for k. We obtain Friedman’s formula: $k \simeq \frac{0.0265\,n}{(0.065-I)+n(I-0.0385)}$

Exercise:

The following ciphertext was produced using a Vigenere cipher with a keyword of unknown length k. “**KIVQXTCQQVRPQOSGWEKVMTRYNHJVUVVKBMDPKVHKWBUWFVZFNOPQOSGWEKVMTRYNHJGDPKOV**”.

- Write a function that computes the index of coincidence of a ciphertext.
- Find an approximation of the length of the keyword, using Friedman's test. Now you know that approximatively k Caesar ciphers were used to encode the text.
- Try to break all of them. If frequency analysis does not work, then use the additional information that the cipher block “PQOSG” is an encoding for the plaintext “SOLVE”.
- Decode the ciphertext.

Prime numbers

A natural number $p \neq 1$ is prime if and only if it has no factors (divisors) other than 1 and itself.

Fundamental Theorem of Arithmetics.

Any positive integer n can be expressed as the product of powers of primes in a way that is unique up to a possible reordering of factors.

Thm (Euclid 300 BC) There are infinitely many prime numbers.

proof: Assume that there are exactly N primes, $p_1 = 2, p_2 = 3, \ldots p_N$
Then $m = (p_1 \cdot p_2 \cdot \ldots p_N) + 1$ is an integer bigger than $p_1, p_2, \ldots p_N$ which is not divisible by any of the primes $p_1, p_2, \ldots p_N$, contradiction.

Thm (Prime Number Theorem) let $\pi(x)$ be the number of primes less than or equal to x. Then

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1$$

Fermat's little Theorem let p be a prime number.

a) if a is relatively prime to p, then $a^{p-1} \equiv 1 \pmod{p}$.

b) $a^p \equiv a \pmod{p}$ for any integer a

proof a) let $a \in \{1, 2, \ldots p-1\}$. Consider the numbers

$$a \cdot 1 \bmod p \qquad a \cdot 2 \bmod p \qquad \ldots \qquad a \cdot (p-1) \bmod p$$

These are all distinct, and in the range $1, 2, \ldots p-1$. Otherwise, suppose by contradiction, that $a \cdot i \bmod p = a \cdot j \bmod p$ for some $1 \leq i < j \leq p-1$
Then $a(i-j) \equiv 0 \pmod{p} \Rightarrow a(i-j)$ is divisible by p. However this is impossible because p is prime, and $p \nmid a$ and $p \nmid i-j$.

In conclusion, multiplying by a has rearranged the numbers. $1, 2, \cdots p-1$.

$$(a\cdot 1)\cdot(a\cdot 2)\cdots(a\cdot(p-1)) \equiv 1\cdot 2\cdots p-1 \pmod p.$$

$$a^{p-1}\cdot(1\cdot 2\cdots(p-1)) \equiv (1\cdot 2\cdots(p-1)) \pmod p.$$

$\Rightarrow\ a^{p-1} \equiv 1 \pmod p$ because $1\cdot 2\cdots(p-1)$ is relatively prime to p. hence invertible mod p.

<u>Theorem (Euler)</u> let p and q be distinct primes.

a) if a is relatively prime to p and to q, then $a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$

b) If a is any integer, ~~then~~ and K is any positive integer, then

$$a^{K(p-1)(q-1)+1} \equiv a \pmod{pq}.$$

<u>proof</u>: a)

a is relatively prime to p and p is prime, so by Fermat's theorem we have

$$a^{p-1} \equiv 1 \pmod p.$$

a is relatively prime to q (which is prime) so by Fermat's Thm we know that

$$a^{q-1} \equiv 1 \pmod q.$$

Then we have:

$$\left.\begin{array}{l} (a^{p-1})^{q-1} \equiv 1 \pmod p \Rightarrow (a^{p-1})^{q-1} - 1 \equiv 0 \pmod p. \\ (a^{q-1})^{p-1} \equiv 1 \pmod q \Rightarrow (a^{q-1})^{p-1} - 1 \equiv 0 \pmod q \end{array}\right| \Rightarrow$$

$$\left.\begin{array}{l} p \,/\, (a^{p-1})^{q-1} - 1 \\ q \,/\, (a^{q-1})^{p-1} - 1 \end{array}\right| \Rightarrow pq \,/\, a^{(p-1)(q-1)} - 1$$

Since p and q are distinct primes $\quad \Leftrightarrow a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$

<u>Example</u>: Compute $3^{12} \pmod{26}$

$$3^{(13-1)(2-1)} \equiv 1 \pmod{26}.$$

Thm. Fermat let p be a prime number.

