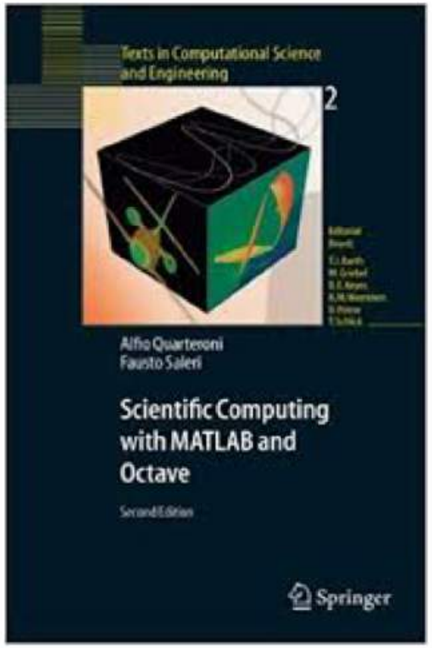
Introduction to scientific computing

Spring Semester Year 2022



Prof Joseph MORLIER

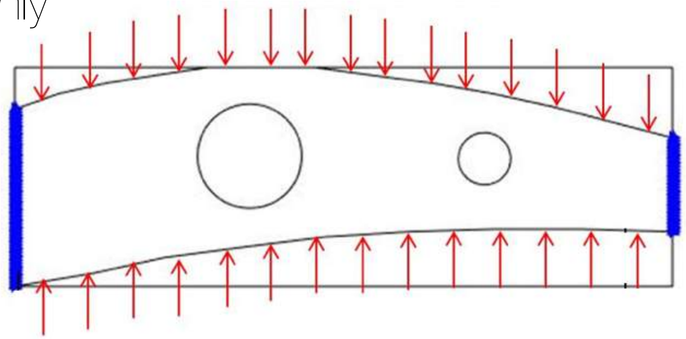
About Me? http://institut-clement-ader.org/author/jmorlier/

- Prof in Structural and Multidisciplinary Optimization
- Bat38 SUPAERO
- Research Lab (ICA)

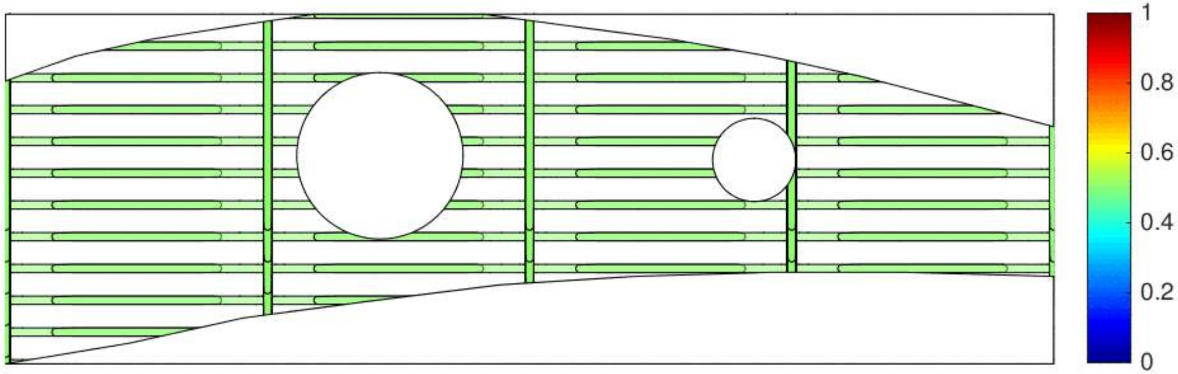
joseph.morlier@isae-supaero.fr

A typical Aerostructures

- Rib with pressure loads only



Fast convergence…

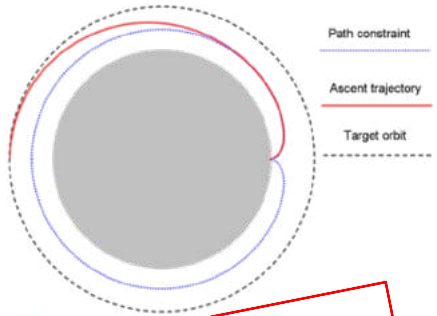


Play with

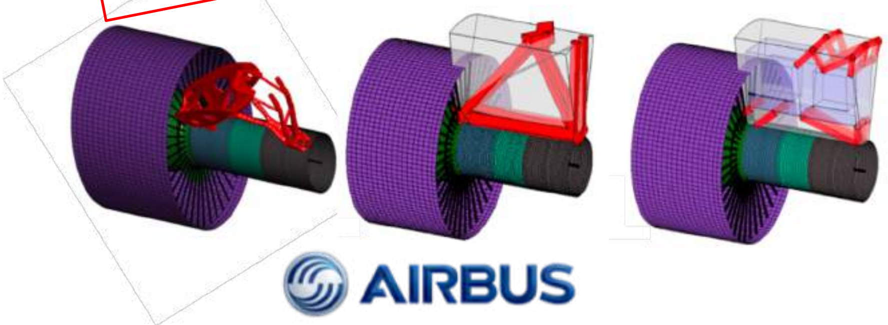
https://github.com/topggp/blog

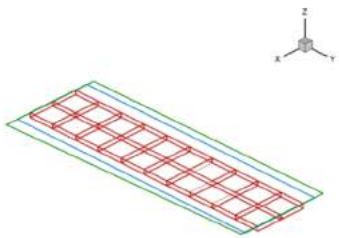
INSTITUT CLÉMENT ADER

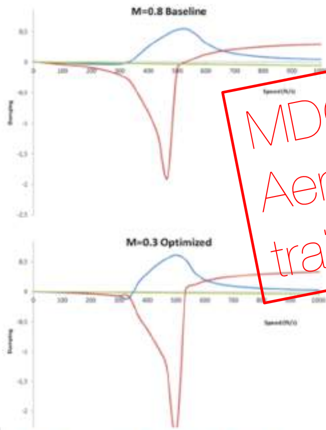
joseph.morlier@isae-supaero.fr



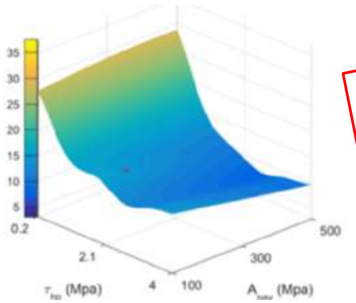
Structural Optimization & Ecodesign



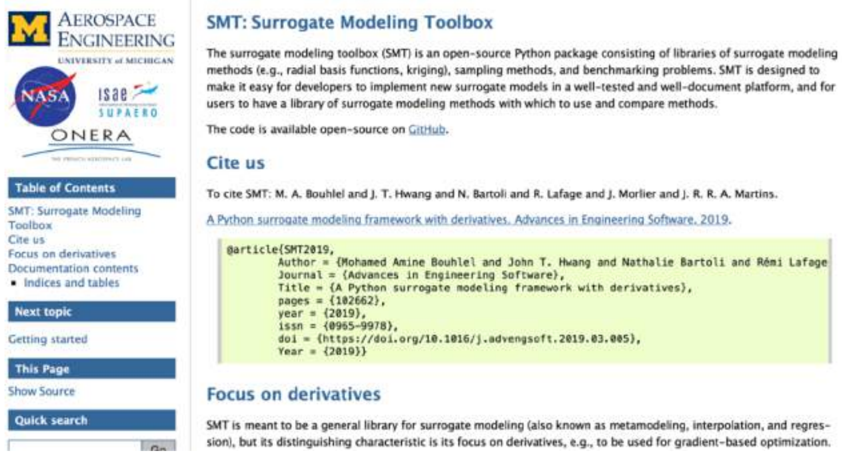




MDO with Aeroelasticity, trajectory or control



#AI4E
Artificial Intelligence For Engineers



AEROSPACE ENGINEERING
UNIVERSITY of MICHIGAN
NASA
isae SUPAERO
ONERA

Table of Contents
SMT: Surrogate Modeling Toolbox
Cite us
Focus on derivatives
Documentation contents
- Indices and tables

Next topic
Getting started

This Page
Show Source

Quick search

SMT: Surrogate Modeling Toolbox

The surrogate modeling toolbox (SMT) is an open-source Python package consisting of libraries of surrogate modeling methods (e.g., radial basis functions, kriging), sampling methods, and benchmarking problems. SMT is designed to make it easy for developers to implement new surrogate models in a well-tested and well-document platform, and for users to have a library of surrogate modeling methods with which to use and compare methods.

The code is available open-source on GitHub.

Cite us

To cite SMT: M. A. Bouhlel and J. T. Hwang and N. Bartoli and R. Lafage and J. Morlier and J. R. R. A. Martins.

A Python surrogate modeling framework with derivatives. Advances in Engineering Software, 2019.

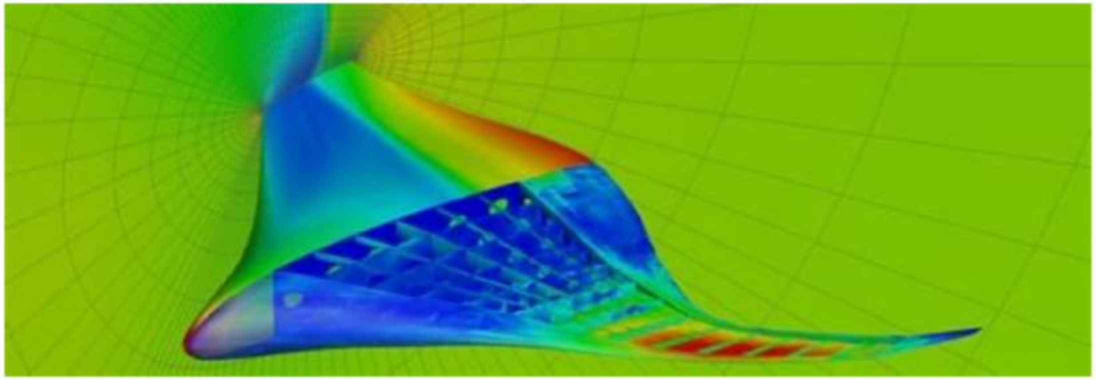
```
@article{SMT2019,
        Author = {Mohamed Amine Bouhlel and John T. Hwang and Nathalie Bartoli and Rémi Lafage
        Journal = {Advances in Engineering Software},
        Title = {A Python surrogate modeling framework with derivatives},
        pages = {102662},
        year = {2019},
        issn = {0965-9978},
        doi = {https://doi.org/10.1016/j.advengsoft.2019.03.005},
        Year = {2019}}
```

Focus on derivatives

SMT is meant to be a general library for surrogate modeling (also known as metamodeling, interpolation, and regression), but its distinguishing characteristic is its focus on derivatives, e.g., to be used for gradient-based optimization.

https://github.com/SMTorg/smt

Popularization



http://mdolab.engin.umich.edu

Optimization [MDO] for connecting people?

