Scientific Computing

Contact Details

Course Convenor: Ilian Iliev

Office: Pev III 4C5, tel. (01273-87) 3737

Check Study Direct regularly (lectures, assignments, links, codes and other materials will be provided).

Communication will be done either in-class or through e-mail – check often your university e-mail!

Office Hours: open-door, please email me beforehand to make sure I will be there: I.T.Iliev@sussex.ac.uk

Generally, feel free to contact me with any (course-related) questions you might have.

ATs: Jonathan Davies, Kerim Suruliz, Keri Dixon, Hannah Ross, Mateja Gocenca, Oliver Winston, Scott Clay

Course Aims

- learn the fundamental numerical computing algorithms ("building blocks") for your future programming needs, regardless of specialization.
- develop intuition for what computers can (and, importantly, cannot) do for you and how to best use the available resources.
- learn the basics of how to use Python for science.
- develop general problem-solving strategies.

Course Topics

- Brief introduction to Python
- Linear algebra, systems of linear equations
- Interpolation, data fitting
- Solving algebraic equations numerically
- Numerical differentiation and integration
- Numerical methods for solving ordinary differential equations
- Fourier transforms and their applications in physics and other areas

Course Pre-requisites: what are you expected to know

- Basic algebra.
- Linear algebra (vectors, matrices, etc.) - an idea.
- A bit of calculus (differentiation, integration).
- Ordinary differential equations – just the basics.

Note: no much previous experience with Python is expected, but do go carefully through the provided tutorials!

Please let me know promptly about any existing gaps, if I don't know about a problem I can't help you with it.

Work and Assessment

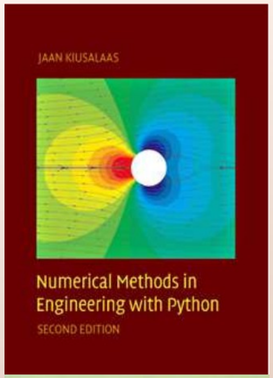
Teaching / learning activities:

- 12 weekly lectures (1h).
- 12 weekly exercise practical classes (1h) + 12x4 weekly help sessions online (Tue 17-18, Pev 1 1B4; Mon 11-12, Tue 10-11 and Fri 11-12 online on Study Direct – use chat functionality).
- hours of independent work on your own per week.

Assessment:

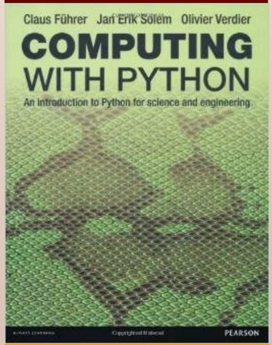
- Assessed exercises due in weeks 6, 9 and 12 (13.3% each; only 1 problem per week assessed). Technical help will be provided. Discussions are encouraged, but any work you turn in should be your own! Assignments will be turned in through Study Direct, as a single zip/tar file per person. Deadline for assessed work: about week after the last problem is assigned. Up to 24h late work will be accepted (with penalties).
- Exam (60%). Note that for all courses now pass means 40% or more on both the exam and overall (coursework+exam).

Books on scientific computing with Python



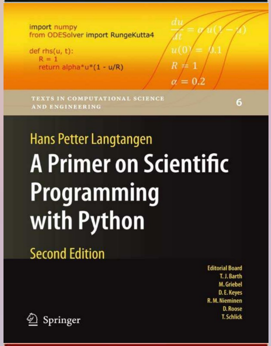
Numerical Methods in Engineering with Python
by Jaan Kiusalaas

This is the main text for our course. Excellent pedagogical text, with a short introduction to Python (newer edition exists, but that uses Python 3).


Claus Führer Jan Erik Solem Olivier Verdier
COMPUTING WITH PYTHON
An introduction to Python for science and engineering
PEARSON

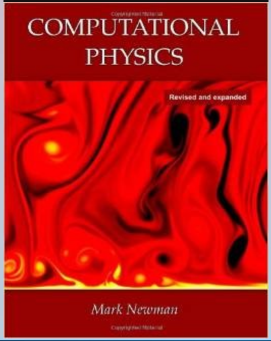
Computing with Python
by Claus Führer, Jan Erik Solem and Olivier Verdier

Excellent introduction into Python for science
Covers some, but not all topics we will do.


TEXTS IN COMPUTATIONAL SCIENCE AND ENGINEERING
6
Hans Petter Langtangen
A Primer on Scientific Programming with Python
Second Edition
Springer

A Primer on Scientific Programming with Python
2nd ed., by Hans Petter Langtangen

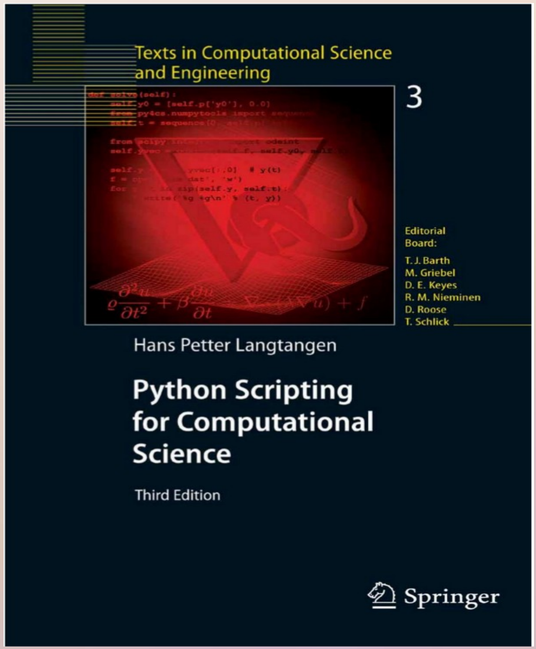
Well-written, gentle introduction into Python with examples from scientific computing. Does not cover all topics we will do, however.

