Lecture 13: Other Concepts and Utilities in WOLFRAM MATHEMATICA.

A8B17CAS

Jozef Lukáč

Department of Radio Engineering
Czech Technical University in Prague
Czech Republic
lukacjo1@fel.cvut.cz

December 16
Winter semester 2024/25

Outline

1. `Export` to a file.
2. Plotting + `Options`.
3. Conditions
4. Procedural Programming.
5. Applying Functions Repeatedly
6. `Module`
7. Trace – Debugging in MATHEMATICA
8. Calculus
9. Array shaping
10. Array/Matrix Broadcasting (Python/MATLAB vs MATHEMATICA).
11. Logical Indexing and multiple assignment
12. `Manipulate`
13. Example of Code.

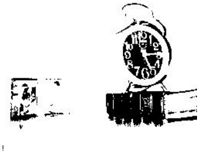
Export to a file.

- To save a part of an expression (intended to be used only in MATHEMATICA), you can use **Put**, resp. **>>**. To read the saved expression, use **Get**, resp. **<<**, *e.g.*

```
SetDirectory[NotebookDirectory[]];
expr = Table[RandomInteger[], 5, 2];
expr >> "outFile"; (*Put[expr,"outFile"]*)
<< "outFile"; (*Get["outFile"]*)
a1 = % (*set a1 to the expression from file*)
```

- To export an expression using a specific format, use **Export["dest.ext",expr]**, *e.g.*

```
SetDirectory[NotebookDirectory[] <> "data"];
im = Import["im03_desk.tiff"];
imData = ImageData[im]; (*2D matrix or Reals*)
thr[n_] := If[n > 0.34, 1, 0];(*threshold function*)
thrImData = Map[thr, imData, {2}];
Image[thrImData] (*create image from the thresholded data*)
Export["im03_desk_thr.png", %]
```

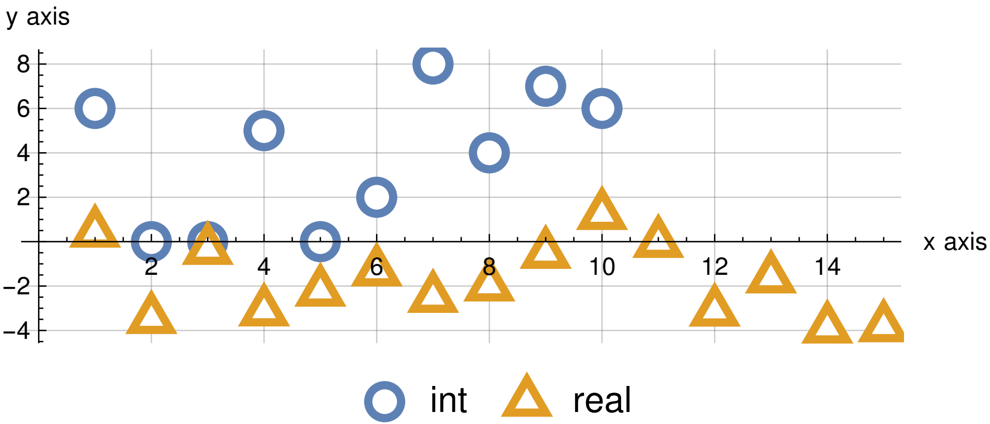


Plotting + Options.