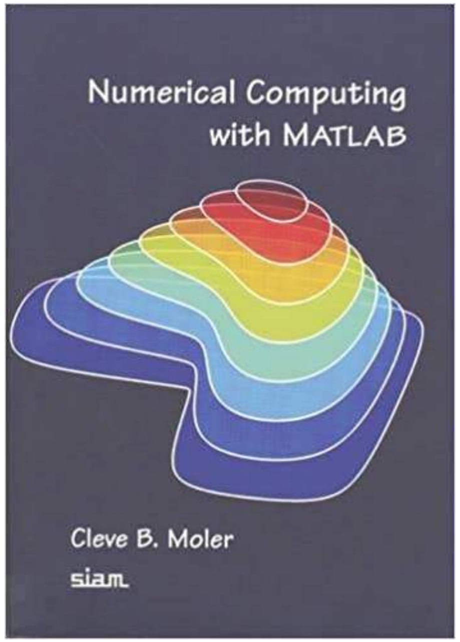
Publié le 14 février 2019 Modifier l'article | Voir les stats

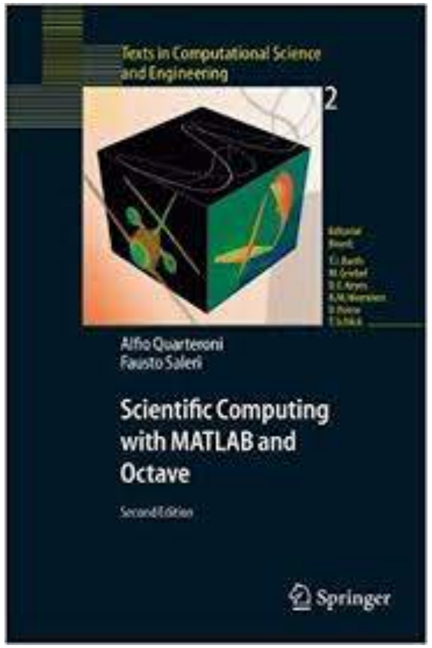
joseph morlier
Professor in Structural and Multidisciplinary Design Optimization, ... any idea?
2 articles

74 31 3 0

https://www.linkedin.com/pulse/optimization-mdo-connecting-people-joseph-morlier/

Start with scientifc computing

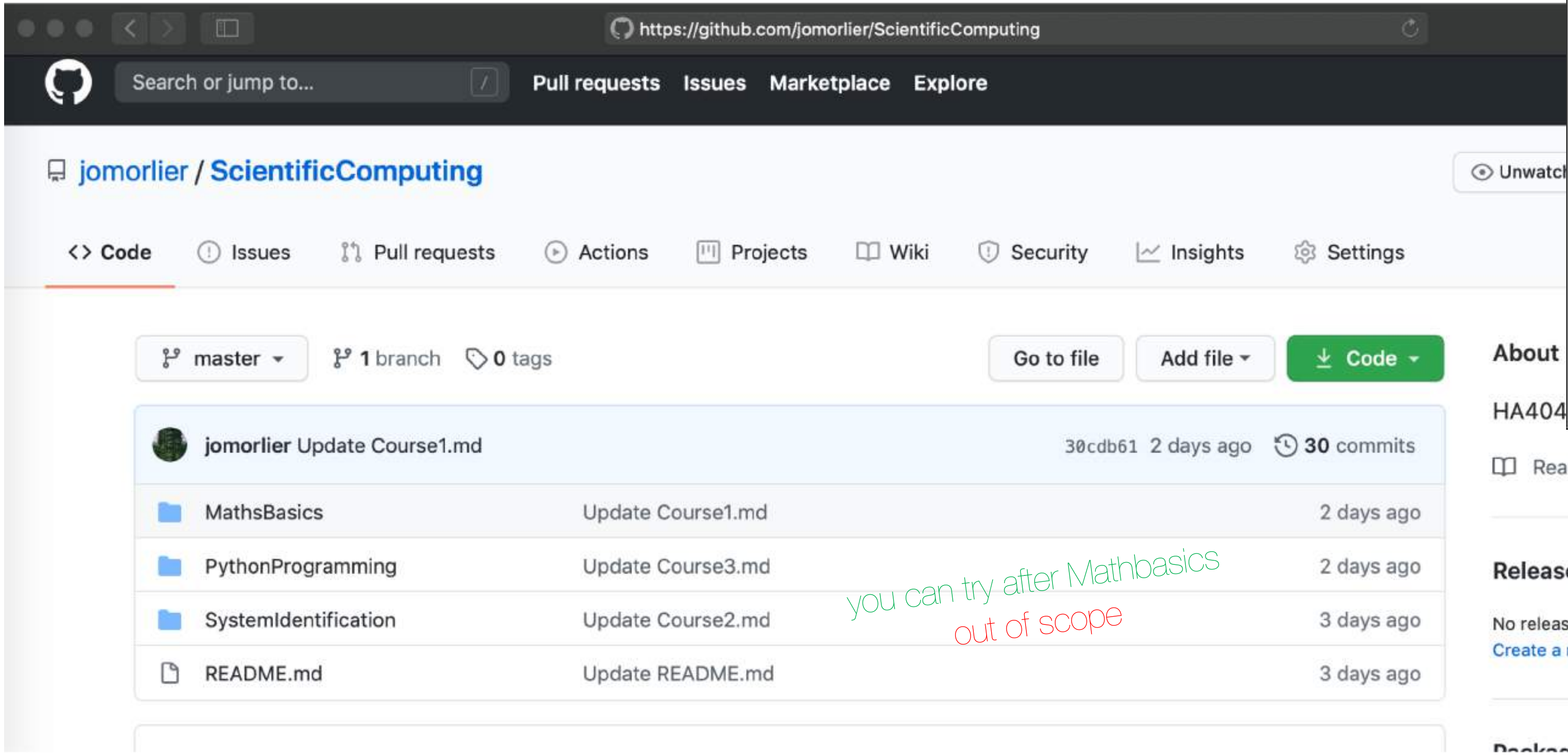




WHY?

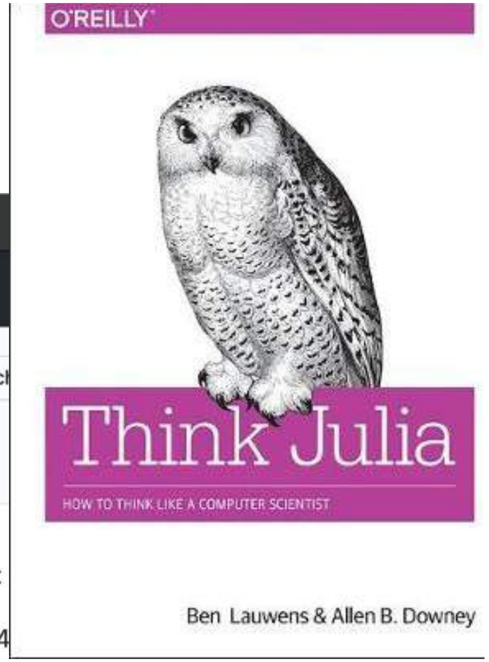
Give you the mathematical basis

Github



you can try after Mathbasics

out of scope



The way we will work together

- Basics of Scientific computing 3H with MATLAB

- MATLAB is the corporate solution, especially for engineering. It's probably still the easiest to learn for basic numerical tasks. Meticulous documentation and decades of contributed learning tools definitely matter.

https://cheatsheets.quantecon.org

Creating Matrices

	MATLAB	PYTHON	JULIA
Create a matrix	`A = [1 2; 3 4]`	`A = np.array([[1, 2], [3, 4]])`	`A = [1 2; 3 4]`
2 x 2 matrix of zeros	`A = zeros(2, 2)`	`A = np.zeros((2, 2))`	`A = zeros(2, 2)`
2 x 2 matrix of ones	`A = ones(2, 2)`	`A = np.ones((2, 2))`	`A = ones(2, 2)`
2 x 2 identity matrix	`A = eye(2, 2)`	`A = np.eye(2)`	`A = I # will adopt # 2x2 dims if demanded by # neighboring matrices`
Diagonal matrix	`A = diag([1 2 3])`	`A = np.diag([1, 2, 3])`	`A = Diagonal([1, 2, 3])`
Uniform random numbers			

DEMO

- Prof will explain you the basics of Matlab in 10'
- Use Script
- Then copy paste your script in the Livescript (equivalent to jupyter notebook)

Aim of this lecture:

- Review linear algebra & pseudoinverse & SVD

- Learn about the BIG picture of numerical methods (finite differences, finite elements) and understand their similarities, differences, and domains of applications

- Learn how to replace simple ODE /PDE Ordinary/Partial Differential Equations by their numerical approximation

Workbook= livescript to fill in matlab

- 1. Defining scalar variables
- 2. Vector operations
- 3. Matrices operation
- 4. For Loop
- 5. More programming
- 6. Optimization
- BONUS : Numerical integration in 2D
- 7. Eigenvalues and Eigenvectors
- 8. 2D Laplace Equation (analytical)
- 9. 2D Laplace Equation using 2D FD (Jacobi)
- 10. 1D Boundary Value Problem (1D FD)
- BONUS Intro to FEA

AMATH 301
Beginning Scientific Computing*

J. Nathan Kutz†

January 5, 2005

Abstract

This course is a survey of basic numerical methods and algorithms used for linear algebra, ordinary and partial differential equations, data manipulation and visualization. Emphasis will be on the implementation of numerical schemes to practical problems in the engineering and physical sciences. Full use will be made of MATLAB and its programming functionality.