1. If a is relatively prime to p, then $a^{p-1} \equiv 1 \pmod p$.
2. If a is any integer, then $a^{P} \equiv a \pmod p$

Thm Euler let p and q be distinct prime numbers

1. If a is relatively prime to p, then $a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$
2. If a is any integer, and k is any positive integer, then

$$a^{k(p-1)(q-1)+1} \equiv a \pmod{pq}$$

Observation important for Cryptography

whenever two positive integers d and e satisfy $d \cdot e = 1 + k(p-1)(q-1)$

or equivalently $d \cdot e \equiv 1 \mod ((p-1)(q-1))$

then the functions

$E(x) = x^{e}$ MOD pq $\quad 1 \le x \le pq-1$ and

$D(y) = y^{d}$ MOD pq $\quad 1 \le y \le pq-1$ are inverses

$$E(D(y)) = E(y^{d}) = (y^{d})^{e} = y^{de} = y^{1+k(p-1)(q-1)} = y \ (\text{MOD } pq)$$

$$D(E(x)) = D(x^{e}) = (x^{e})^{d} = x^{de} = x^{1+k(p-1)(q-1)} = x \pmod{pq}.$$

$$x \xrightarrow{E} y = x^{e} \text{ MOD } pq \xrightarrow{D}$$

RSA public key cryptosystem

100-200 decimal digits

Alice – selects two distinct primes p and q.

– computes $m = pq$, $n = (p-1)(q-1)$.

– selects a number e which is relatively prime to n

– and uses the Extended Euclidean Algorithm to find the multiplicative inverse of e mod n, that is an integer d such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$

Alice broadcasts e and m.

m is called the modulus

e is the public key or encryption key

Alice keeps d and p and q a secret

d is the private key or decryption key

Bob – has a message to send to Alice

message = number x in range $0, m-1$.

- uses Alice's pubic key e and modulus m to encrypt the message:

$$y = x^e \text{ MOD } m$$

- sends y to Alice

Alice – receives y from Bob

- uses her private key d to decode the message:

$$x = y^d \text{ MOD } m$$

$$y^d \equiv (x^e \cancel{\text{MOD } m})^d \equiv x^{ed} \pmod m \equiv x \pmod m$$ because

$$ed \equiv 1 \bmod ((p-1)(q-1)) \Rightarrow ed = 1 + k(p-1)(q-1)$$

By Euler's theorem $x^{1+k(p-1)(q-1)} \equiv x \pmod m$

Bob – performs the same setup as Alice choose two distinct primes p' and q', etc ...

Cryptanalysis – essentially equivalent to factoring m

- the factors are large ~ 200 digits.
- the security of the RSA resides in the hardness/time consuming problem of factoring

Example :

+ select primes $p = 11$ and $q = 3$

- compute $m = pq = 33$ and $n = (p-1)(q-1) = (11-1)(3-1) = 20$
- choose e relatively prime with $n = 20$

  $e = 3$

- find d such that $e \cdot d \equiv 1 \pmod{20}$

  $d = 7$

  Euclidean Alg: find s, t such that $3s + 20t = 1$.

  $20 = 6 \cdot 3 + 2 \Rightarrow 0 = -2 - 3 \cdot 6 + 20$ | $\Rightarrow 1 = 7 \cdot 3 - 20$

  $3 = 2 + 1 \Rightarrow 1 = -2 + 3$

  $\Rightarrow 3^{-1} \equiv 7 \pmod{20}$.

  $\gcd(20, 3) = \gcd(3, 2) = \gcd(2, 1) = 1$

- public key $(e, m) = (3, 33)$
- private key $(d, m) = (7, 33)$.

Bob wants to encrypt the message $x = 14$, using Alice's public key

$y = x^e \pmod{m}$

$y = 14^3 \pmod{33} = 2744 \pmod{33} =$ ~~$83 \cdot 33 +$~~ $5 \pmod{33}$

$2744 = 83 \cdot 33 + 5$

Hence the ciphertext is ~~5~~ $y = \boxed{5}$

Alice - receives the ciphertext $y = 5$ from Bob.
- uses her private key to decrypt it.

$x = y^d \pmod{m}$

$x = 5^7 \pmod{33} = \boxed{14} \pmod{33}$

$5^7 = (5^3)^2 \cdot 5 = 125 \cdot 125 \cdot 5 = 15625 \cdot 5 = 78125$

$(-7)^2 \cdot 5 = 7 \cdot 2 = 14$

~~(mod 33)~~

$5^7 = 5^2 \cdot 5^2 \cdot 5^2 \cdot 5 = 25 \cdot 25 \cdot 25 \cdot 5$

$25 \equiv -8 \pmod{33}$

$5^7 \pmod{33} \equiv (-8)(-8)(-8) \cdot 5 \equiv$ ~~$4096(-35) \equiv 13 \cdot (-2) \equiv -26 \equiv 7$~~

$-64 \cdot 40 = 2 \cdot 7 = 14 \pmod{33}$

- receives the ciphertext $y = 17$

$D(17) = 17^7 \pmod{33} \equiv (-16)^7 \equiv -2^{28} \equiv -2^{20} \cdot 2^8 \equiv -2^8 \equiv -2^5 \cdot 2^3 \equiv (-32) \cdot 8 \equiv 8 \pmod{33}$

MAT 331: Project 3

Project description:

Cryptography has been used for secure communication for thousands of years. The modern world simply depends on cryptography; every time you make a mobile phone call, make an electronic transaction or an online payment with a credit card, or send a secure email, encryption is vital for protecting sensitive information. In this project we are interested in the theoretical foundations of the asymmetric/symmetric key cryptosystems, and on how they can be efficiently used together to make the information exchange more secure.

