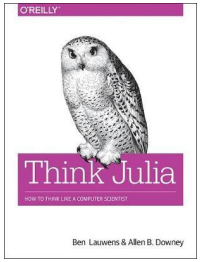
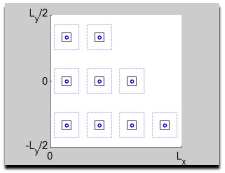
Introduction to scientific computing

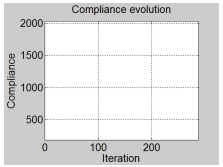
AES



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

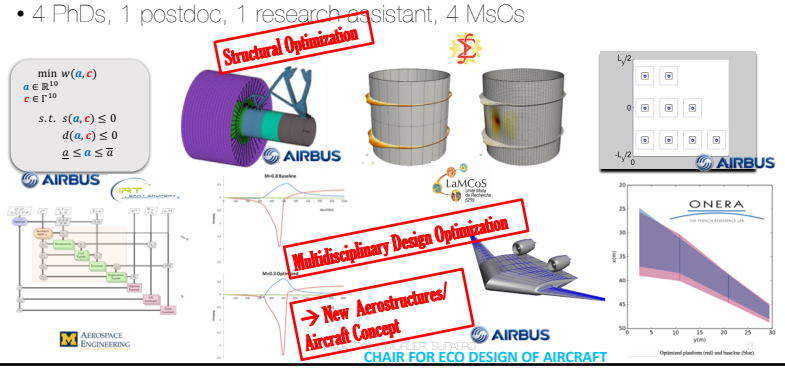
- Prof in Structural and Multidisciplinary Optimization
- Bat38 SUPAERO
- Research Lab (ICA)



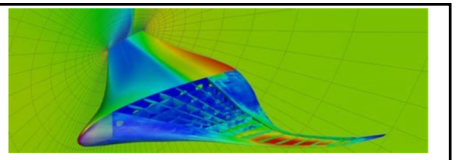


My Research Group

- 4 PhDs, 1 postdoc, 1 research assistant, 4 MsCs



Popularization



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

http://mdolab.engin.umich.edu

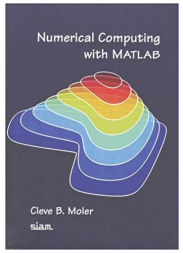
Optimization [MDO] for connecting people?

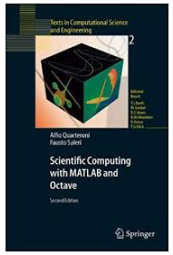
Publié le 14 février 2019 · Modifier l'article · Voir les stats

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

74 31 3 0

Start with scientifc computing





WHY?

Give you the basics

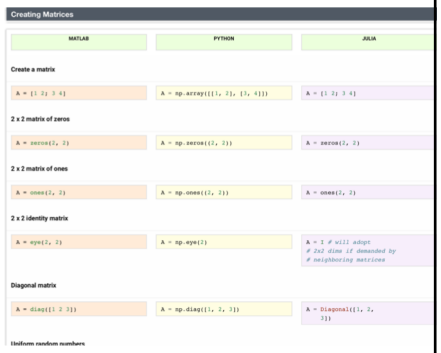
The way we will work together

- Basics of Scientific computing 4H with MATLAB/Python
- Basics of FEA 2H with MATLAB to review basics of CSM

https://cheatsheets.quantecon.org

Online notebooks available for System Identification problem on MATLAB with application to modal analysis

see github -→ Online standardization to Flexible aircraft course

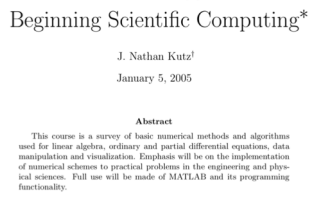


Aim of this lecture:

- 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
7. Numerical integration in 2D
7. Eigenvalues and Eigenvectors
8. 2D Laplace Equation
9. 1D Boundary Value Problem
10. 2D Laplace Equation using Jacobi and Gauss-Seidel
11. SVD low rank approximation of an image



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

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

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
- $A + B = [a_{ij}]_{MXN} + [b_{ij}]_{MXN} = [a_{ij} + b_{ij}]_{MXN}$
- $A - B = [a_{ij}]_{MXN} - [b_{ij}]_{MXN} = [a_{ij} - b_{ij}]_{MXN}$
- $cA = [ca_{ij}]_{MXN}$
- $pA + qB = [pa_{ij} + qb_{ij}]_{MXN}$

Matrices operation

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

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

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

properties

$A + B = B + A$
$0 + A = A + 0$
$A - A = 0$
$(A+B) + C = A + (B+C)$
$(p+q)A = pA + qA$
$p(A+B) = pA + pB$
$p(qA) = (pq)A$

$AB \neq BA$ ← very important

Matrices operation

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

- $\vec{x} = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} \qquad \vec{x}^T = (2 \ 3 \ 5)$
- $A = [a_{ij}]_{MXN} \qquad A^T = [a_{ji}]_{NXM}$