Matrices operation

- Exercices:

1. Defining scalar variables
2. Vector operations
3. Matrices operation

Matrices operation

Matrices : Define the following

$$A = [a_{ij}]_{M\times N} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & & \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{pmatrix} \qquad M\times N \text{ matrix}$$

i-row
j-column

Example

$$A = \begin{pmatrix} -2 & 4 & 9 \\ 5 & -7 & 1 \\ 0 & -3 & 8 \\ 4 & 6 & -5 \end{pmatrix}$$

row vectors: $v_1 = (-2 \;\; 4 \;\; 9)$
$v_2 = (5 \;\; -7 \;\; 1)$
$v_3 = (0 \;\; -3 \;\; 8)$
$v_4 = (4 \;\; 6 \;\; -5)$

column vectors: $c_1 = \begin{pmatrix} -2 \\ 5 \\ 0 \\ 4 \end{pmatrix}$ $c_2 = \begin{pmatrix} 4 \\ -7 \\ -3 \\ 6 \end{pmatrix}$ $c_3 = \begin{pmatrix} 9 \\ 1 \\ 8 \\ -5 \end{pmatrix}$

- $A = B \rightarrow$ only if $a_{ij} = b_{ij}$ for all i,j
- $A + B = [a_{ij}]_{M\times N} + [b_{ij}]_{M\times N} = [a_{ij} + b_{ij}]_{M\times N}$
- $A - B = [a_{ij}]_{M\times N} - [b_{ij}]_{M\times N} = [a_{ij} - b_{ij}]_{M\times N}$
- $cA = [c a_{ij}]_{M\times N}$
- $pA + qB = [p a_{ij} + q b_{ij}]_{M\times N}$

MATLAB

Matrices operation

- $O = [0]_{m\times N} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{pmatrix}$
- $I = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} = [\delta_{ij}]_{m\times N}$ $\quad \delta_{ij} = \begin{cases} 1 & i=j \\ 0 & i\neq j \end{cases}$

<u>Example</u> $\quad A = \begin{pmatrix} -1 & 2 \\ 7 & 5 \\ 3 & -4 \end{pmatrix} \quad B = \begin{pmatrix} -2 & 3 \\ 1 & -4 \\ -9 & 7 \end{pmatrix}$

$$2A - 3B = 2\begin{pmatrix} -1 & 2 \\ 7 & 5 \\ 3 & -4 \end{pmatrix} - 3\begin{pmatrix} -2 & 3 \\ 1 & -4 \\ -9 & 7 \end{pmatrix} = \begin{pmatrix} -2 & 4 \\ 14 & 10 \\ 6 & -8 \end{pmatrix} - \begin{pmatrix} -6 & 9 \\ 3 & -12 \\ -27 & 21 \end{pmatrix} = \begin{pmatrix} 4 & -5 \\ 11 & 22 \\ 33 & -29 \end{pmatrix}$$

<u>properties</u>

$A + B = B + A$

$0 + A = A + 0$

$A - A = 0$

$(A+B)+C = A + (B+C)$

$(p+q)A = pA + qA$

$p(A+B) = pA + pB$

$p(qA) = (pq)A$

$AB \neq BA$ ← very important

Matrices operation

<u>Transpose</u>

$\vec{x}^T$ or A^T

- $\vec{x} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$ $\quad \vec{x}^T = (2 \quad 3 \quad 5)$

- $A = [a_{ij}]_{M \times N}$ $\quad A^T = [a_{ji}]_{N \times M}$

$$A = \begin{pmatrix} -2 & 5 & 12 \\ 1 & 4 & -1 \\ 7 & 0 & 6 \\ 11 & -3 & 8 \end{pmatrix} \qquad A^T = \begin{pmatrix} -2 & 1 & 7 & 11 \\ 5 & 4 & 0 & -3 \\ 12 & -1 & 6 & 8 \end{pmatrix}$$

4×3 $\quad\quad 3 \times 4$

- Symmetric Matrix (Hermitian or Self-Adjoint)

$$A = A^T$$

$$A = \begin{pmatrix} 1 & -7 & 4 \\ -7 & 2 & 0 \\ 4 & 0 & 3 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & -7 & 4 \\ -7 & 2 & 0 \\ 4 & 0 & 3 \end{pmatrix} = A$$

Matrices operation

matrix multiplication

Consider the two matric

$A = [a_{ik}]_{m\times n}$ m*n

$B = [b_{kj}]_{n\times p}$ n*p

then

$AB = C = [c_{ij}]_{m\times p}$ m*p $\Rightarrow$ columns of A must equal rows of B

Example

$A = \begin{pmatrix} 2 & 3 \\ -1 & 4 \end{pmatrix}$ $B = \begin{pmatrix} 5 & -2 & 1 \\ 3 & 8 & -6 \end{pmatrix}$

$$AB = \begin{pmatrix} 2 & 3 \\ -1 & 4 \end{pmatrix}\begin{pmatrix} 5 & -2 & 1 \\ 3 & 8 & -6 \end{pmatrix}$$

$$= \begin{pmatrix} 5\cdot 2 + 3\cdot 3 & -2\cdot 2 + 3\cdot 8 & 2\cdot 1 + 3\cdot -6 \\ -1\cdot 5 + 3\cdot 4 & -1\cdot -2 + 4\cdot 8 & -1\cdot 1 + 4\cdot -6 \end{pmatrix}$$

$$= \begin{pmatrix} 10+9 & -4+24 & 2-18 \\ -5+12 & 2+32 & -1-24 \end{pmatrix}$$

$$= \begin{pmatrix} 19 & 20 & -16 \\ 7 & 34 & -25 \end{pmatrix}$$

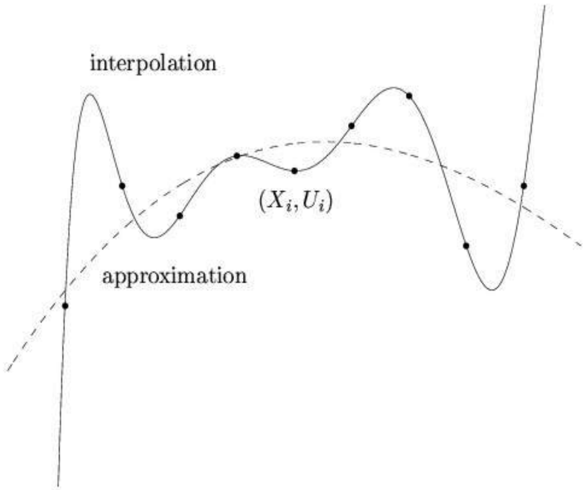
$$\boxed{\begin{array}{l} \text{Note: } AB \neq BA \text{ in general} \\ \text{if } AB = BA \Rightarrow \text{then } A \text{ and } B \text{ commute} \end{array}}$$

Interpolation

- Exercices

3. Matrices operations

Interpolation, approximation & extrapolation…



a priori basis functions

$$u(x) \approx u^h(x) = \sum_{j=0}^{n} a_j \; \phi_j(x)$$

Unknowns paramters

Interpolation :

The fonction $u^h(x)$ pass exactly through the points. Interpolated values between the points and extrapolated values outside the range .

Approximation :

The fonction $u^h(x)$ does not pass through the points, but comes close according to a criterion to define

Interpolation
(ONLY 4 LINEAR kernels)

Trouver $(a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$ tels que

$$\sum_{j=0}^{n} \underbrace{a_j \, \phi_j(X_i)}_{u^h(X_i)} = U_i \qquad\qquad i = 0, 1, \ldots, n.$$

$$\begin{bmatrix} \phi_0(X_0) & \phi_1(X_0) & \ldots & \phi_n(X_0) \\ \phi_0(X_1) & \phi_1(X_1) & \ldots & \phi_n(X_1) \\ \phi_0(X_2) & \phi_1(X_2) & \ldots & \phi_n(X_2) \\ \phi_0(X_3) & \phi_1(X_3) & \ldots & \phi_n(X_3) \\ \phi_0(X_4) & \phi_1(X_4) & \ldots & \phi_n(X_4) \\ \vdots & \vdots & & \vdots \\ \phi_0(X_n) & \phi_1(X_n) & \ldots & \phi_n(X_n) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ \vdots \\ U_n \end{bmatrix}$$

Polynomial interpolation

[1 1 1 ….1]'
is X^{?} ?

$$\phi_j(x) = x^j \qquad j = 0, 1, 2, \ldots, n.$$

$$\boxed{u^h(x) = \sum_{j=0}^{n} a_j \, x^j}$$

Vandermonde Matrix

$$\begin{bmatrix} 1 & X_0 & \ldots & X_0^n \\ 1 & X_1 & \ldots & X_1^n \\ \vdots & \vdots & & \vdots \\ 1 & X_n & \ldots & X_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_n \end{bmatrix}$$

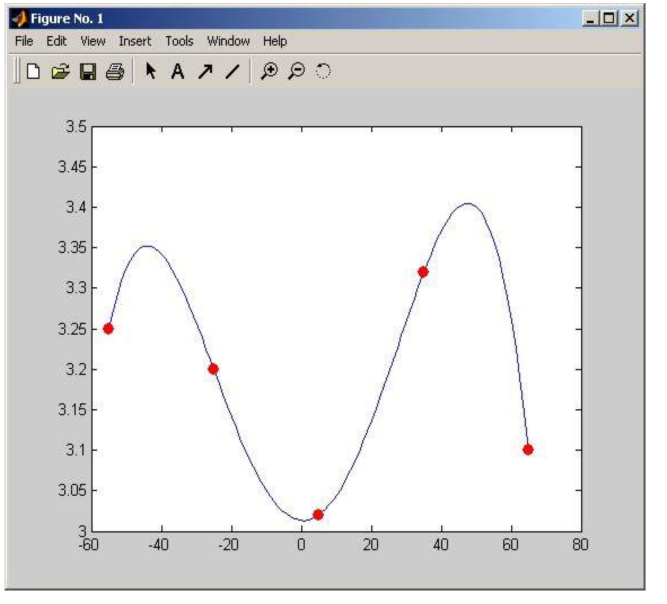
Uniqueness of the polynomial interpolation :
There is one and only one degree interpolating polynomial n which passes through at most n + 1 separate abscissa points.