Tasks:

Introduction

Browse the web and describe some areas of our everyday life for which cryptography is essential. Describe the concept of End-to-End encryption and Digital Signatures.

Mathematical explanations

Describe the mathematics behind the RSA cryptosystem and prove Euler's Theorem. Also describe the mathematics behind Friedman's test (Friedman's test uses the index of coincidence of a ciphertext to calculate a probable keyword length for the Vigenere cipher).

Project overview

In this project we will use a Vigenere cipher to communicate secret information. Our alphabet will be the 26-letter English alphabet, numerically represented as integers between 0 and 25. As always, we first need to agree on a secret keyword for the Vigenere cipher (to be used for both encoding and decoding). Each letter from the plaintext/ciphertext message will be encoded/decoded separately, using the corresponding letter from the keyword. We will use RSA to securely exchange the keyword. My RSA scheme will the following:
public modulus m=**2480646863865156240312764602400149650804447106891126179090516987413492462101441** and public key e=**17**.

Part I - RSA

- Your first task will be to set up your own RSA scheme. For the purpose of this project, we will assume that it is secure enough if you choose two prime numbers with exactly 40 (decimal) digits. Submit your public RSA information on Blackboard. **Note: Send only the public information!**
  There are tables of prime numbers available online. You can also tell *Mathematica* to generate a pseudo-random prime number with k digits by using the command RandomPrime[10^k]. Double check that the number generated is indeed prime by using the Boolean function PrimeQ[...].
- After you complete the first step, you will receive message-signature pairs encoded with your RSA public information. Eve is also trying to get access to our private communications, so she will also try to send you fake Vigenere keywords encoded with your RSA public key. You will decode the message-signature pairs, and keep only the information for which the signature is valid, and discard the rest.

- The decoded message with valid digital signature will be an integer x between 1 and your RSA modulus. You will take this integer x and convert it to base 26. This will be our keyword for the Vigenere cipher.

  For example, if the decoded message is x=190, then its base 26 representation is 7 8, which corresponds to the English word “HI”.

Part 2 - Vigenere

- Build an interactive model in *Mathematica* for encoding/decoding English text messages that have been encrypted using a Vigenere cipher with keyword k. Your model should have a textbox s, in which the text to be encoded/decoded can be written, and a textbox k, for the keyword. By default, these boxes should be populated with the encrypted text below, and respectively with the key that you have obtained after the RSA decoding process. For these default settings, the model should output the decoded text.

  s=“KLIQEJXUULFYTECJXFRPKLOVCNLOPYBITKCSISWMLXOIRYIWDWKLIQEJXUULFYT”

- Add some extra functionality to your model from part 2. We know that the Vigenere cipher is vulnerable to statistical analysis, so you will now try to make your model work even if no key is provided. If the keyword box is empty, then you will compute the index of coincidence of the ciphertext and use Friedman’s test to guess the probable keyword length. Remember that you need to have a reasonably long chunk of encrypted text for statistical analysis to work; also recall that Friedman’s test is not very accurate so it is usually used in combination with other tests (if you get 7.2 by Friedman’s test, then the keyword could easily have length 6, 7 or 8). The keyword length tells you how many Caesar ciphers were used, so you can now proceed and break each of them by using ***letter frequency analysis*** (or in the worst case scenario, by brute-force approach).

  Your model should output the index of coincidence of the ciphertext and the decoded message (or ***a very short list*** with attempted decodings, including the correct one). Do not output thousands and thousands of attempted decodings! If you want to print other relevant information (like the probable keyword length), please feel free to do so.

  s=”**DLCZVMDIADMMXSDCILCMRSZCSRDYVKKXGYRYQEGXWRERYEXFYVGJIBKGAOWQYVDBESNYJORRMLYXKCCLYCFCORMPTPSQCMSLMIPXXFBSSQLMEXRRIAORREVGOWKYHCBRAYQKERGMERSSLDIARRGAYCCYQSREMSKZYROVQMSLXIADIBDLPYYERRCDAMBOQWEIOEJVHYDECFILWSPOZSVRCBEZVIDYVRRIQOXFBIYDWYVWMXIUSWQEIQRETOGMWISZXFKXUOVCXSRBIJOZYXXZOJMBIFYARYEBNEBSKGDEJCMEXEREVCDSYXIJOGRBSLSGBYGSWILDMLCYAREUKCRRERDLCCMEXIPMELXSRNILIPYDIPYRRRERDLCNSAEQCXXUKWQSKLOHZILGWLCB**”.