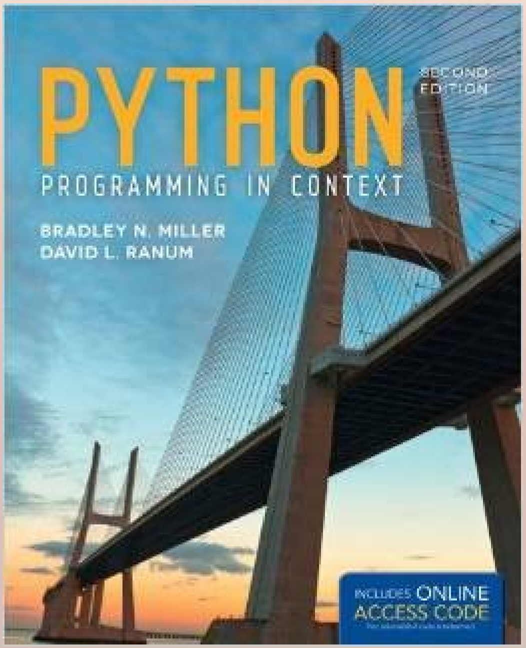


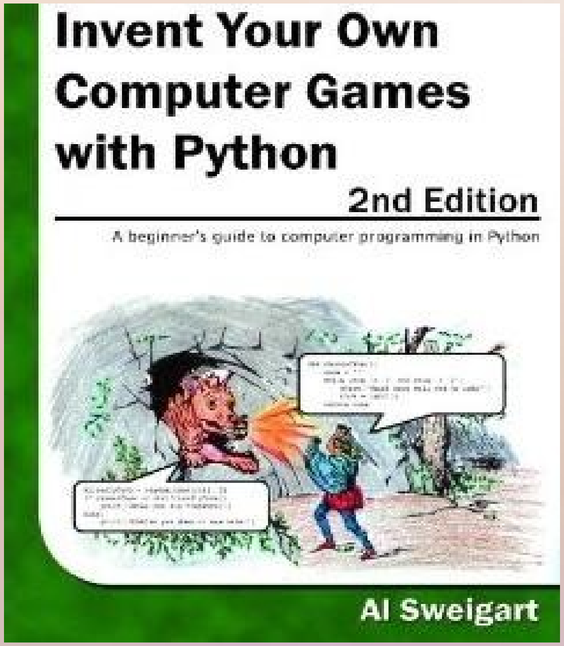
Computational Physics
by Mark Newman

Both basic Python and variety of Physics applications considered. Excellent textbook, but does not cover all areas.

Books on Python






Invent Your Own Computer Games with Python
2nd Edition
A beginner's guide to computer programming in Python
Al Sweigart

http://inventwithpython.com/

Many are available. Above are a couple of very nice introductions assuming no previous knowledge, as well as one (left) more advanced book. They teach Python using examples from different areas (from cryptography to statistics to games). Last 2 books are based on Python 3.

Further info: web-based resources

- Python home page (http://www.python.org)
- Pylab/Matplotlib (http://matplotlib.sourceforge.net/): plotting as in Matlab
- Numpy - NUMerical PYthon and SciPy – Scientific Python (http://numpy.scipy.org/): vectors, matrices, other math tools, scientific algorithms.
- Visual Python - 3D visualisation (http://www.vpython.org/): available, but will not be covered in this course.
- SymPy - Symbolic calculation (http://sympy.org/): available, but will not be covered in this course.
- and many, many web tutorials, etc. Some will be made available on Study Direct.

Other books on numerical methods

Any other good book on Scientific Computing could be used (few are based on Python, however, and they do not necessarily cover all topics, or at the same level as our course), e.g.:

P.L. DeVries, J.E. Hasbun "A First Course in Computational Physics", Jones & Bartlell, 2011

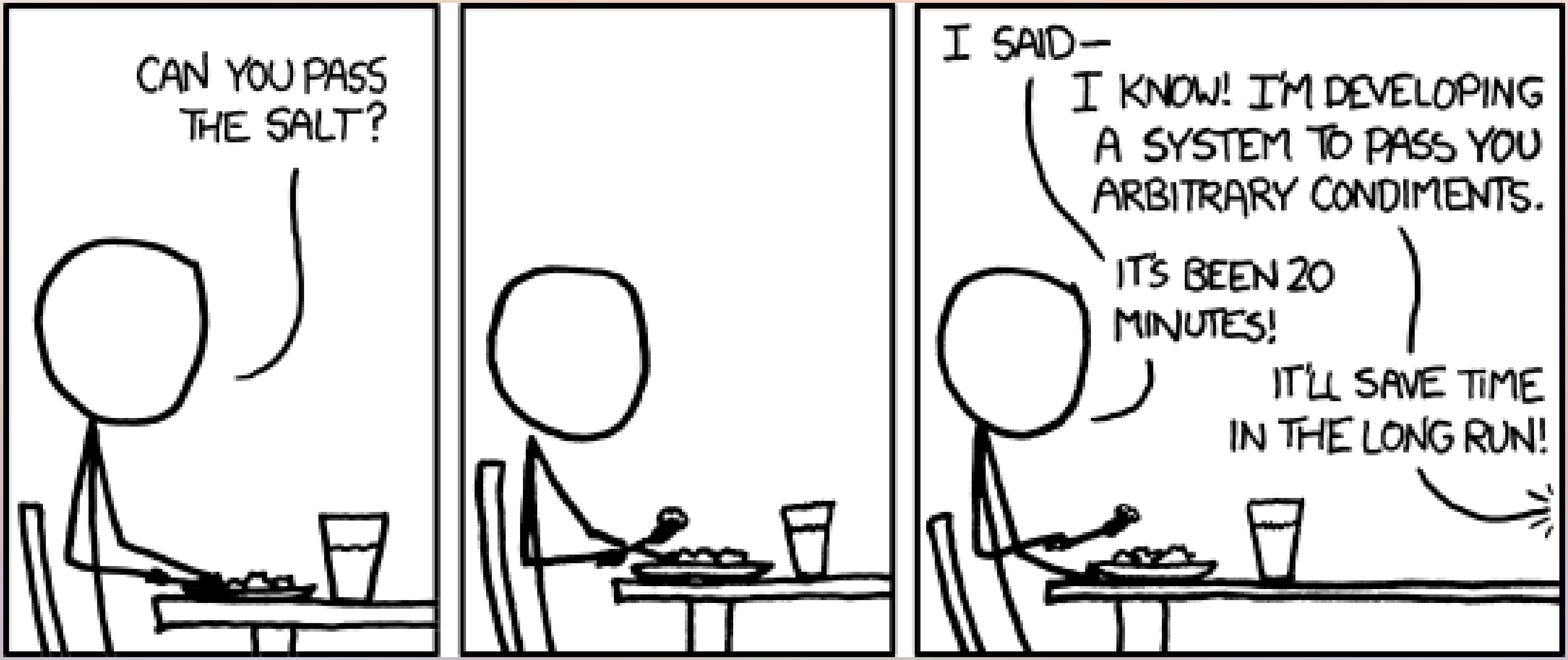
C. Moler, "Numerical Computing with MATLAB": http://www.mathworks.com/moler/chapters.html

R. Fitzpatrick, "Introduction to Computational Physics": http://farside.ph.utexas.edu/teaching/329/329.pdf

Good programming style

- First think about the problem, algorithm and then the program.
- Use comments liberally for your own sake, as well as for others reading/using your code. ALWAYS start a program with a short description of what it does.
- Define all variables (with sensible names, preferably), describing meaning briefly
- Use indenting for commands within 'if', 'while' and 'for' structures – this is especially critical for Python!
- Write explicitly all operations (e.g. ab DOES NOT mean a*b)
- Try the algorithm out by hand (e.g. the first 2-3 iterations of a loop) and see if you get the answers you expect
- Printing out intermediate values of variables is a powerful debugging tool

Good programmer's approach


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

Introduction to Python for Science

(just a few notes, most intro will be done in the first 2 workshops)

What is Python?