Example

```
X=[-55 -25 5 35 65];
U = [3.25 3.20 3.02 3.32 3.10];
a = polyfit(X,U,4);
x = linspace(X(1),X(end),100);
uh = polyval(a,x);
plot(x,uh); hold on
plot(X,U,'r.','MarkerSize', 25);
```

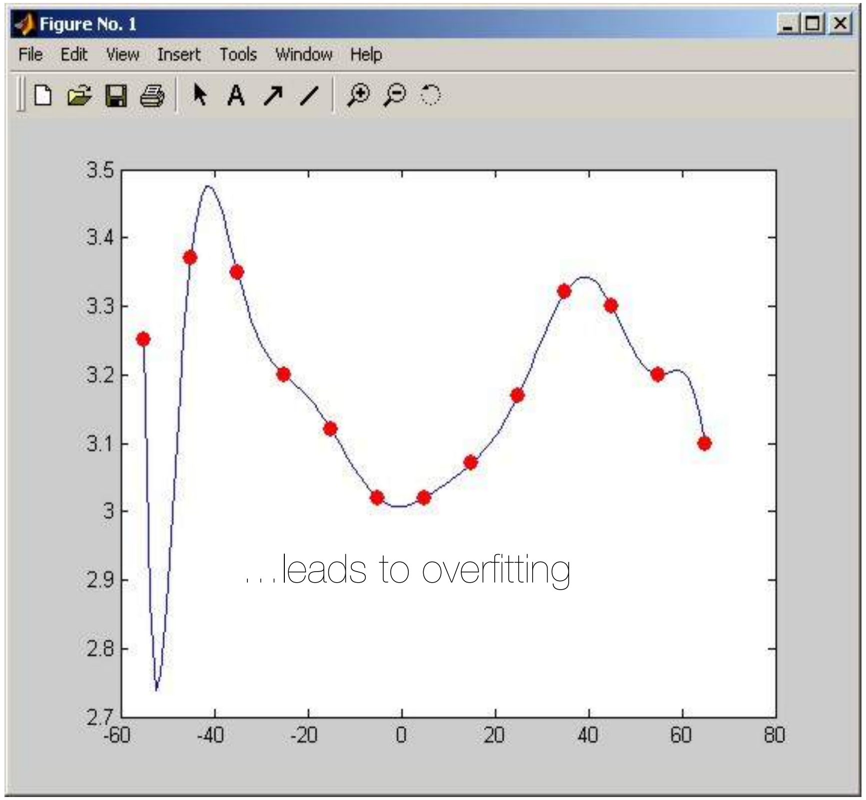
$$\begin{bmatrix} 1 & X_0 & \dots & X_0^n \\ 1 & X_1 & \dots & X_1^n \\ \vdots & \vdots & & \vdots \\ 1 & X_n & \dots & X_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_n \end{bmatrix}$$

What is the order ?



Add More points --> increase order?

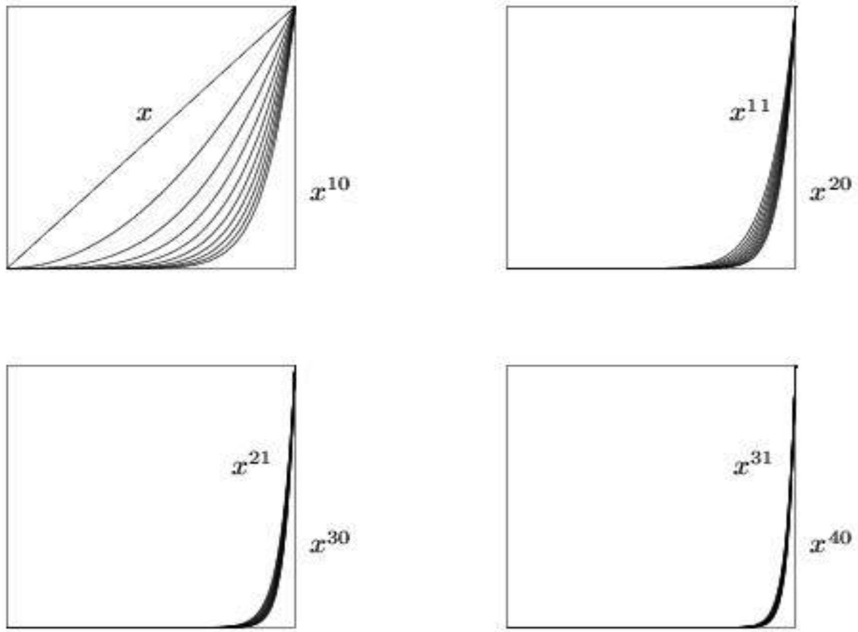
Latitude	
65	3.10
55	3.22
45	3.30
35	3.32
25	3.17
15	3.07
5	3.02
-5	3.02
-15	3.12
-25	3.20
-35	3.35
-45	3.37
-55	3.25



Polynomial matrix

$$u^h(x) = \sum_{j=0}^{n} a_j \boxed{x^j}$$

$$\begin{bmatrix} 1 & X_0 & \dots & X_0^n \\ 1 & X_1 & \dots & X_1^n \\ \vdots & \vdots & & \vdots \\ 1 & X_n & \dots & X_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} U_0 \\ U_1 \\ \vdots \\ U_n \end{bmatrix}$$



Vandermonde matrix…

Linear system becomes ill conditionned while n is increasing

Ill What??

Data

$$\begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

2 systems: Only one is well conditionned !

Exact solution:
a=2.0 et b=0.0

Ill-conditioned linear system

A linear system is ill conditioned if a small variation of the data leads to a very large variation in results .

This is a property which is directly related to the linear system and is therefore totally independent of the numerical method for solving this system .

Let's perturbate

Ill conditionned system

Perturbed solution: a=1.0 et b=1.0

Small perturbation on data

$$\begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \begin{bmatrix} a_\epsilon \\ b_\epsilon \end{bmatrix} = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$$

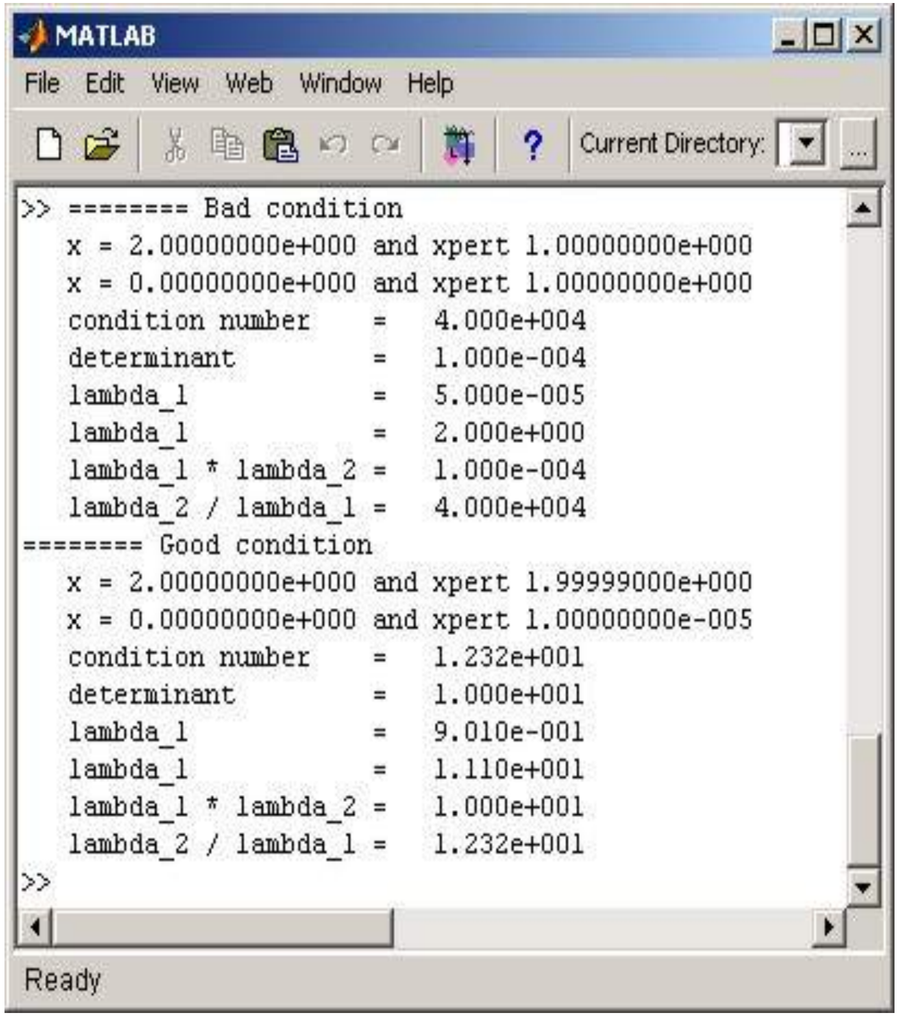
$$\begin{bmatrix} 1 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} a_\epsilon \\ b_\epsilon \end{bmatrix} = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$$

Perturbed solution: a=1.99999 et b=0.00001

Well conditionned system

Check?

```
A = [1 1;1 11];
b = [2;2];
bpert = [2;2.0001];
x = A \ b;
xpert = A \ bpert;
fprintf('   x = %12.8e and xpert %12.8e \n', [x' ; xpert']);
lambda = eig(A);
fprintf('   condition number     = %12.3e \n', cond(A));
fprintf('   determinant          = %12.3e \n',
det(A));  fprintf('             lambda_1     = %12.3e \n', lambda(1));
fprintf('   lambda_1           = %12.3e \n', lambda(2));
fprintf('   lambda_1 * lambda_2 = %12.3e \n',
lambda(1)*lambda(2));  fprintf('  lambda_2 / lambda_1 = %12.3e \n',
lambda(2)/lambda(1));
```



In Matlab

```
x = inv(A) * b;
```

```
x = A\b;
```

On résout un système linéaire,
on ne l'inverse jamais….
(J. Meinguet)

A frequent misuse of **inv** arises when solving the system of linear equations . One way to solve this is with

x = inv(A)*b

A better way, from both an execution time and numerical accuracy standpoint, is to use the matrix division operator

x = A\b

This produces the solution using Gaussian elimination, without forming the inverse.