- There are many plot commands: **Plot**, **Plot3D**, **CountourPlot**, **ListPlot**, **ListLinePlot**, **ParametricPlot**, **ParametricPlot3D**, **DensityPlot**, **DiscretePlot**, **ArrayPlot**, **MatrixPlot**, **Histogram**, **PolarPlot**, **RegionPlot**, **VectorPlot**, ...
- Some commands, especially plotting commands, usually have options. To get them, use `Options[`*symbolName*`]`, *e.g.* **Options[Plot]**
- Options are in the form of a rule. And have default values unless explicitly changed.
- Example of a plot:

```
ListPlot[{RandomInteger[{0, 8}, 10], RandomReal[{-5, 3}, 15]},
  PlotLegends -> {"int", "real"}, PlotMarkers -> {"OpenMarkers", 12},
  AspectRatio -> 1/3, AxesLabel -> {"x axis", "y axis"},
  GridLines -> Automatic]
```

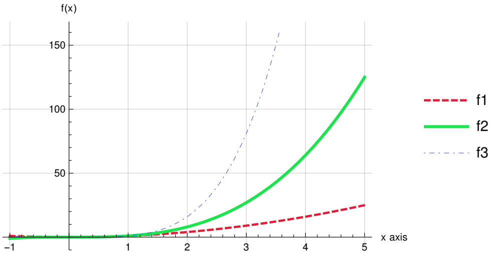
Plotting + Options.



Example of using ListPlot from the previous slide. (exported to pdf with Export)

Plotting + `Options`.

- Try to get similar figure of functions $f_1(x) = x^2$, $f_2(x) = x^3$, $f_3(x) = x^4$ in range of $x \in (-1, 5)$. Use **`Plot`** command.
- Usuful options are: **`PlotLegends`**, **`AxesLabel`**, **`GridLines`**, **`PlotStyle`**.
- **`PlotStyle`** substitutes for a list of line options, *e.g.* **`PlotStyle->{lineOptionList1, lineOptionList2, lineOptionList3}`**, where **`lineOptionList`** is a list and can contain (a) color specification (*e.g.* **`RGBColor[r,g,b]`**, or name of a color *e.g.* **`Red`**); (b) line thickness (*e.g.* **`Thick`**, **`Thin`**, **`Thickness[ratio]`**); (c) line style (*e.g.* **`Dotted`**, **`DotDashed`**, **`Full`**, **`Dashed`**)



Exercise

Conditions

- `If[condition,t,f]` – evaluates `t` if `condition` is true otherwise evaluates `f`.
- `Whitch[test1,val1,test2,val2,...]` – returns first $\mathtt{val}_i$ for which $\mathtt{test}_i$ evaluates true. Useful *e.g.* for classification.
- `Switch[expr,form1,val1,form2,val2,...]` – evaluates expression and checks a match with each $\mathtt{form}_i$, and returning $\mathtt{val}_i$ of the first matched form.
- Composition of boolean expressions, see `And` (`&&`) and `Or` (`||`).
- Examples:

```
classif[x_] := Which[x<1,range1,1<=x<3,range2,3<=x<5,range3,True,otherRange];
RandomInteger[{-3, 9}, 10]
classif /@ %
expr = x^(-1 + 4);
Switch[expr,_^2,"square power",_^_,"power",_Plus, "sum",_,"dont know"]
```

Procedural Programming.

- Mathematica has also commands for procedural programming, *e.g.* **For**, **While**, **Do**, **Break**, **Continue**, **Return**, (**Print**). They are not used much – usually, there is a simple, functional way to do the same, *e.g.*

```
v=RandomReal[{0,1},10]
For[i=1,i<Length[v],i++,If[v[[i]]>0.6,Print["OK"],Print["notOK"]]]
```

vs.

```
Scan[If[#>0.6,Print["OK"],Print["notOK"]]&,v]
```

Applying Functions Repeatedly

Besides **Map** and **Apply**, other commands are used to apply functions/symbols repeatedly/multiple times.