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

- 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 matrices

$A = [a_{ik}]_{mxn}$

$B = [b_{kj}]_{nxp}$

then

$AB = C = [c_{ij}]_{mxp}$ ⇒ columns of A must equal rows of B

Example

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

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

$= \begin{pmatrix} 5\cdot2+3\cdot3 & -2\cdot2+3\cdot8 & 2\cdot1+3\cdot-6 \\ -1\cdot5+3\cdot4 & -1\cdot-2+4\cdot8 & -1\cdot1+4\cdot-6 \end{pmatrix}$

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

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

Note: $AB \neq BA$ in general

if $AB = BA$ ⇒ then A and B commute

Choleski : LL'

$$A = LDL' \quad \text{et} \quad A = \tilde{L}\tilde{L}'$$
$$\tilde{L} = L\sqrt{D}$$

D must be positive!

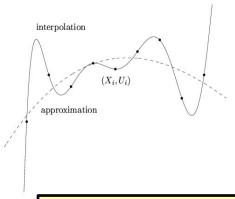
axiom :
if A is symetric positive definite (SPD)
it has an unique decomposition
A=LL'
where L est triangular (inferior) matrix with every diagonal components positives

Backslash

Matlab : x =A\b ;
if A triangular : x=trisup(A,b)
else if A SPD ; L=chol(A)
else : (* general case*)
[L,U,P]=lu(A); z=L\(P*b); x=U\z;

Interpolation

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 problem (LINEAR kernel)

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

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

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

Polynomial interpolation

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

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

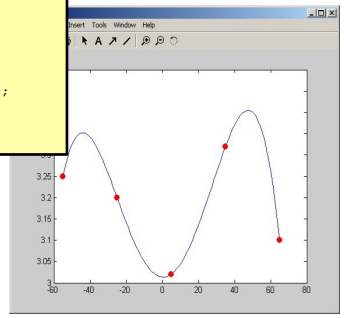
Vandermonde Matrix

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

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.

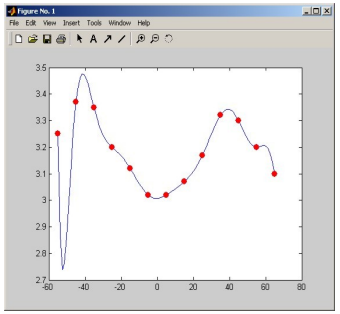
Example

```
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);
```



More points…

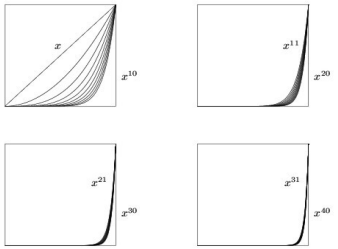
Latitude	
65	3.10
55	3.22
45	3.30
35	3.32
25	3.17
15	3.07
5	3.02
-5	3.02
-15	3.12
-25	3.20
-35	3.35
-45	3.37
-55	3.25



Polynomial matrix

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

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



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

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

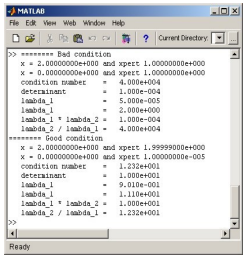
$$\begin{bmatrix} 1 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} a_\epsilon \\ b_\epsilon \end{bmatrix} = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$$

Perturbed solution: a=1.99999 et b=0.00001

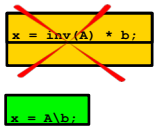
Well conditionned system

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



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

- 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

- 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

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

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 higher accuracy schemes by using more points: i.e. $f(x+2\Delta x)$, $f(x-2\Delta x)$, etc.

FD

Second derivative? $f''(t)$

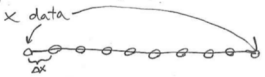
$$f(t+\Delta t) + f(t-\Delta t) = 2f(t) + \Delta t^2 \frac{d^2 f(t)}{dt^2} + \frac{\Delta t^4}{4!}\left(\frac{d^4 f(t)}{dt^4}\right) + \mathcal{O}(\Delta t^4).$$

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

(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

Eigen Analysis

Eigenvalues & Eigenvectors

'Eigen' = latent or characteristic

$Ax = \lambda x$ for special vectors x and special values λ.

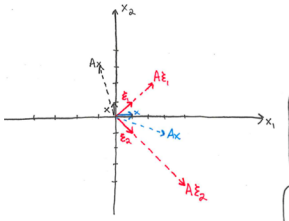
Eigenvalue eq^n for single eigen pair (x, λ).

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



Special $\underline{\xi}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow A\underline{\xi}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ $\lambda_1 = 2$

Special $\underline{\xi}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow A\underline{\xi}_2 = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$ $\lambda_2 = 4$

Exactly 2 eigenvalues λ_1, λ_2 and 2 eigenvectors $\underline{\xi}_1, \underline{\xi}_2$.

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 $\underline{0}$.

$\det(A - \lambda I) = 0$ polynomial equation whose roots are eigenvalues!

Characteristic Equation

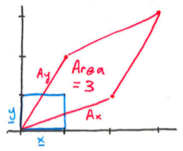
Remember 3×3 determinant...