Example 3

- spring is a mechanical element which, for the simplest model, is characterized by a linear force deformation Relationship

$$F = kx,$$

- F being the force loading the spring, k the spring constant or stiffness and x the spring deformation. In reality the linear force /deformation relationship is only an approximation, valid for small forces and deformations.
- A more accurate relationship, valid for larger deformations, is obtained if nonlinear terms are taken into account. Suppose a spring model with a quadratic relationship

$$F = k_1 x + k_2 x^2$$

Example

Force F [N]	Deformation x [cm]
5	0.001
50	0.011
500	0.013
1000	0.30
2000	0.75

[.001 .011 .13 .3 .75]

- Using the quadratic force-deformation relationship together with the experimental data yields an overdetermined
- system of linear equations and the components of the residual are given by

$$
\begin{aligned}
r_1 &= x_1 k_1 + x_1^2 k_2 - F_1 \\
r_2 &= x_2 k_1 + x_2^2 k_2 - F_2 \\
r_3 &= x_3 k_1 + x_3^2 k_2 - F_3 \\
r_4 &= x_4 k_1 + x_4^2 k_2 - F_4 \\
r_5 &= x_5 k_1 + x_5^2 k_2 - F_5.
\end{aligned}
$$

$$
A = \begin{bmatrix} x_1 & x_1^2 \\ x_2 & x_2^2 \\ x_3 & x_3^2 \\ x_4 & x_4^2 \\ x_5 & x_5^2 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5. \end{bmatrix}
$$

Finite Differences

- Exercices:

4. For Loop
5. More programming
6. Optimization

HELP GRADIENT

Numerical Gradient

The *numerical gradient* of a function is a way to estimate the values of the partial derivatives in each dimension using the known values of the function at certain points.

For a function of two variables, $F(x,y)$, the gradient is

$$\nabla F = \frac{\partial F}{\partial x}\hat{i} + \frac{\partial F}{\partial y}\hat{j}\,.$$

The gradient can be thought of as a collection of vectors pointing in the direction of increasing values of F. In MATLAB®, you can compute numerical gradients for functions with any number of variables. For a function of N variables, $F(x,y,z,\ldots)$, the gradient is

$$\nabla F = \frac{\partial F}{\partial x}\hat{i} + \frac{\partial F}{\partial y}\hat{j} + \frac{\partial F}{\partial z}\hat{k} + \ldots + \frac{\partial F}{\partial N}\hat{n}\,.$$

Tips

- Use `diff` or a custom algorithm to compute multiple numerical derivatives, rather than calling `gradient` multiple times.

Algorithms

`gradient` calculates the *central difference* for interior data points. For example, consider a matrix with unit-spaced data, A, that has horizontal gradient `G = gradient(A)`. The interior gradient values, `G(:,j)`, are

```
G(:,j) = 0.5*(A(:,j+1) - A(:,j-1));
```

The subscript `j` varies between 2 and N–1, with `N = size(A,2)`.

`gradient` calculates values along the edges of the matrix with *single-sided differences*:

```
G(:,1) = A(:,2) - A(:,1);
G(:,N) = A(:,N) - A(:,N-1);
```

If you specify the point spacing, then `gradient` scales the differences appropriately. If you specify two or more outputs, then the function also calculates differences along other dimensions in a similar manner. Unlike the `diff` function, `gradient` returns an array with the same number of elements as the input.

Finite Differences

Forward difference $\frac{f(x+\Delta x) - f(x)}{\Delta x}$ $\mathcal{O}(\Delta x)$ error

Backward difference $\frac{f(x) - f(x-\Delta x)}{\Delta x}$ $\mathcal{O}(\Delta x)$ error

Central difference $\frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$ $\mathcal{O}(\Delta x^2)$ error

error analysis involves Taylor expansions...

can get <u>higher</u> accuracy schemes by using more points : i.e. $f(x+2\Delta x)$, $f(x-2\Delta x)$, etc.

FD

Second derivative? $f''(t)$

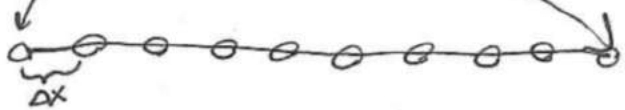
$$f(t+\Delta t) + f(t-\Delta t) = 2f(t) + \Delta t^2 \frac{d^2 f(t)}{dt^2} + \frac{\Delta t^4}{4!}\left(\frac{d^4 f(t)}{dt^4}\right) + O(\Delta t^6).$$

$$\frac{d^2 f}{dt^2}(t) = \frac{f(t+\Delta t) - 2f(t) + f(t-\Delta t)}{\Delta t^2} + O(\Delta t^2)$$

(looks a lot like what we would get if we "finite differenced" starting with $f'(x)$, $f'(x+\Delta x)$, $f'(x-\Delta x)$...)

Central difference is generally better (when possible!):

- not possible when computing $f'(t)$ in real-time
- not possible when computing $f'(x)$ at boundaries of x data



See Complexstep

Eigen Analysis

- Exercices:

7. Eigenvalues and Eigenvectors

Eigen Analysis

Eigenvalues & Eigenvectors

'Eigen' = latent or characteristic

$$\boxed{Ax = \lambda x \quad \text{for special vectors } x \text{ and special values } \lambda.}$$

Eigenvalue eq$^{\underline{n}}$ for single eigen pair (x, λ).

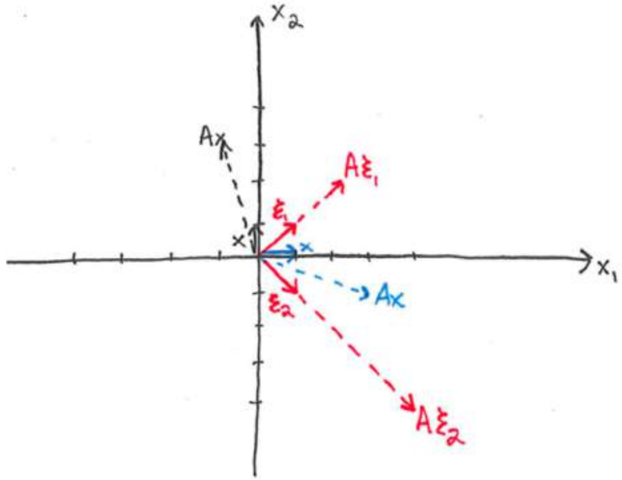
Eigen Analysis

Example : $A = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix}$

try $\underline{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow A\underline{x} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$

try $\underline{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow A\underline{x} = \begin{bmatrix} -1 \\ 3 \end{bmatrix}$

$Ax = \lambda x = \lambda I x$ identity matrix



Special $\xi_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow A\xi_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$\lambda_1 = 2$

Special $\xi_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow A\xi_2 = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$

$\lambda_2 = 4$

$Ax = \lambda x = \lambda I x$ identity matrix

Exactly 2 eigenvalues λ_1, λ_2
and 2 eigenvectors ξ_1, ξ_2.

Eigen

Eigenvalues & Eigenvectors in general:

$$Ax = \lambda x = \lambda I x$$

identity matrix

$$(A - \lambda I)\underline{x} = \underline{0}$$

Case 1: $\underline{x} = \underline{0}$ (not interesting)

Case 2: $\underline{x} \neq \underline{0}$ and $\det(A - \lambda I) = 0$

"$A - \lambda I$" is singular

meaning that it maps some vectors to $\underline{0}$.

$$\boxed{\det(A - \lambda I) = 0}$$

polynomial equation whose roots are eigenvalues!

Characteristic Equation

Eigen

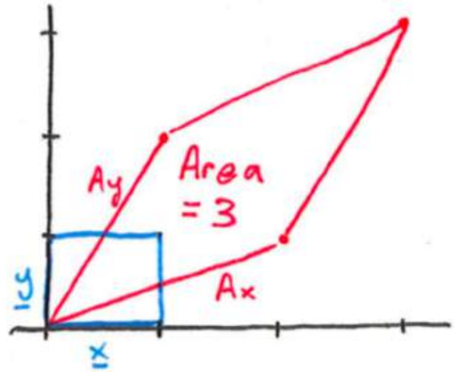
Remember 3×3 determinant...

$$\det\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = b_{11}\cdot\det\begin{pmatrix} b_{22} & b_{23} \\ b_{32} & b_{33} \end{pmatrix} - b_{12}\cdot\det\begin{pmatrix} b_{21} & b_{23} \\ b_{31} & b_{33} \end{pmatrix} + b_{13}\cdot\det\begin{pmatrix} b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

Determinant measures the volume of a unit cube after mapping through $\underline{\underline{A}}$

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\det(A) = 4 - 1 = \underline{\underline{3}}$$



Example: $A = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \Rightarrow A - \lambda I = \begin{bmatrix} 3-\lambda & -1 \\ -1 & 3-\lambda \end{bmatrix}$

Step 1

Compute λ

$$\det(A - \lambda I) = (3-\lambda)^2 - 1$$

$$= \lambda^2 - 6\lambda + 8 = (\lambda - 4)(\lambda - 2) = 0$$

Recall

$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$

$\det(B) = b_{11} b_{22} - b_{12} b_{21}$

$\Longrightarrow$ $\boxed{\text{eigenvalues are } \lambda_1 = 2, \ \lambda_2 = 4.}$

Step 2

compute $\underline{x}$ given λ.