  s=”**AFUXMCKRJTIEXBCRJNTTCWFCLEMMDCTERARRMYCLPQMVKBJQXTSKVMGKFXTMGKPTIWUPQBDNEPAHLLMZLZKFXCCMKCKWPXHSXEABVQBERAVABGFXIRXORTEBVFKIRPBEEMYCLVUBKFMYCEVRMVPYICJLCGTGXJMYKFXVLZCGLYYEGFTSCMFLXMCKPONZADCWYZLWJRAVGFREXJSGUCKKFXGCKDSMRRBFLIFDMYCFFQMWPXHSXEREVRMVPLZLMYCICYBERXOR**”.

Part 3 - RSA Common Modulus Problem

- Does it work to break the RSA modulus m=**2480646863865156240312764602400149650804447106891126179090516987413492462101441** by a brute force approach? How long does it take?

- In this part, we will explore a vulnerability of the RSA system, known as the common modulus problem. Suppose that Alice and Kate use the same RSA modulus m as above, but with different encryption exponents $e_{\text{Alice}} = 17$ and $e_{\text{Kate}} = 169$. Bob wants to send the message x to both Alice and Kate. He first encrypts x using Kate’s public exponent, and sends Kate the ciphertex **249081508444719456881769789206569419957026519534551194619251016972858511647493.** Then he encrypts x using Alice’s public exponent and sends the ciphertext

5480386857778480218593900000000000000000 to Alice. If Eve intercepts both of the ciphertexts, explain how she can recover the plaintext *without knowing either decryption exponent*, then apply her method to find x.

Conclusions and references

- Give some brief instructions on how your model should be tested (how to use the controls, what functionalities you have implemented, etc.).
- Formulate your conclusions after testing your interactive model. Is the index of coincidence an accurate estimate for the length of the key? How secure is our communication protocol? How secure is RSA?
- Don't forget to include all relevant references.

RSA (Rivest, Shamir, Adleman)

- asymmetric key cryptosystem/ public key cryptosystem
- can be used for both encryption/decryption and authentification; in RSA we use the sender's private key to sign the message and the receipient's public key to encrypt the message.
- the security of the RSA cryptosystem resides in the hardness/time consuming problem of factoring a large number (the public modulus m) into its two prime factors (private primes p and q). Factoring integers is much harder than multiplying them!

```
p = 999 999 599;
q = 9 999 999 967;
m = p * q;
```

```
Timing[p * q]
Timing[FactorInteger[m]]
```

```
p = 2 425 967 623 052 370 772 757 633 156 976 982 469 681;
q = 5 570 373 270 183 181 665 098 052 481 109 678 989 411;
m = p * q;
```

```
Timing[p * q]
Timing[FactorInteger[m]]
```

★ The largest known prime number as of October 2015 is $2^{57\,885\,161} - 1$, a number with 17,425,170 decimal digits. RSA protocols usually use a 2048-bit modulus m (which corresponds to 617 decimal digits).

To generate a random prime in *Mathematica* we can use the command **RandomPrime[..]**

```
r1 = RandomPrime[{10^20, 10^21}]
PrimeQ[r1]
IntegerLength[r1]
BitLength[r1]
```

Mathematics behind RSA

- RSA is based on Euler's Theorem. Let p and q be two distinct prime numbers. Let m=pq.

0.1. If a is relatively prime to m, then $a^{(p-1)\,(q-1)} \equiv 1 \text{ (mod } m)$.

0.2. If a is any integer (not necessarily relatively prime to m), and k is any positive integer, then $a^{k(p-1)\,(q-1)+1} \equiv a (\text{mod } m)$.

RSA Setup

- Alice selects two distinct primes p and q and computes m=pq and n=(p-1)(q-1).

- She chooses a number e, relatively prime to n=(p-1)(q-1), and finds the multiplicative inverse of e modulo n, that is, she finds an integer d such that ed≡1(mod n).
- Alice makes (m,e) public. m is called the public modulus, and e the public key or the encryption key.
- Alice keeps p,q,n,d a secret. d is called the private key or the decryption key.

Encryption function: $E(x) = x^e (\text{mod } m)$, where 1≤x≤m-1.

Decryption function: $D(y) = y^d (\text{mod } m)$, where 1≤y≤m-1.