- High level programming language (i.e. closer to the human way of thinking)
- Interpreted language, i.e. statements executed line-by-line, no need for compilation into machine language first (though this could be done for faster execution)
- Similar to MATLAB, but somewhat different syntax (cheat sheet available on Study Direct)
- More flexible, can do much more than just numerical computations (searches, databases, GUIs,...) – 'transferrable skills'.

Availability

- Python is free
- Python is platform independent (works on Windows, Linux/Unix, Mac OS, even smartphones/tablets)

Python versions

- There are currently two versions of Python:
  - Python 2.x and
  - Python 3.x
- We are using version 2.6. Any other version of Python 2 should be (more or less) fine, too (latest is 2.7). Python 2 is compatible with the numerical extension modules scipy, numpy, pylab.

  Note: Python 2.x and 3.x are incompatible although the changes only affect a few commands. Make sure you use version 2.x and literature that covers 2.x.

Getting your own Python

You can get Python on your own laptop/desktop:

- Linux and Mac machines should already have Python installed

  (default Mac versions tend to be old, but can be updated, see https://www.python.org/downloads/mac-osx/).

- The installation instructions for Windows can be found at:
  - https://code.google.com/p/pythonxy/
- IPython shell: http://pypi.python.org/pypi/ipython
- Numpy, Matplotlib and SciPy:

  http://sourceforge.net/projects/matplotlib/?source=recommended
  http://sourceforge.net/projects/numpy/?source=recommended
  http://sourceforge.net/projects/scipy/files/latest/download?source=recommended

Starting with Python

- Start IPython shell (with scilab for scientific modules and plotting):

  ipython -pylab

  (in newer Python versions: ipython --pylab)

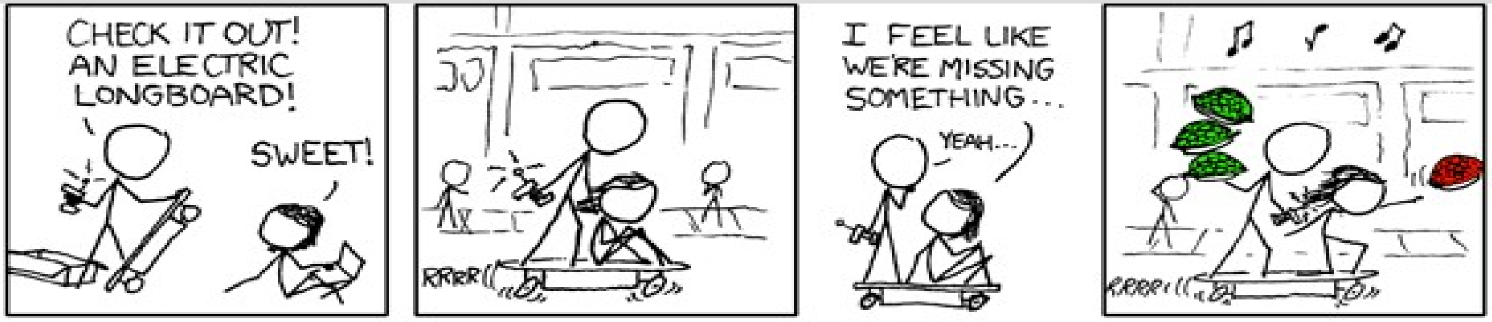
- IPython is an user friendly interface for testing and debugging of code.

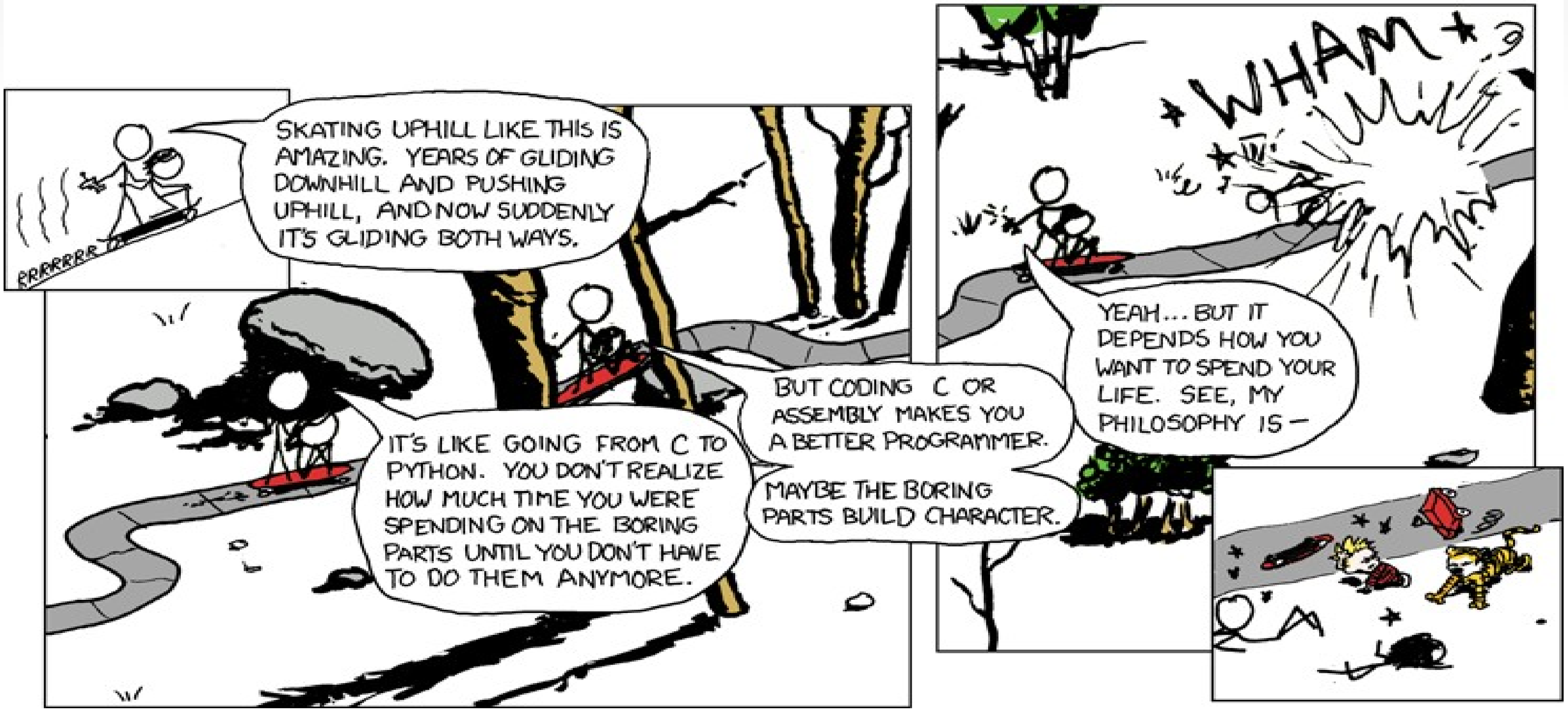
- For more details on how to use ipython and what it can do for you see:
  http://ipython.org/ipython-doc/stable/interactive/tutorial.html