$\lambda_1 = 2$: $A - 2I = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Longrightarrow x_1 = x_2$

$\boxed{\xi_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \lambda_1 = 2}$

Note $\xi = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 3 \\ 3 \end{bmatrix} \ldots$ also work...

$\lambda_2 = 4$: $A - 4I = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$

$\begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow x_1 = -x_2$

$\boxed{\xi_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \ \lambda_2 = 4}$

```
>> [T, D] = eig(A)
```

$$T = \begin{bmatrix} | & | & & | \\ \xi_1 & \xi_2 & \cdots & \xi_n \\ | & | & & | \end{bmatrix} \qquad D = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \ \lambda_n \end{bmatrix}$$

ODE

- Exercices:

8. ODE

ODE

(1) Harmonic oscillator : $\ddot{x} + x = 0$

(a) Taylor series

(b) Try $x(t) = e^{\lambda t}$

(c) Suspend Variables

$$\dot{x} = v \qquad \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix} \qquad \text{(Linear!)}$$
$$\dot{v} = -x$$

(2) Damped oscillator : $m\ddot{x} + \delta\dot{x} + kx = 0$

(a) Try $x(t) = e^{\lambda t}$ (again!)

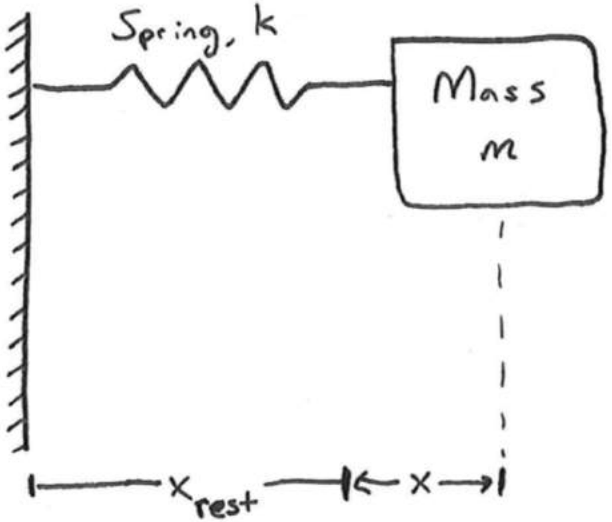
: Characteristic polynomial

$$m\lambda^2 + \delta\lambda + k = 0$$

(b) Plot in Matlab

ODE

Second-order systems



Newton's 2^{nd} Law:

$$F = ma$$
$$= m\ddot{x}$$

$$\Longrightarrow \boxed{m\ddot{x} = -kx}$$

x is the displacement of the mass from a rest position x_{rest}, where spring exerts no net force.

First consider $k = m = 1 \Longrightarrow \boxed{\ddot{x} = -x}$

ODE

$$\ddot{x} = -x$$

Method 4[redacted] : Suspend variables & solve as linear system

$$\dot{x} = v \leftarrow \text{new variable}$$

$$\dot{v} = -x$$

$$\Longrightarrow \quad \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix}$$

Much more on this later!

Method 1 : Guess!

$$x(t) = \cos(t)\, x(0)$$

$$\dot{x}(t) = -\sin(t)\, x(0)$$

$$\ddot{x}(t) = -\cos(t)\, x(0) \quad \checkmark \quad \ddot{x} = -x$$

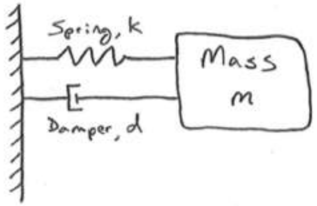
For general m, k:

$$x(t) = \cos\left(\sqrt{k/m}\; t\right) x(0)$$

so frequency of oscillation is $\omega = \sqrt{k/m}$

ODE

Example Damped Harmonic Oscillator



$F = ma$

$m\ddot{x} = -kx - d\dot{x}$

$\implies m\ddot{x} + d\dot{x} + kx = 0$

Try $x(t) = e^{\lambda t} \dots$

$\dot{x}(t) = \lambda e^{\lambda t}$

$\ddot{x}(t) = \lambda^2 e^{\lambda t}$

$\implies m\lambda^2 e^{\lambda t} + d\lambda e^{\lambda t} + k e^{\lambda t} = 0$

$\implies \left[m\lambda^2 + d\lambda + k\right] e^{\lambda t} = 0$

$\implies m\lambda^2 + d\lambda + k = 0$

Let $d/m = \zeta$ and $k/m = \omega^2$, so

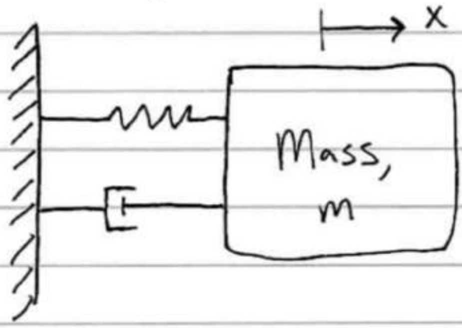
$\implies \lambda^2 + \zeta\lambda + \omega^2 = 0$

$\implies \boxed{\lambda = \dfrac{-\zeta \pm \sqrt{\zeta^2 - 4\omega^2}}{2}}$ 2 solutions (+/-) λ_1 and λ_2.

☆ $\boxed{x(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}}$ ☆ Need to initial conditions for c_1, c_2.

Example

Spring-Mass-Damper



$$m\ddot{x} = -kx - c\dot{x}$$

$$m\ddot{x} + kx + c\dot{x} = 0$$

$$\ddot{x} + \frac{k}{m}x + \frac{c}{m}\dot{x} = 0$$

If $\omega_0 = \sqrt{\frac{k}{m}}$ natural frequency

$\zeta = \frac{c}{2\sqrt{km}}$ damping ratio.

$$\boxed{\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = 0}$$

$$\boxed{\ddot{x} + 2\zeta\omega_0\dot{x} + \omega_0^2 x = 0}$$

Second order <u>linear</u> differential equation.

$$\left.\begin{aligned}\dot{x} &= v\\ \dot{v} &= -2\zeta\omega_0 v - \omega_0^2 x\end{aligned}\right\} \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{bmatrix}}\begin{bmatrix} x \\ v \end{bmatrix}$$

ω_0 & ζ determine eigenvalues of A, hence, the behavior of the system.

Cases: ① Under-damped. $\zeta < 1$

system oscillates w/ freq $\omega_d = \omega_0 \sqrt{1-\zeta^2}$

② Over-damped $\zeta > 1$

③ Critically Damped $\zeta = 1$

Lets code up forward Euler

$$x_{k+1} = (I + A\Delta t)\, x_k$$

... try $dt = .01$ $T = 10$
... compare w/ RK4

... try $dt = 0.1$, $d = 0.5$, $d = 1$, $d = 2$.

What went wrong?...

Look at eig$(I + A\Delta t)$.

Jacobi FD in 2D

- Exercices:

9. 2D Laplace Equation

Jacobi

Laplace's Equation (numerical):

① Use $u_t = \alpha \nabla^2 u$

and iterate forward... i.e. finite difference in space & time!

crudest $\frac{\partial}{\partial t}$ possible! (but it works!)

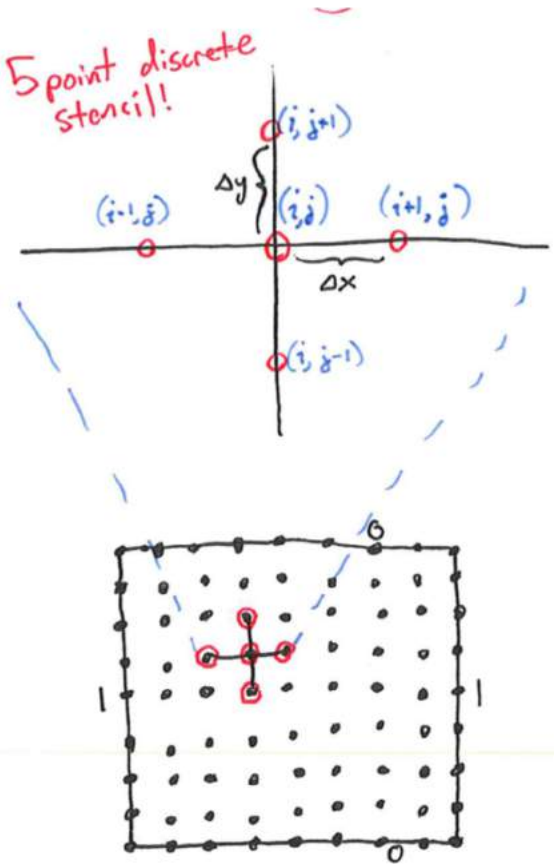
$$\frac{u(t+\Delta t) - u(t)}{\Delta t} = \alpha \nabla^2 u(t)$$

$$\Longrightarrow u(t+\Delta t) = u(t) + (\alpha \Delta t) \nabla^2 u(t)$$

Ⓐ $\nabla^2 u$ can be ~~[illegible]~~ computed using 'del2' function

Ⓑ $\nabla^2 u$ computed by hand using a stencil:

Jacobi and GS



$$\frac{\partial^2 u}{\partial x^2} \approx \frac{\dfrac{u(i+1,j) - u(i,j)}{\Delta x} - \dfrac{u(i,j) - u(i-1,j)}{\Delta x}}{\Delta x}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{\dfrac{u(i,j+1) - u(i,j)}{\Delta y} - \dfrac{u(i,j) - u(i,j-1)}{\Delta y}}{\Delta y}$$

Assume $\Delta x = \Delta y \; (= 1)$

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{\Delta x^2}\Big(u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - 4u(i,j) \Big)$$

$$= u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) - 4u(i,j)$$

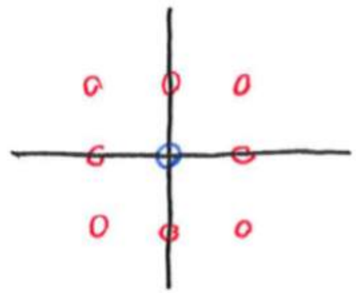
(A) 5-point stencil: (set $\nabla^2 u = 0$, solve for u_{ij})

$$u(i,j) = \frac{1}{4}\Big(u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1) \Big)$$

i.e. average neighbors.