Example

Alice sets up an RSA scheme

- select primes p=11 and q=3.
- compute m=pq=33 and n=(p-1)(q-1)=(11-1)(3-1)=20
- choose a number e, relatively prime to n=20. Suppose we choose e=3.
- find an integer d such that ed≡1(mod 20). The integer d is the multiplicative inverse of e, modulo 20. In this case, d=$e^{-1}(\text{mod } 20) = 3^{-1} (\text{mod } 20) = 7 (\text{mod } 20)$.
- public key (m,e)=(33,3)

  e=3 is the public encryption key.

  m=33 is the public modulus.
- private key (p,q, n,d)=(11,3,20,7).

  d=7 is the private decryption key.

Bob wants to send the message x=14 to Alice.

- encrypt the message x=14 using Alice's public key.
- compute y=$x^e(\text{mod } m) = 14^3 (\text{mod } 33) = 2744 (\text{mod } 33) = 5 (\text{mod } 33)$.
- the ciphertext is **y=5**.
- Bob sends the ciphertext y=5 to Alice.

Alice receives the ciphertext y=5 (from Bob?)

- use the private key d=7 to decrypt it.
- compute x=$y^d(\text{mod } m) = 5^7 (\text{mod } 33)$.

  x=$5^7 (\text{mod } 33) = \left(5^3\right)^2 * 5 \ (\text{mod } 33) = 125^2 * 5 = (-7)^2 * 5 = 7 * 2 = 14 (\text{mod } 33)$
- the decrypted message is x=14. (This is exactly Bob's unencrypted message!)

RSA with *Mathematica*

```
EncryptionRSA[x_ ?IntegerQ, e_ ?IntegerQ, m_ ?IntegerQ] := PowerMod[x, e, m]
SetAttributes[EncryptionRSA, Listable]
```

```
DecryptionRSA[x_ ?IntegerQ, d_ ?IntegerQ, m_ ?IntegerQ] := PowerMod[x, d, m]
SetAttributes[DecryptionRSA, Listable]
```

To decrypt the cipher message y = 5 received from Bob, Alice computes :

```
DecryptionRSA[5, 7, 33]
```

```
14
```

To decrypt a list of cipher messages, say {5, 17, 10} received from Bob, Alice computes:

```
DecryptionRSA[{5, 17, 10}, 7, 33]
```

```
{14, 8, 10}
```

Short exercise : Produce a table with the encryptions of all integers between 0 and 32.

```
T = Table[i -> EncryptionRSA[i, 3, 33], {i, 0, 32}]
```

```
{0 → 0, 1 → 1, 2 → 8, 3 → 27, 4 → 31, 5 → 26, 6 → 18, 7 → 13,
 8 → 17, 9 → 3, 10 → 10, 11 → 11, 12 → 12, 13 → 19, 14 → 5, 15 → 9, 16 → 4,
 17 → 29, 18 → 24, 19 → 28, 20 → 14, 21 → 21, 22 → 22, 23 → 23, 24 → 30,
 25 → 16, 26 → 20, 27 → 15, 28 → 7, 29 → 2, 30 → 6, 31 → 25, 32 → 32}
```

Unconcealed Messages:

- All 33 values of x (from 0 to 32) map to unique ciphertext values y in the same range, in a sort of random manner.
- There are 9 values of x that map to themselves, 0,1,10,11,12, 21, 22, 23, 32! These are called unconcealed messages. The numbers 0,1,m-1 are no surprise, because they always map to themselves under the map x^e (mod m). To see how many unconcealed messages there are for a chosen pair (m,e), we must solve a modular equation $x^e \equiv x$ (mod m), or equivalently $x^d \equiv x$ (mod m). In this case one solves $x^3 \equiv x$ (mod 33), which is equivalent to $x(x-1)(x+1) \equiv 0$ (mod 33).

```
Solve[x^3 == x, x, Modulus → 33]
```

```
{{x → 0}, {x → 1}, {x → 10}, {x → 11}, {x → 12}, {x → 21}, {x → 22}, {x → 23}, {x → 32}}
```

Theorem: Let m = pq be the product of two disctinct prime numbers, and suppose that e is relatively prime to (p-1)(q-1). Then the number of elements x in $\mathbb{Z}_m$ that satisfy the relation $x^e = x$ (mod m) is given by **(GCD[e-1,p-1]+1)(GCD[e-1,q-1]+1).**

```
p = 3; q = 11; e = 3;
(GCD[e - 1, p - 1] + 1) (GCD[e - 1, q - 1] + 1)
```

Since p - 1, q - 1, e - 1 are all even, the number of unconcealed messages is at least 9. In practice, the encryption exponent e is small, whereas p and q are quite large, so the number of unconcealed messages will be very small compared to the total number of possible messages.