- **Nest[f,expr,n]** – **n**-times applies **f** to **expr**.
- **NestList[f,expr,n]** – gives all intermediate results of **Nest[f,ex,n]** in a list.
- **NestWhile[f,expr,test]** – applies **f** to **expr** while **test** yields true.
- **NestWhileList[f,expr,test]** – same as above with all intermediate results in list.
- **FoldList[f,x,{a,b,...}]** – gives `{x,f[x,a],f[f[x,a],b],...}`.
- **Fold[f,x,list]** – last element of **FoldList[f,x,list]**.
- **FixedPoint[f,expr]** – similar to **Nest[f,expr,n]**, but nesting stops when the result no longer changes. *E.g.*

```
Nest[f,x,3]                                      f[f[f[x]]]
NestList[f,x,3]                                  {x,f[x],f[f[x]],f[f[f[x]]]}
NestWhileList[2#+1&,6,Mod[#,5]!=0&]              {6, 13, 27, 55}
mat={{1,-0.75},{0.4,0.2}};NestList[mat.#&,{1,0},3]   {{1,0},{1.,0.4},{0.7,0.48},{0.34,0.376}}
FoldList[#1^2-15#2&,2,{2,-3,1}]                  {2, -26, 721, 519826}
```

Applying Functions Repeatedly. Examples.

- Iteratively find minimum from each column of a matrix, get also intermediate results

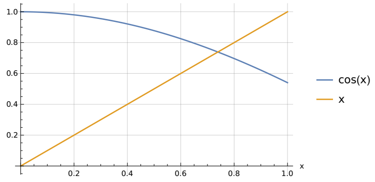
```
mtx=RandomInteger[{-4,5},{4,5}];mtx//MatrixForm
min[l1_List,l2_List]:=Inner[Min,l1,l2,List];
FoldList[min,mtx]//MatrixForm
Last[%]
```

  - ...other way with `Map` (`/@`):

```
Min/@Transpose[mtx]
```

- Fixed point[1]. Solution to $\cos(x) = x$:

```
Plot[{Cos[x],x},{x,0,1}]
NestList[Cos[#]&,0.,20]
FixedPoint[Cos[#]&,0.]
```



Plot of the functions $\cos(x)$ and x.

- gradient search of a local maximum:

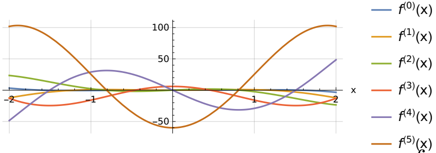
```
Cos[x] + y^2 - 1/z (*function of multiple variables*)
Grad[%, {x, y, z}] (*gradient*)
NestList[# + 0.05 %/.MapThread[Rule, {{x, y, z}, #}] &, {-0.1, 0.1, 0.2}, 5]
(*gradient search -- return 3D points*)
%%% /. MapThread[Rule,{{x,y,z}, #}] & /@ % (*evalutate the function in the
points*)
```

[1] https://en.wikipedia.org/wiki/Fixed_point_(mathematics)

Applying Functions Repeatedly. Examples.

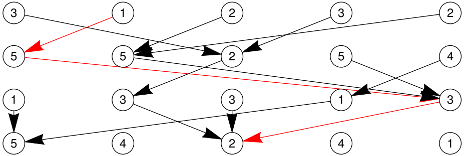
Exercise

- Compute first 5 derivatives of the function $x^3\cos(x)$ and plot the result. Use **`NestList`**, then plot them using **`Plot`** command.



- Assume a matrix of indices **`mtx=RandomInteger[{1,n},{m,n}]`** for **`m=4,n=5`**.

$$\mathtt{mtx}=\begin{bmatrix} 3 & 1 & 2 & 3 & 2 \\ 5 & 5 & 2 & 5 & 4 \\ 1 & 3 & 3 & 1 & 3 \\ 5 & 4 & 2 & 4 & 1 \end{bmatrix},$$



Each row corresponds to indices in the next row. Start at row 1, at position 2: $\mathtt{mtx}_{1,2} = 1$, go to next row at the position 1 – there is number 5, and so on. What is the sequence of indices for a randomly generated matrix? (in this case it would be **`{2, 1, 5, 3, 2}`**) Start at the first row at position 2. You can use **`FoldList`**.

Applying Functions Repeatedly. Examples (2).