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

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

$\Rightarrow$ eigenvalues are $\lambda_1 = 2$, $\lambda_2 = 4$.

Step 2 compute x given λ.

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

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

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

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

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

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

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

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

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

EDO

- Exercices:

7. Eigenvalues and Eigenvectors

EDO

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

(a) Taylor series

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

(c) Suspend Variables

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

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

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

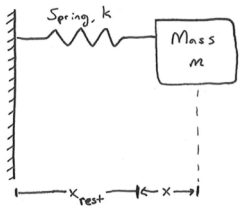
: Characteristic polynomial

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

(b) Plot in Matlab

Second-order systems

EDO



Newton's 2nd Law:

$F = ma$

$= m\ddot{x}$

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

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

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

41

EDO

$\ddot{x} = -x$

Method 1: Guess!

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

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

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

For general m, k:

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

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

Method 4: Suspend variables & solve as linear system

$\dot{x} = v$ (new variable)

$\dot{v} = -x$

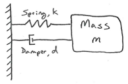
$\Rightarrow \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix}$

Much more on this later!

HA404- Prof J. MORLIER, SUPAERO 42

EDO

Example Damped Harmonic Oscillator



$F = ma$

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

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

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

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

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

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

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

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

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

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

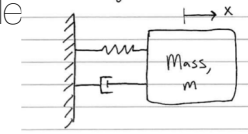
$\Rightarrow \boxed{\lambda = \frac{-\zeta \pm \sqrt{\zeta^2 - 4\omega^2}}{2}}$ 2 solutions (+/-) λ_1 and λ_2.

$\boxed{x(t) = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}}$ Need to initial conditions for C_1, C_2.

HA404- Prof J. MORLIER, SUPAERO 43

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 linear differential equation.

$\dot{x} = v$

$\dot{v} = -2\zeta\omega_0 v - \omega_0^2 x$ $\Rightarrow \frac{d}{dt}\begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{bmatrix}\begin{bmatrix} x \\ v \end{bmatrix}$

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

, SUPAERO 44

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 $(I + A\Delta t)$.

Time Domain Analysis	Frequency Domain Analysis
$\dot{y} + ay = u_s(t)$	$\dot{y} + ay = u_s(t)$
1 Set the right hand side to zero: $\dot{y} + ay = 0$	1 Take the Laplace transform: $[s+a]Y(s) - y(0) = \frac{1}{s}$
2 Assume exponential $y_c(t)$: $y(t) = Ae^{st} \Rightarrow [s+a]Ae^{st} = 0$	2 Solve for $Y(s)$: $Y(s) = \frac{1}{s(s+a)} + \frac{y(0)}{s+a}$
3 Solve the characteristic equation: $s + a = 0 \Rightarrow s = -a$	3 Expand using partial fractions: $\frac{1}{s(s+a)} = \frac{A}{s} + \frac{B}{s+a} = \frac{1/a}{s} - \frac{1/a}{s+a}$
4 Plug roots back into guess: $y_c(t) = Ae^{-at}$	4 Take the inverse Laplace transform: $y(t) = \frac{1}{a}u_s(t) - \frac{1}{a}e^{-at} + y(0)e^{-at}$
5 Assume $y_p(t)$ with same form as input: $y(t) = Bu_s(t)$	$= \frac{1}{a} + \left(y(0) - \frac{1}{a}\right)e^{-at}$ for $t > 0$
6 Solve for particular solution: $0 + aBu_s(t) = u_s(t) \Rightarrow y_p(t) = \frac{1}{a}u_s(t)$	
7 Write the solution as $y_c + y_p$: $y(t) = Ae^{-at} + \frac{1}{a}u_s(t)$	
8 Apply I.C.s: $y(t) = \frac{1}{a} + \left(y(0) - \frac{1}{a}\right)e^{-at}$ for $t > 0$	

Jacobi FD in 2D

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!)

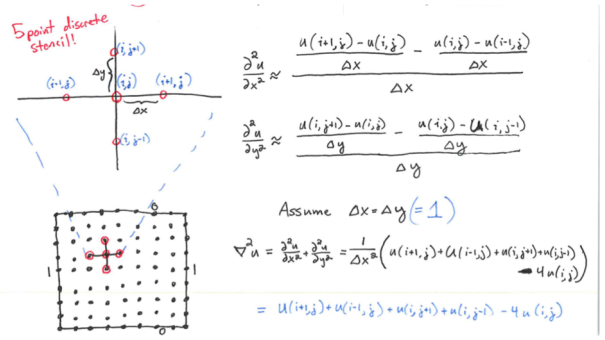
$$\frac{u(t+\Delta t) - u(t)}{\Delta t} = \alpha \nabla^2 u(t)$$

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

(A) $\nabla^2 u$ can be computed using 'del2' function

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

Jacobi and GS



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

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

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

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

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

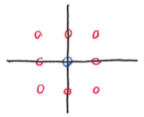
(A) 5-point stencil: (set $\nabla^2 u = 0$, solve for u_{ij})

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

i.e. average neighbors.

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

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



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 $\mathbf{X}^T$