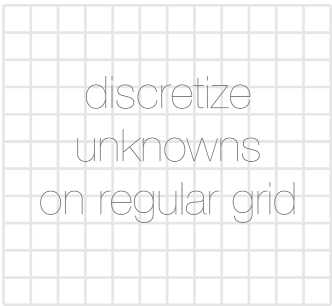
(B) 8-point stencil. Similar, average neighboring 8 pts.

$$u_{ij} = \frac{1}{8} \Sigma \text{ neighbors.}$$



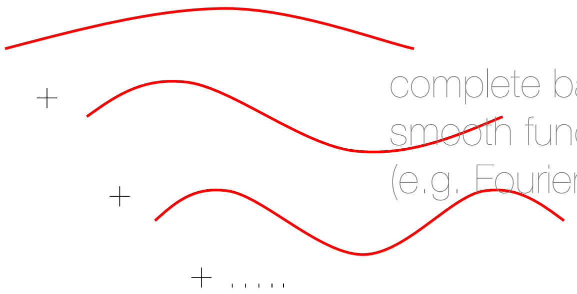
Numerical Methods: Basis Choices

finite difference



discretize
unknowns
on regular grid

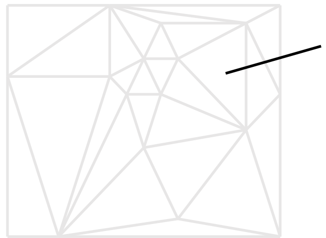
spectral methods



complete basis of
smooth functions
(e.g. Fourier series)

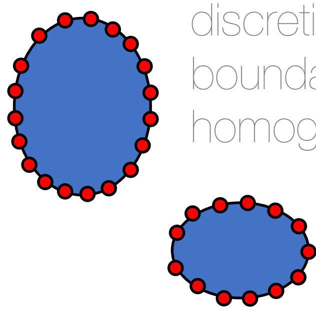
Much easier to analyze, implement,
generalize, parallelize, optimize, …

finite elements



in irregular "elements,"
approximate unknowns
by low-degree polynomial

boundary-element methods



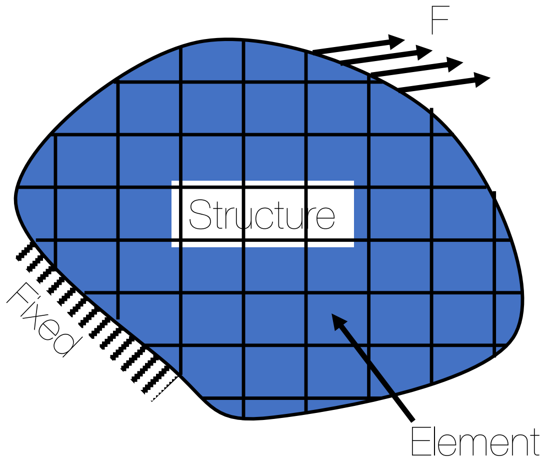
discretize only the
boundaries between
homogeneous media

…solve
integral equation
via Green's functions

Potentially much more efficient,
especially for high resolution

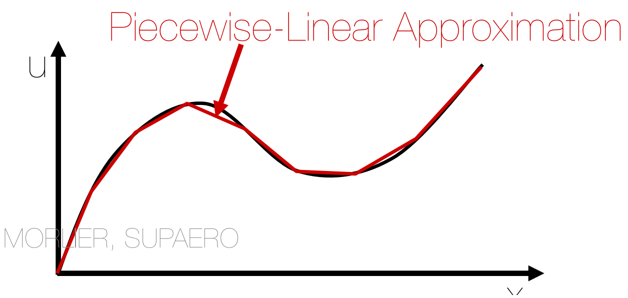
INTRODUCTION TO FINITE ELEMENT

- What is the finite element method (FEM)?
  - A technique for obtaining approximate solutions to boundary value problems.
  - Partition of the domain into a set of simple shapes (element)
  - Approximate the solution using piecewise polynomials within the element



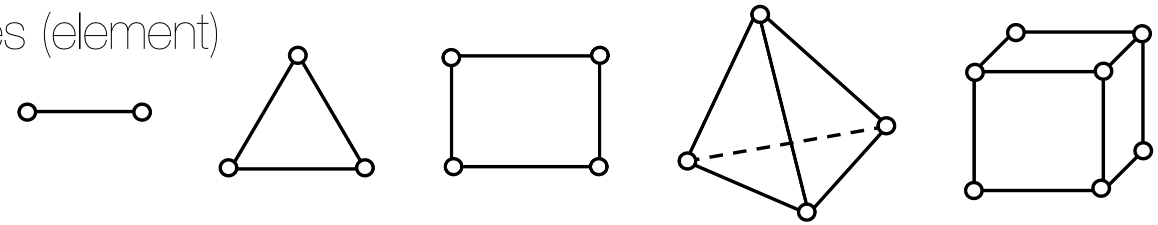
$$\begin{cases} \dfrac{\partial \sigma_{xx}}{\partial x} + \dfrac{\partial \tau_{xy}}{\partial y} + f_{vx} = 0 \\ \dfrac{\partial \tau_{xy}}{\partial x} + \dfrac{\partial \sigma_{yy}}{\partial y} + f_{vy} = 0 \end{cases}$$

$$\underline{div}\left(\underline{\underline{\sigma}}\right) + \underline{f}_{v} = \underline{0}$$

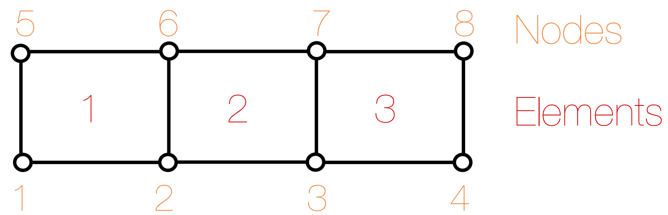


INTRODUCTION TO FEM *cont.*

- How to discretize the domain?
  - Using simple shapes (element)

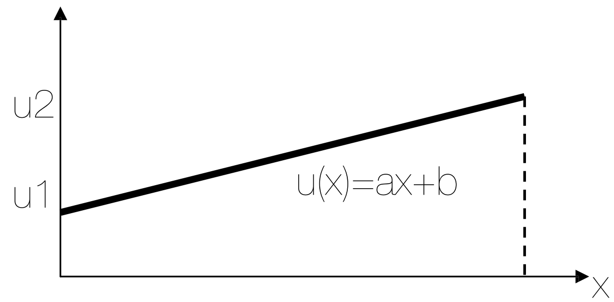


  - All elements are connected using "nodes".

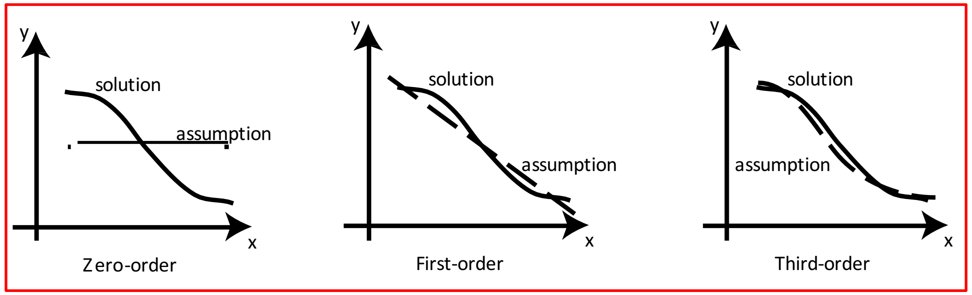


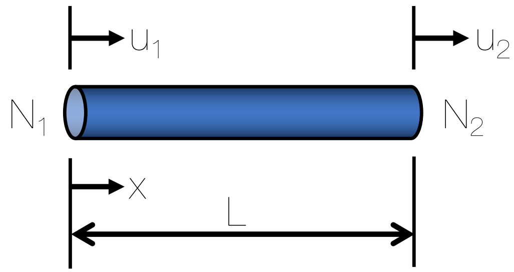
  - Solution at Element 1 is described using the values at Nodes 1, 2, 6, and 5 (Interpolation).
  - Elements 1 and 2 share the solution at Nodes 2 and 6.