Symmetric vs Asymmetric key Criptography

Symmetric Key Cryptography

- same key used for encryption/decryption
- fast encryption/decryption
- key exchange - a big problem!
- used mainly for ecryption/decryption, not so much for digital signatures

Asymmetric Key Cryptography

- one key (public) used for encryption, another key (private) used for decryption
- slower encryption/decryption
- key exchange - no problem at all!
- can be used for ecryption/decryption, as well as for digital signatures

SSL (Secure Sockets Layer), TLS (Transport Layer Security)
Handshake Protocol

Several versions of the protocols are in widespread use in applications such as web browsing, email, Internet faxing, Instant messaging, and voice - over - IP (VoIP). Major web sites (including Google, YouTube, Facebook and many others) use TLS to secure all communications between their servers and web browsers.

- provides security and privacy over the Internet by using encryption and server/client authentification based on RSA.
- the SSL protocol negotiates encryption keys, as well as authenticates the server before data is exchanged by the higher level applications.
- HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol) can be layered on top of SSL

  ⟹ HTTPS (Hypertext Transfer Protocol Secure).

  HTTPS is especially important over unencrypted networks such as wifi where anyone on the same local network can eavesdrop (packet sniffing) and discover sensitive information.

  HTTPS provides a guarantee that one is communicating with precisely the web site/ web server that one intended to comunicate with, as well as ensuring that the messages between the user and the site cannot be forged by any third party.

SSL Handshake protocol has two parts

1. server authentification (based on RSA).
2. (optional) client authentification (RSA).

Part 1

- The handshake protocol begins when a client connects to a TLS or SSL-enabled server requesting a secure connection and presents a list of supported symmetric key ciphers (DES, Triple DES, AES, IDEA, MD5, RC2, RC4, etc.).
- the server, in response to a client’s request, sends its digital certificate (server name, trusted certificate authority, public RSA key) and its symmetric key cipher method preference.
- the client generates a master key k, which he/she/it encrypts with the server’s public key using RSA, and transmits the encrypted master key to the server.
- the server decrypts (recovers) the master key k, using his RSA private decryption key.
- the server authenticates itself to the client by returning a message signed with his private RSA key and encrypted with the symmetric cipher with key k.
- subsequent data is encrypted with the symmetric key cipher and with keys derived from the master key k.

Part 2 (optional)

- the server sends a challenge to the client
- the client authenticates itself to the server by returning the client’s digital signature on the challenge, as well as its public-key certificate.

Electronic commerce

IKP (Internet Keyed Payment Protocol)

- secure payments involving three or more parties; a buyer and a seller interact with a third party “acquirer” such as a credit card system or a bank, to authorize transactions.

SET (Secure Electronic Transaction Protocol)

- secure, cost effective bankcard transactions over open networks, implemented by VISA, MasterCard, etc.
- SET includes protocols for purchasing goods and services electronically, requesting authorization of payment and requesting “credentials” (digital certificates) binding public keys to identities.
- SET supports DES for bulk data encryption and RSA for signatures and public-key encryption of the DES encryption keys and bankcard numbers.

RSA Encoding/Decoding

Recall the setup:

- Alice selects two distinct primes p_A and q_A and computes $m_A = p_A * q_A$ and $n_A = (p_A - 1)(q_A - 1)$.
- She chooses a number e_A, relatively prime to $n_A = (p_A - 1)(q_A - 1)$.
- She finds the multiplicative inverse of e_A modulo n_A, that is, she finds an integer d_A such that $e_A\, d_A \equiv 1 \pmod{n_A}$.
- Alice makes m_A and e_A public. m_A is called the RSA modulus, and e_A the public key or the encryption key.
- Alice keeps the other information a secret (p_A, q_A, n_A, d_A). The integer d_A is called the private key or the decryption key.

Encryption function: $E(x) = x^{e_A} (\mathrm{mod}\;\; m_A)$, where $1 \leqslant x \leqslant m_A - 1$.

Decryption function: $D(y) = y^{d_A} (\mathrm{mod}\;\; m_A)$, where $1 \leqslant y \leqslant m_A - 1$.

```
EncryptionRSA[x_ ?IntegerQ, e_ ?IntegerQ, m_ ?IntegerQ] := PowerMod[x, e, m]
SetAttributes[EncryptionRSA, Listable]
```

```
DecryptionRSA[x_ ?IntegerQ, d_ ?IntegerQ, m_ ?IntegerQ] := PowerMod[x, d, m]
SetAttributes[DecryptionRSA, Listable]
```

Digital signatures with RSA

We want a digital signature that can help confirm the identity of the sender of a message. The sender should not be able to deny that it/he/she is the author the message (non - repudiation).

Two party-communication via RSA

- Alice and Bob
- Alice has her RSA scheme: public modulus $m_A = p_A * q_A$, public encryption key e_A, private decryption key d_A.
- Bob sets up his own RSA scheme: public modulus $m_B = p_B * q_B$, public encryption key e_B, private decryption key d_B.

Evil Third Party

- Eve wants to impersonate Bob. She writes a message x="Alice, I don't like you, Sincerely Bob", Encrypts x using Alice's public key y= $x^e (\mathrm{mod}\, m)$, and send y to Alice. Alice decrypts the message y using her private key d, reads the message, but she has no way of verifying if the sender was really Bob.