  Go through the Python introduction provided on Study Direct!

  Try all the examples and do the exercises contained in it.

The joys and dangers of Python programming ...


CHECK IT OUT! AN ELECTRIC LONGBOARD!
SWEET!
RRRR
I FEEL LIKE WE'RE MISSING SOMETHING...
YEAH...
RRRR


RRRRRRR
SKATING UPHILL LIKE THIS IS AMAZING. YEARS OF GLIDING DOWNHILL AND PUSHING UPHILL, AND NOW SUDDENLY IT'S GLIDING BOTH WAYS.
IT'S LIKE GOING FROM C TO PYTHON. YOU DON'T REALIZE HOW MUCH TIME YOU WERE SPENDING ON THE BORING PARTS UNTIL YOU DON'T HAVE TO DO THEM ANYMORE.
BUT CODING C OR ASSEMBLY MAKES YOU A BETTER PROGRAMMER.
MAYBE THE BORING PARTS BUILD CHARACTER.
YEAH... BUT IT DEPENDS HOW YOU WANT TO SPEND YOUR LIFE. SEE, MY PHILOSOPHY IS —
WHAM

Very brief intro to linear algebra

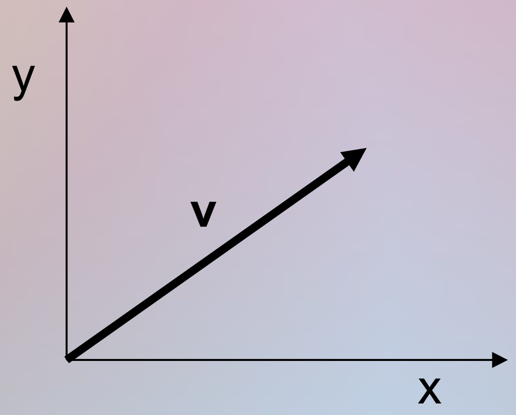
Based on: Linear Algebra: A gentle introduction
by Shivkumar Kalyanaraman

What is a Vector ?

- A vector is a *directed line segment in N-dimensions*! (has "length" and "direction")

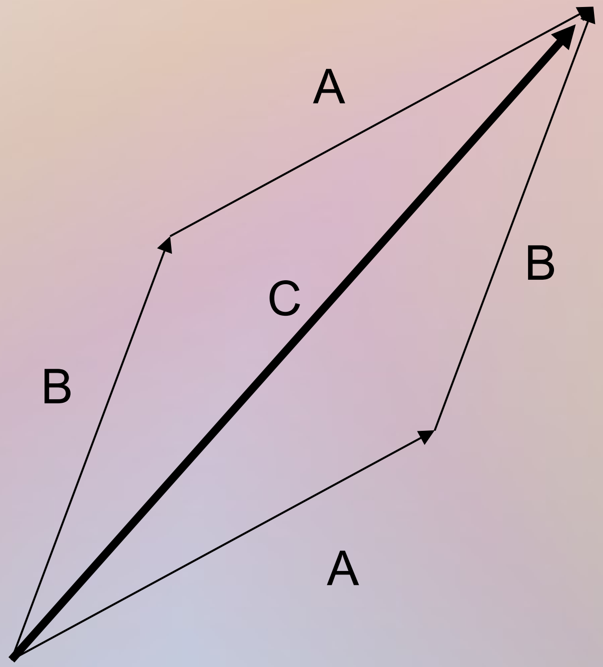
- Basic idea: convert geometry in higher dimensions into algebra!
  - Once you define a "nice" *basis* along each dimension: x-, y-, z-axis …
  - Vector becomes a 1 x N matrix!
  - **v** = [a b c]^T
  - Geometry starts to become linear algebra on vectors like **v**!

$$\vec{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$



Vector Addition: A+B

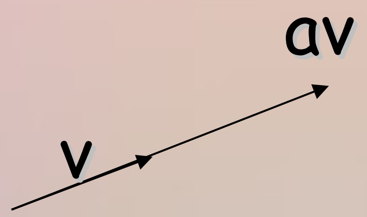
$$\mathbf{A+B} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$



A+B = C

Scalar Product: *a***v**

$$a\mathbf{v} = a(x_1, x_2) = (ax_1, ax_2)$$



Change only the length ("scaling"), but keep *direction fixed*.

Sneak peek: matrix operation (**Av**) can change *length, direction and also dimensionality*!

Vectors: Dot Product

$$A \times B = A^T B = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = ad + be + cf$$

The dot product produces a scalar (i.e. a number) and is a special case of matrix multiplication.

$$\|A\|^2 = A^T A = aa + bb + cc$$

The magnitude (length) of a vector is the dot product of a vector with itself

$$A \cdot B = \|A\| \, \|B\| \cos(\theta)$$

The dot product is also related to the angle between the two vectors

What is a Matrix?

- A matrix is a set of elements, organized into rows and columns

$$\begin{array}{cc} & \xrightarrow{\text{rows}} \\ \text{columns}\ \Big\downarrow & \begin{bmatrix} a & b \\ c & d \end{bmatrix} \end{array}$$

Basic Matrix Operations

- Addition, Subtraction, Multiplication: creating new matrices (or functions)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

Just add elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

Just subtract elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Multiply each row by each column

Matrix Times Matrix

$$\mathbf{L} = \mathbf{M} \cdot \mathbf{N}$$

$$\begin{bmatrix} l_{11} & \boxed{l_{12}} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} = \begin{bmatrix} \cancel{m_{11}} & \cancel{m_{12}} & \cancel{m_{13}} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \cdot \begin{bmatrix} n_{11} & \cancel{n_{12}} & n_{13} \\ n_{21} & \cancel{n_{22}} & n_{23} \\ n_{31} & \cancel{n_{32}} & n_{33} \end{bmatrix}$$

$$l_{12} = m_{11}n_{12} + m_{12}n_{22} + m_{13}n_{32}$$

Multiplication

- Is AB = BA? Maybe, but maybe not!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} e & f \\ g & h \end{bmatrix}=\begin{bmatrix} ae+bg & \dots \\ \dots & \dots \end{bmatrix} \quad \begin{bmatrix} e & f \\ g & h \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix}=\begin{bmatrix} ea+fc & \dots \\ \dots & \dots \end{bmatrix}$$

- Multiplication is NOT commutative!