INTRODUCTION TO FEM *cont.*



- Finite element analysis solves for nodal values.
  - All others can be calculated (or interpolated) from nodal solutions





  - Displacement within the element

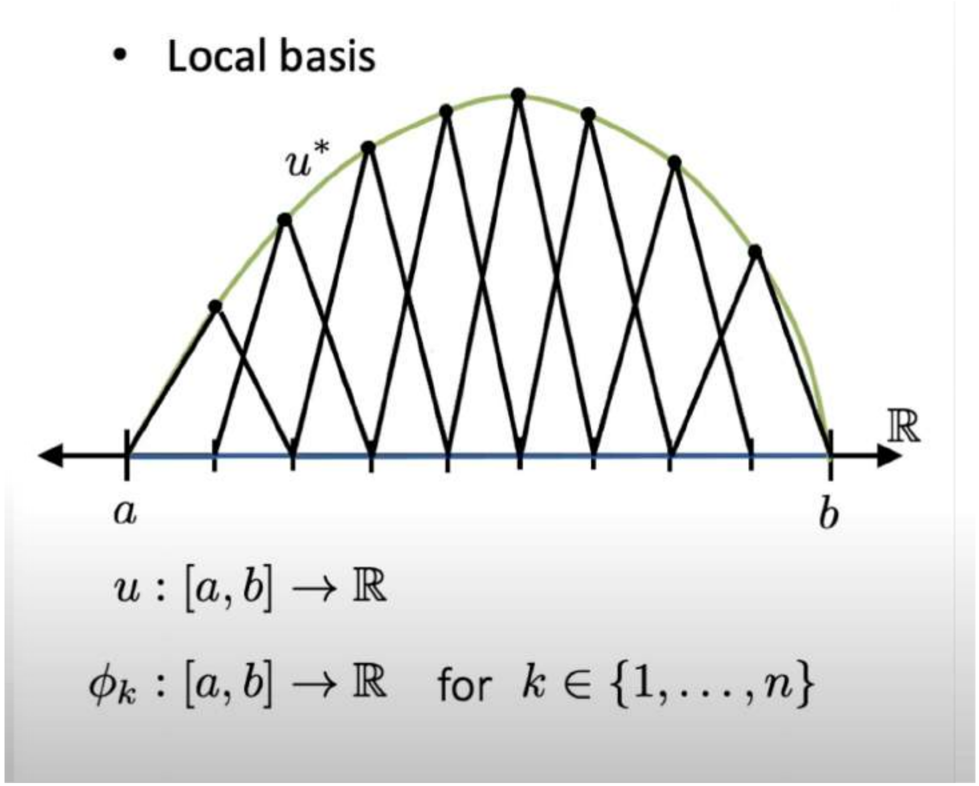
$$u(x) = a + bx = u_1 + \frac{u_2 - u_1}{L}x = \frac{L - x}{L}u_1 + \frac{x}{L}u_2$$

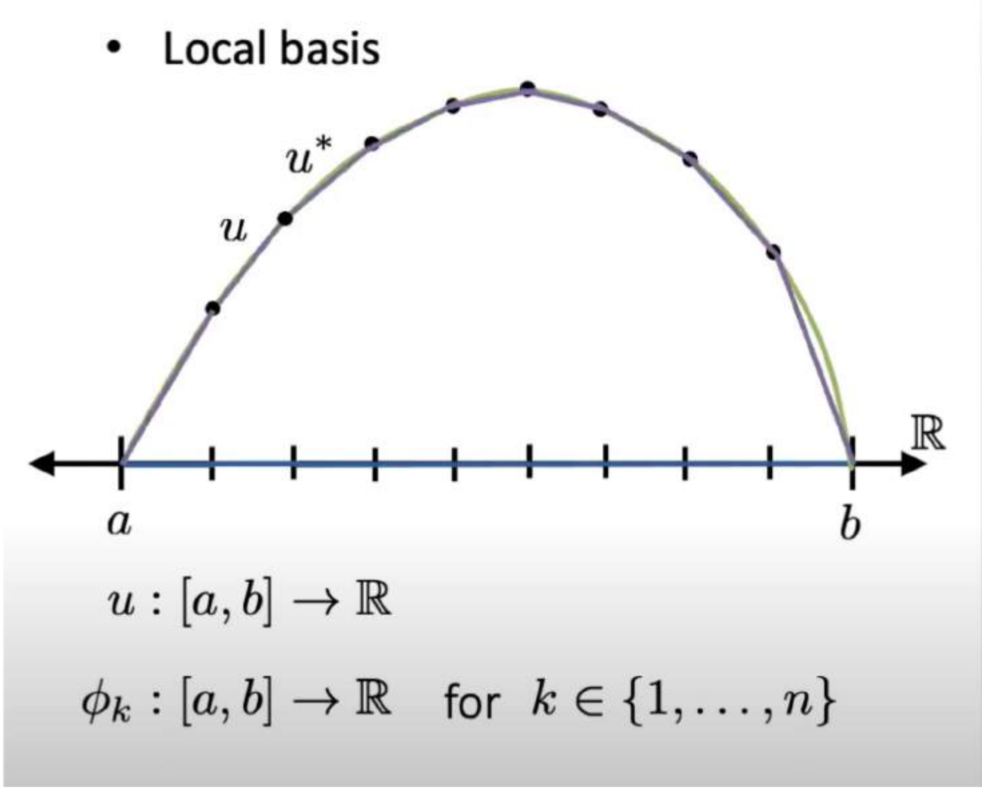
Interpolation (Shape) Function

  - Strain of the element
  - And the stress?

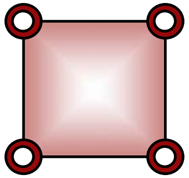
$$\varepsilon(x) = \frac{\partial u}{\partial x} = -\frac{1}{L}u_1 + \frac{1}{L}u_2$$

HAT function (the simplest shape Function)



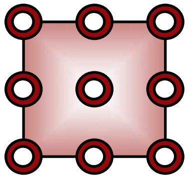


steps of a finite element simulation



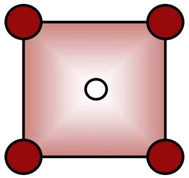
#1 modeling (boundary value problem)

$$\mathrm{div}\boldsymbol{\sigma} + \boldsymbol{f} = \mathbf{0}$$



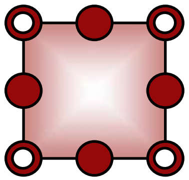
#2 pre-processing (discretization with finite elements)

#3 algebraization (system of equations)



$$\mathbf{K} \cdot \mathbf{u} = \mathbf{F}$$

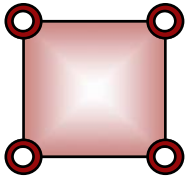
#4 solution $\mathbf{u} = \mathbf{K}^{-1} \cdot \mathbf{F}$



#5 post-processing (final calculation and visualization)

$$\boldsymbol{\sigma} = \mathbf{E} \cdot \boldsymbol{\epsilon}(\boldsymbol{u})$$

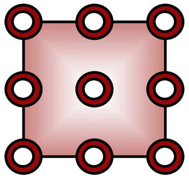
… and what can go wrong…



#1 modeling (boundary value problem)

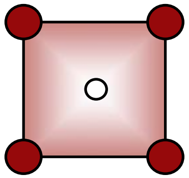
$$\mathrm{div}\boldsymbol{\sigma} + \boldsymbol{f} = \mathbf{0}$$

modeling error
(wrong equations)



#2 pre-processing (discretization with finite elements)

discretization error
(wrong elements)



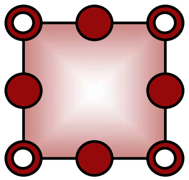
#3 algebraization (system of equations)

$$\mathbf{K} \cdot \mathbf{u} = \mathbf{F}$$

#4 solution

$$\mathbf{u} = \mathbf{K}^{-1} \cdot \mathbf{F}$$

solution error
(wrong algorithms)



#5 post-processing (final calculation and visualization)

$$\boldsymbol{\sigma} = \mathbf{E} \cdot \boldsymbol{\epsilon}(\boldsymbol{u})$$

visualization error
(wrong colorscales)

SVD

- Exercice:

BONUS image compression

SVD

1. Singular Value Decomposition (SVD)

   * Dimensionality reduction

   * Data Analysis

   * Machine Learning

2. Today:

   A. What is the SVD

   B. $X = U \Sigma V^*$

   C. Image Compression.

SVD

Generally, we are interested in analyzing a large data set $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}.$$

The columns $\mathbf{x}_k \in \mathbb{C}^n$ may be measurements from simulations or experiments. For example, columns may represent images that have been reshaped into column vectors with as many elements as pixels in the image. The column vectors may also represent the state of a physical system that is evolving in time, such as the fluid velocity at each point in a discretized simulation or at each measurement location in a wind-tunnel experiment.

The index k is a label indicating the k^{th} distinct set of measurements; for many of the examples in this book $\mathbf{X}$ will consist of a *time-series* of data, and $\mathbf{x}_k = \mathbf{x}(k\Delta t)$. Often the *state-dimension* n is very large, on the order of millions or billions in the case of fluid systems. The columns are often called *snapshots*, and m is the number of snapshots in $\mathbf{X}$. For many systems $n \gg m$, resulting in a *tall-skinny* matrix, as opposed to a *short-fat* matrix when $n \ll m$.

The SVD is a unique matrix decomposition that exists for every complex valued matrix $\mathbf{X} \in \mathbb{C}^{n\times m}$:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

where $\mathbf{U} \in \mathbb{C}^{n\times n}$ and $\mathbf{V} \in \mathbb{C}^{m\times m}$ are *unitary* matrices[1] and $\mathbf{\Sigma} \in \mathbb{C}^{n\times m}$ is a matrix with non-negative entries on the diagonal and zeros off the diagonal. Here * denotes the complex conjugate transpose[2]. As we will discover throughout this chapter, the condition that $\mathbf{U}$ and $\mathbf{V}$ are unitary is extremely powerful.

The matrix $\mathbf{\Sigma}$ has at most m non-zero elements on the diagonal, and may therefore be written as $\mathbf{\Sigma} = \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix}$. Therefore, it is possible to *exactly* represent $\mathbf{X}$ using the *reduced* SVD:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^{\perp} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \mathbf{V}^* = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V}^*.$$

SVD

The columns of $\mathbf{U}$ are called *left singular vectors* of $\mathbf{X}$ and the columns of $\mathbf{V}$ are *right singular vectors*. The diagonal elements of $\hat{\mathbf{\Sigma}} \in \mathbb{C}^{m\times m}$ are called *singular values* and the are ordered from largest to smallest.

In Matlab, the computing the SVD is straightforward:

```
>>[U,S,V] = svd(X);   % Singular Value Decomposition
```

For non-square matrices $\mathbf{X}$, the reduced SVD may be computed more efficiently using:

```
>>[Uhat,Shat,V] = svd(X,'econ');   % economy sized SVD
```

[1]A square matrix $\mathbf{U}$ is unitary if $\mathbf{U}\mathbf{U}^* = \mathbf{U}^*\mathbf{U} = \mathbb{I}$.
[2]For real-valued matrices, this is the same as the regular transpose $\mathbf{X}^T$