Alice and Bob

- Bob would like to send Alice the following message x="Alice, I like you, Sincerely Bob". He would like to send Alice the message x, and also affix a digital signature to confirm his identity. This will be called a message-signature pair.

Bob

- Bob signs the message x with his own private RSA key. The signature is $\sigma = x^{d_B} \bmod m_B$.
- Bob encrypts the message-signature pair (x, σ) to Alice using Alice's public encryption key e_A. The result is the encrypted message-signature pair (y, β) = ($x^{e_A} \bmod m_A$, $\sigma^{e_A} \bmod m_A$).
- Bob send the encrypted message signature pair (y,β) to Alice.

Alice

- Alice receives the encrypted message signature pair (y,β) to Bob. At this stage, she doesn't know if the sender of the message is really Bob.
- She decrypts the message using her private decryption exponent d_A.

  She computes (x, σ)=($y^{d_A} \bmod m_A$, $\beta^{d_A} \bmod m_A$).
- Alice looks up Bob's public key e_B and public modulus m_B. She checks Bob's digital signature by computing z=$\sigma^{e_B} \bmod m_B$. If z=x, then the message is authentic. If z≠x, then the sender of the message is not Bob.

Example

Alice and Bob want to communicate via RSA and digitally sign their messages so that they cannot be forged by an evil third party.

RSA Setup

- Bob: selects the primes $p_B = 5$, $q_B = 13$, and computes $m_B = p_B * q_B = 65$, $n_B = (p_B - 1)(q_B - 1) = 48$. He chooses $e_B = 5$, relatively prime to 48, and computes $d_B = 29$ (the multiplicative inverse of 5 modulo 48). He makes $m_B = 65$ (RSA modulus) and $e_B = 5$ (RSA encryption key) public. He keeps the other information private.
- Alice: $p_A = 3$, $q_A = 11$, $m_A = p_A * q_A = 33$, $n_A = (p_A - 1)(q_A - 1) = 20$, $e_A = 3$, $d_A = 7$.

  She makes $m_A = 33$ and $e_A = 3$ public and keeps the rest a secret.

Message-Signature Pair

- Bob wants to send Alice the message x = 9.

  First he digitally signs the message using his private RSA key.

  $\boldsymbol{\sigma = x^{d_B} \bmod m_B = 9^{29} \bmod 65 = 3^{58} \bmod 65 = 3^{10} \bmod 65 = 29 \bmod 65.}$

  The digital signature of the message is σ = 29.

```
9^29
3^10
PowerMod[9, 29, 65]
```

The message signature pair is **$(x,\sigma) = (9, 29)$.**

Encrypted Message-Signature Pair

- Bob encrypts the message signature pair (x,σ) using Alice's public modulus m_A and encryption key e_A.

  $y = x^{e_A} \bmod m_A = 9^3 \bmod 33 = 3$

  $\beta = \sigma^{e_A} \bmod m_A = 29^3 \bmod 33 = 2$

```
PowerMod[{9,29}, 3, 33]
```

The encrypted message signature pair is **$(y, \beta) = (3, 2)$.**

Bob sends this encrypted and digitally signed message to Alice.

Decoding the message

- Alice receives the pair **$(y, \beta) = (3, 2)$** (from Bob?). She decrypts the message using her private decryption key $d_A = 7$.

  $x = y^{d_A} \bmod m_A = 3^7 \bmod 33 = 9$

  $\sigma = \beta^{d_A} \bmod m_A = 2^7 \bmod 33 = 29$

```
PowerMod[{3,2}, 7, 33]
```

The decrypted message-signature pair is **$(x,\sigma) = (9,29)$**.

Is the signature valid? Is the message authentic?

- Alice checks to see if the signature σ is valid. She looks up Bob's public RSA modulus $m_B = 65$ and encryption key $e_B = 5$.

  She computes $z = \sigma^{e_B} \bmod m_B = 29^5 \bmod 65 = 9$

  She compares z with x, $z = x = 9$, so the digital signature is valid and the message is authentic.

Exercise:

1. Alice receives several encrypted message-signature pairs. What are the decoded messages? Which messages are authentic (that is, sent from Bob)?

```
L = {{3, 8}, {3, 28}, {6, 27}, {6, 28}, {6, 10}, {6, 6}}
```

A more realistic example:

A user logs on to his bank' s homepage www.bank.com to do online banking. When the user opens www.bank.com homepage, they receive a public key along with all the data that his web browser displays. The public key could be used to encrypt data from the client to the server but the method would not be safe or fast. In fact, any evil third party could easily impersonate the bank, issue a fake public key (for which it would have a matching private key) and the naive client could get tricked into encrypting sensitve information with the fake public key and send it over to the evil third party.