- **Note**: If A and B both represent either pure “*rotation*” or “*scaling*” they can be interchanged (i.e. AB = BA)

Matrix operating on vectors

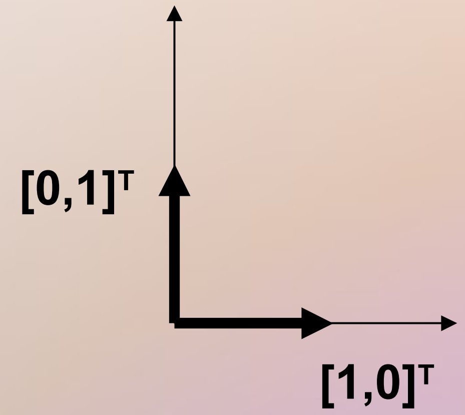
- Matrix is like a <u>function</u> that <u>transforms the vectors on a plane</u>
- Matrix operating on a general point => transforms x- and y-components
- *System of linear equations*: matrix is just the bunch of coeffs !

- $x' = ax + by$
- $y' = cx + dy$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

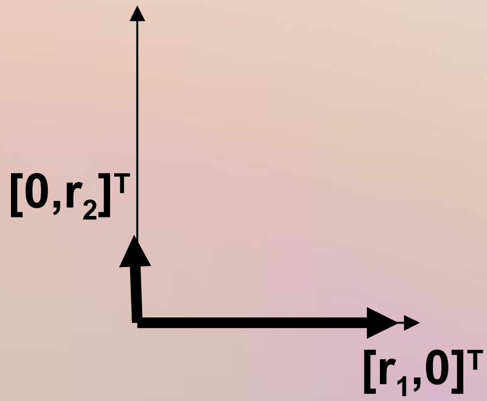
Matrices: Scaling, Rotation, Identity

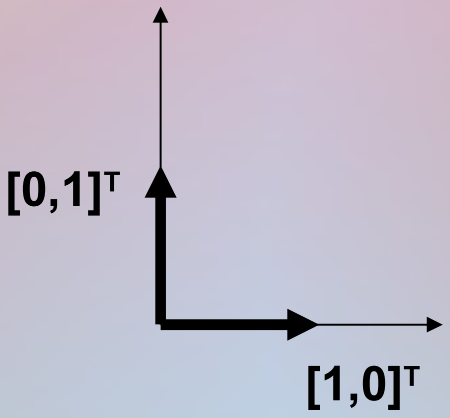
- Pure scaling, no rotation => "**diagonal** matrix" (note: x-, y-axes could be scaled differently!)
- Pure rotation, no stretching => "**orthogonal** matrix" **O**
- **Identity** ("do nothing") matrix = unit scaling, no rotation!



$$\xrightarrow{\begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix}}$$

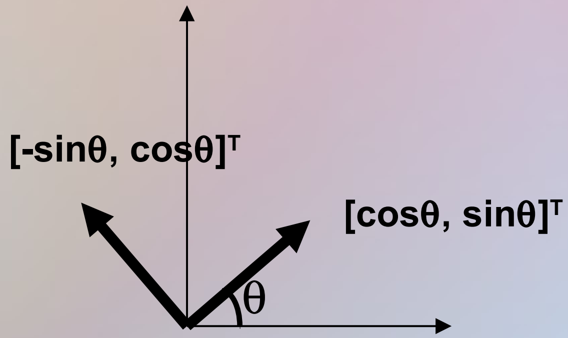
scaling



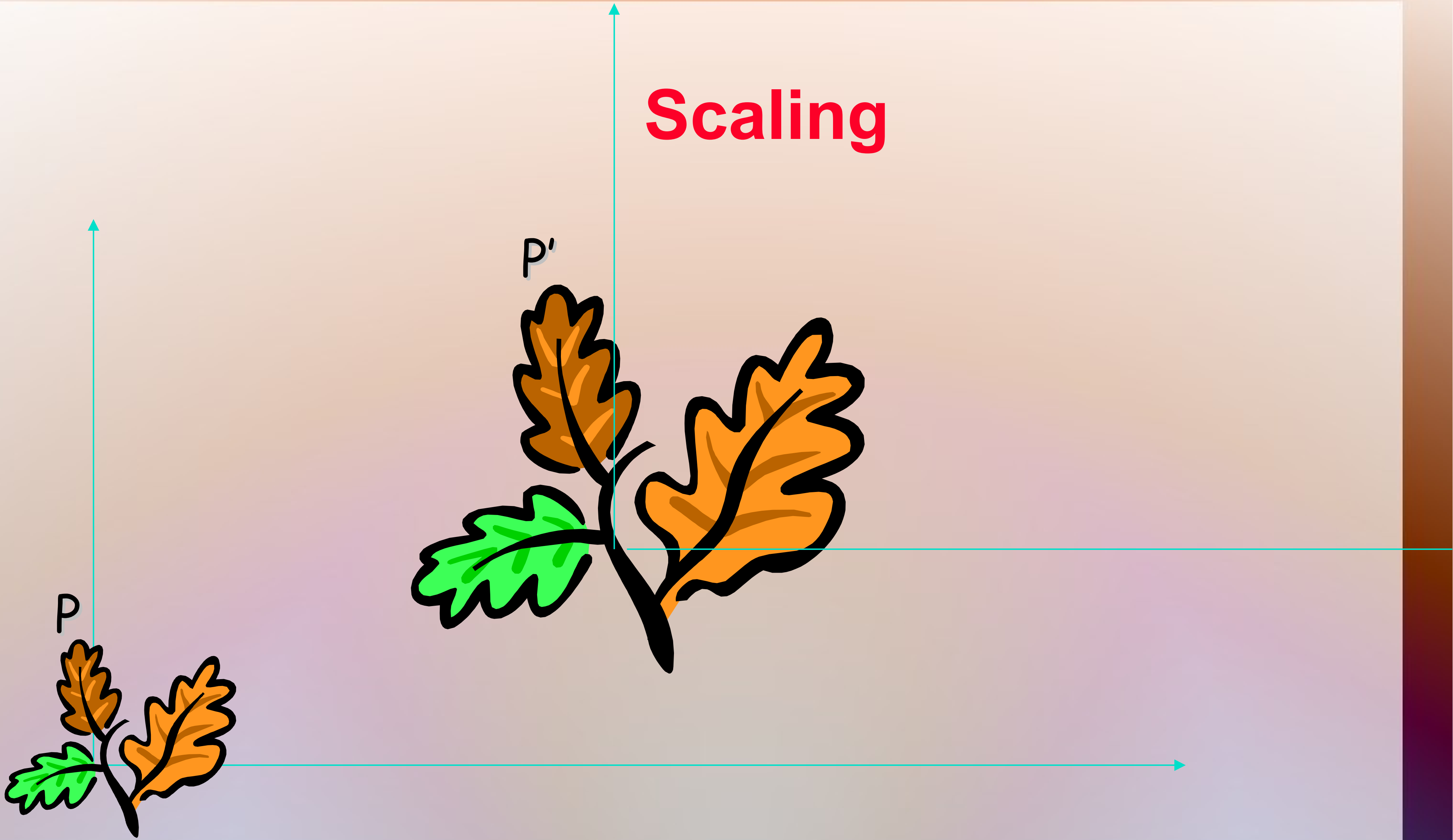


$$\xrightarrow{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}}$$

rotation



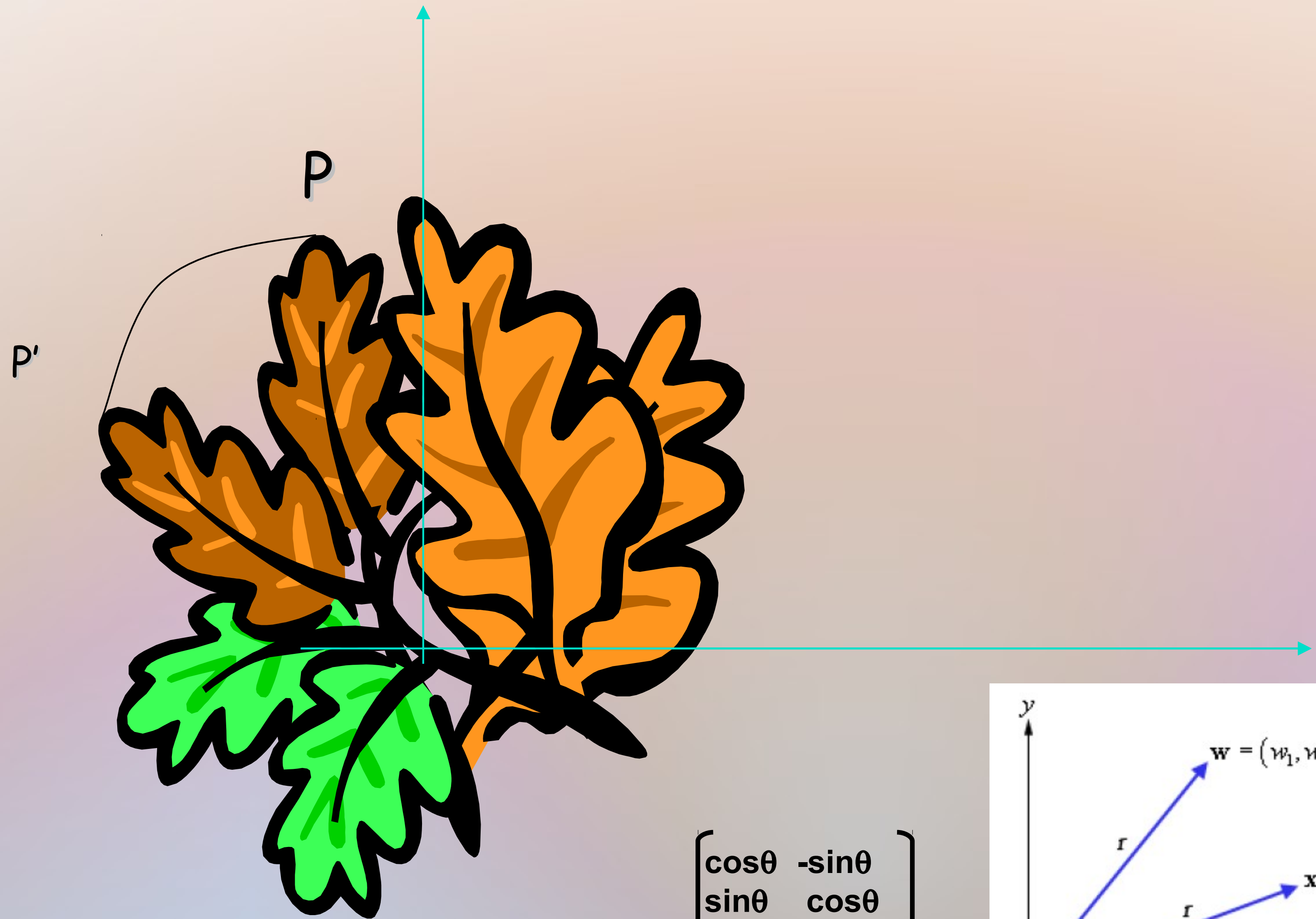
Scaling



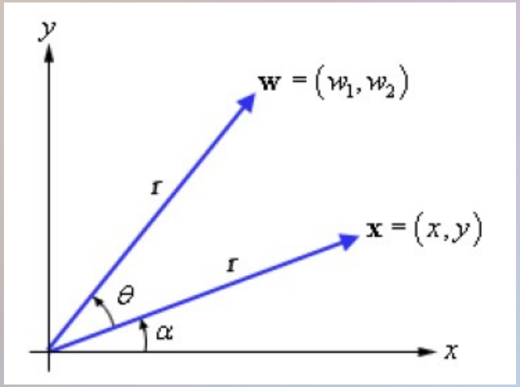
$$\begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}$$

a.k.a: dilation (r >1), contraction (r <1)