The safe procedure is to use asymmetric key cryptography in a handshake protocol that authenticates the bank to the client and also helps exchange a key for a symmetric cryptosystem that will be used in further communications. Let's revisit the handshake protocol:

SSL (Secure Sockets Layer), TLS (Transport Layer Security) Handshake Protocol

Several versions of the protocols are in widespread use in applications such as web browsing, email, Internet faxing, Instant messaging, and voice - over - IP (VoIP). Major web sites (including Google, YouTube, Facebook and many others) use TLS to secure all communications between their servers and web browsers. SSL is what keeps your passwords, communications and credit card details safe on the way between your computer and the servers you want to send this data to.

- HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol) can be layered on top of SSL

  $\Longrightarrow$ HTTPS (Hypertext Transfer Protocol Secure).

  HTTPS is especially important over unencrypted networks such as wifi where anyone on the same local network can eavesdrop (packet sniffing) and discover sensitive information.

  HTTPS provides a guarantee that one is communicating with precisely the web site/ web server that one intended to comunicate with, as well as ensuring that the messages between the user and the site cannot be forged by any third party.

The main purposes of the SSL are the following:

- provides security and privacy over the Internet by using encryption and server/client authentification based on RSA.
- the SSL protocol negotiates encryption keys for a symmetric key algorithm, as well as authenticates the server before data is exchanged by the higher level applications.

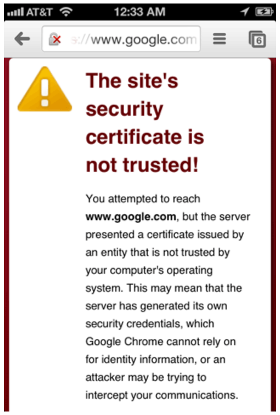
SSL Handshake protocol has two parts:

1. server authentification (based on RSA).
2. (optional) client authentification (RSA).

How it works:

- (Client Hello) The handshake protocol begins when a client connects to a TLS or SSL-enabled server requesting a secure connection and presents a list of supported symmetric key ciphers (DES, Triple DES, AES, IDEA, MD5, RC2, RC4, etc.).
- (Server Hello) The server, in response to a client's request, sends its digital certificate (server name, trusted certificate authority, public RSA key) and its symmetric key cipher method preference. By sending its digital certificate, it proves its identity to the client.
- The client verifies the authenticity of the server's digital certificate. The client checks if the certificate is verified and trusted by one of several Certificate Authorities (CAs) that it implicitly trusts. Your

browser has a pre-installed list of trusted SSL certificates from Certificate Authorities (CAs) that you can view, add and remove from. In Mozilla Firefox for example, you can check this list of trusted CAs from Preferences->Advanced->Certificates->View Certificates->Authorities. These certificates are controlled by a centralised small group of extremely secure, reliable and trustworthy organisations like Symantec (VeriSign), Comodo, GoDaddy, GlobalSign, DigiCert, Entrust, Verizon, Deutsche Telekom and so on. If a server presents a certificate from that list then you know you can trust them. The role of the Certificate Authority is to issue digital certificates that contain a public key and the identity of the owner of the public key and to certify that a certain public key does indeed belong to whoever is identified in the certificate. The digital certificate is digitally signed with the private key of a Certificate Authority. If the user trusts the CA and can verify the CA’s digital signature, then the user will trust the digital certificates signed by the CA. If the server cannot be authenticated, the user is warned of the problem and informed that the digital certificate is not trusted and an encrypted and authenticated connection cannot be established.



The site's security certificate is not trusted!

You attempted to reach **www.google.com**, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Google Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

- the client generates a master key k, which he/she/it encrypts with the server’s public key using RSA, and transmits the encrypted master key to the server. *(If the server has requested client authentication (an optional step in the handshake), the client also digitally signs another piece of data that is unique to this handshake and known by both the client and server. In this case, the client sends both the signed data and the client’s own certificate to the server along with the encrypted master key.)*
- the server decrypts (recovers) the master key k, using his RSA private decryption key. This step is crucial to proving the authenticity of the server. *(If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session ends.)*
- the server authenticates itself to the client by returning a message like “Finished” signed with his private RSA key and encrypted with the symmetric cipher with key k. The message that ends the Hadshake usually also contains a hash of all data previously exchanged between the parties.
- subsequent data is encrypted with the symmetric key cipher and with keys derived from the master key k.

Exercise

1. John decides to use RSA encryption. He chooses p=23, q=31 and e=19. He makes public m=713 and e=19. He also publicizes the way in which English text is to be numerically encoded. Two letter-blocks are interpreted as base 26 representations. For example, the numeric equivalent of H is 7, of I is 8, so the numeric equivalent of the 2-letter block HI will be 78 in base 26, that is, HI=7*26+8=190. So if John decodes an RSA message and discovers that the number is 190, he know that this means HI. Say HI to John, using RSA.