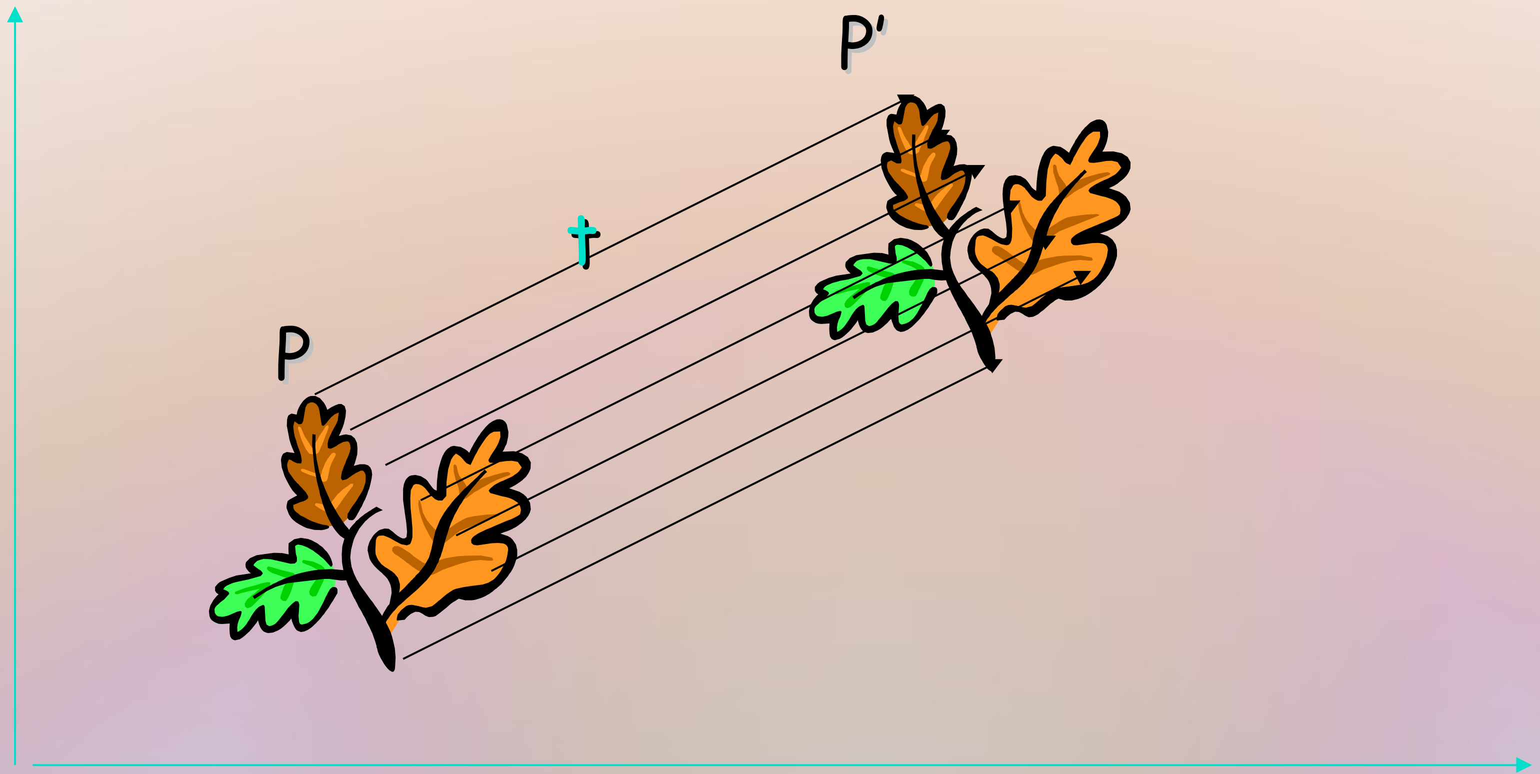
Rotation



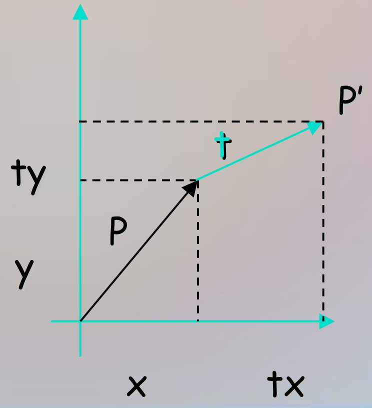
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



2D Translation



$$\mathbf{P}' = (x + t_x, y + t_y) = \mathbf{P} + \mathbf{t}$$



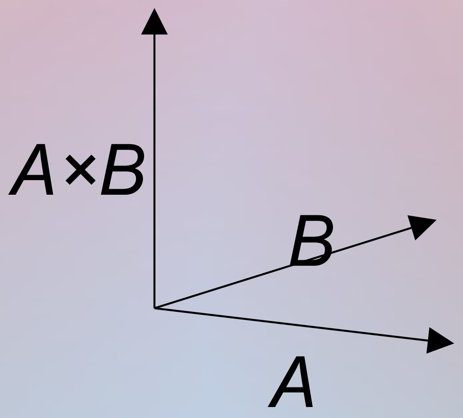
Inverse of a Matrix

- Identity matrix:
  AI = A
- Inverse exists only for square matrices that are non-singular
- Some matrices have an inverse, such that:
  AA^{-1} = I

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

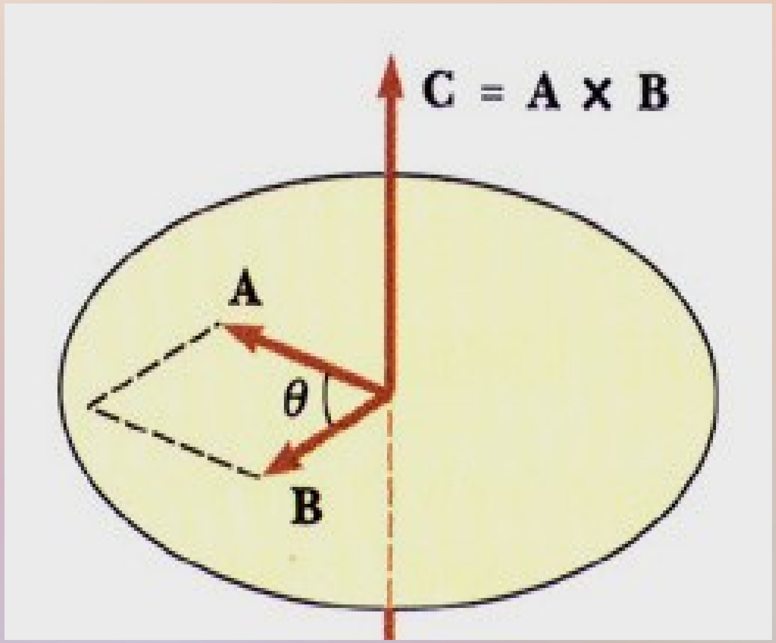
Vectors: Cross Product

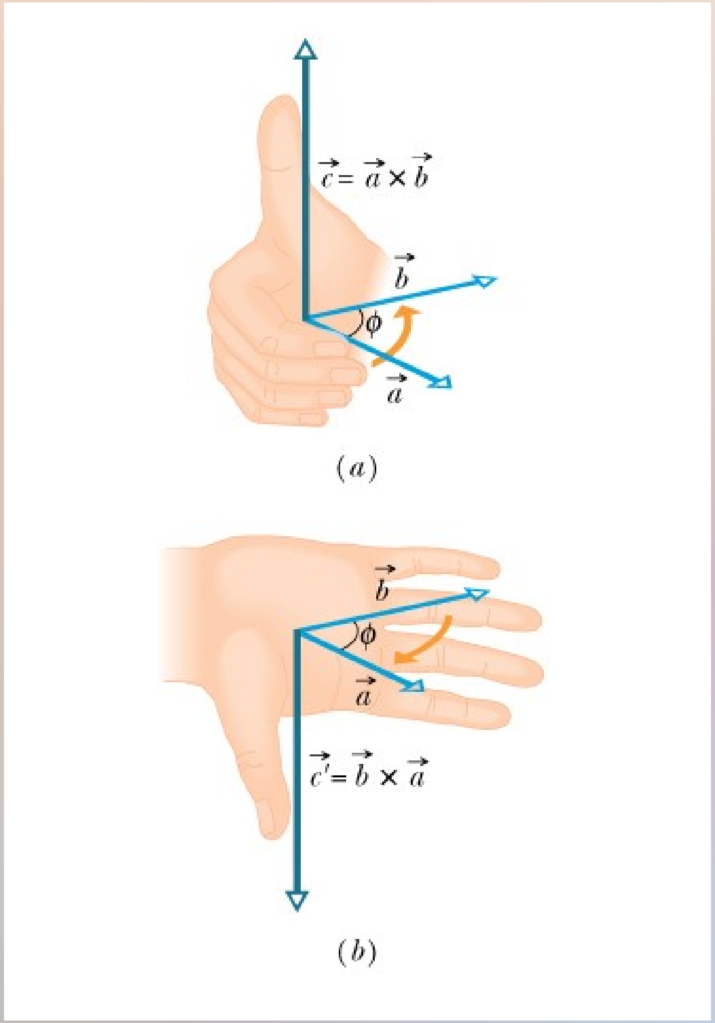
- The cross product of vectors A and B is a vector C which is perpendicular to A and B
- The magnitude of C is proportional to the sin of the angle between A and B
- The direction of C follows the **right hand rule** if we are working in a right-handed coordinate system



$$\|A \times B\| = \|A\| \, \|B\| \sin(\theta)$$

MAGNITUDE OF THE CROSS PRODUCT

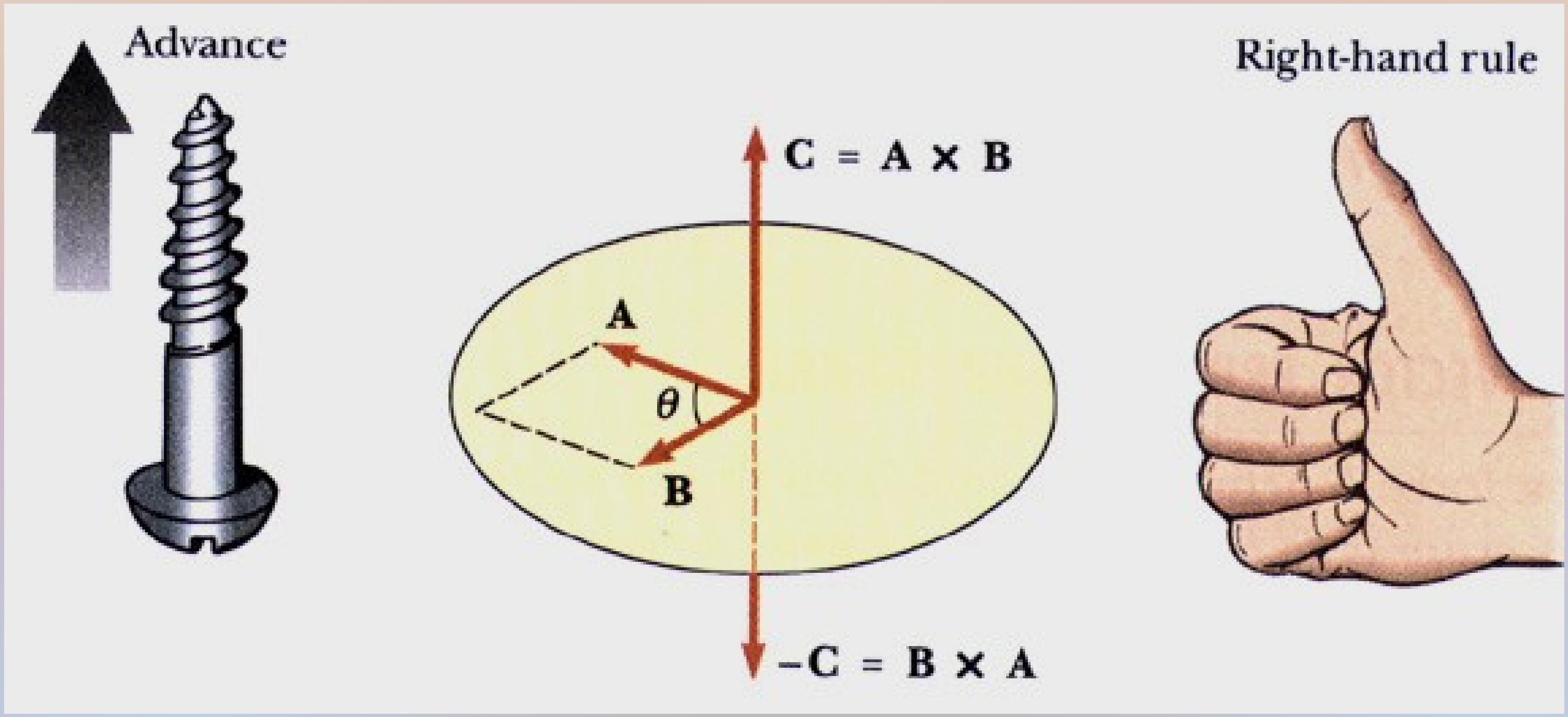




$\vec{c} = \vec{a} \times \vec{b}$
$\vec{b}$
ϕ
$\vec{a}$
(a)
$\vec{b}$
ϕ
$\vec{a}$
$\vec{c}' = \vec{b} \times \vec{a}$
(b)

DIRECTION OF THE CROSS PRODUCT

- The right hand rule determines the direction of the cross product


Advance
Right-hand rule
C = A × B
A
θ
B
−C = B × A