- Implement Heron's method[2] of approximate iterative computation of the square root of s: $\sqrt{s}$. $x_{n+1} = \frac{1}{2}(x_n + \frac{S}{x_n})$. Start with $x_0 = 1$. A) use **NestList** to compute 10 first x_n, B) then use **FixedPoint** to return just the result.

```
s = 5.0; (*number to take square root of*)
NestList ...
FixedPoint ...
Sqrt[s] - % (*compare the result with the true value*)
```

- Implement the discrete-time system: $y[n] = x[n] + \alpha y[n-1]$. For the given vector/sequence $x[n]$, $x = \{x[0], x[1], \ldots, x[N_{\max} - 1]\}$ (and a specified value of α), compute $y[n]$. Use **FoldList**. Start with inital y/output-value 0 ($y[-1]$ assume 0).

```
alpha = 0.95; nOnes = 100; nZeros = 30;
x = Table[0.0, 20]~Join~Table[1.0, nOnes]~Join~Table[0.0, nZeros];
(*x=Table[0,20]~Join~Exp[-2.5/nOnes Range[nOnes]]~Join~Table[0,nZeros];*)
FoldList ...
ListPlot[{x, %}, PlotRange -> All, Filling -> Axis, ImageSize -> Large]
```

Exercise

[2] https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Heron's_method

Module

- More complicated functions can be specified using `Module[{x,y,...}, expr]`, variables/symbols `x,y,...` are local. *E.g.*

```
g[vec_List,n_]:=Module[{idc},
  idc=Position[vec,n];
  Print[idc];
  Join[vec,Flatten@idc]];
g[RandomInteger[{0,6},10],2]
```

Trace – Debugging in MATHEMATICA.

- When writing a code, we iteratively adjust it to do what we originally intended. The process of looking for and correcting errors is called *debugging*[3].
- Steps in the evaluation of an expression can be revealed by `Trace[expr]`, *e.g.*
  `Trace[2^3 + 4^2 + 1]` $\longrightarrow$ `{{2^3,8},{4^2,16},8+16+1,25}`

[3] Debugging https://en.wikipedia.org/wiki/Debugging

Calculus

- MATHEMATICA allows users to solve problems in calculus, linear algebra, general algebra, and others, all symbolically and/or numerically.
- Usefull commands are: `N` (numeric evaluation), `D` (derivative), `Integrate`, `NIntegrate` (numeric integration), `FindRoot`, `Roots`, `Solve`, `Limit`.
- The Final result can sometimes be simplified using additional rules. See `Simplify`, `FullSimplify`, ...
- In solutions we can help MATHEMATICA by assumptions (use `Assuming`), *e.g.*

```
Integrate[x Exp[x^n], x]
Assuming[Element[n, Integers] && x > 0, % // Simplify]
```

Calculus

Exercise

Calculate following:

$$\int_0^{2\pi} \mathrm{e}^{x\cos(\alpha)}\,\mathrm{d}\alpha, \tag{1}$$

```
...
```

```
2 Pi BesselI[0, x]
```

$$\int_0^1 \cos(\sin(x))\,\mathrm{d}x, \quad \text{numerically}, \tag{2}$$

```
...
```

```
0.86874
```

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n, \tag{3}$$

```
Limit[(1 + 1/n)^n, n -> Infinity]
```

```
E
```

$$\cos\left(\tan\left(\mathrm{e}^x\right)\right)'. \tag{4}$$

```
D[Cos[Tan[Exp[x]]], x]
```

```
-E^x Sec[E^x]^2 Sin[Tan[E^x]]
```

Array shaping

- When working with multidimensional arrays/expressions, we often need to rearrange shapes. In MATLAB, we can use `reshape(mat,dims)`. In MATHEMATICA, we have `ArrayReshape[mat,dims]`.
- Additionally, very useful are `Flatten` and `Partition`, *e.g.*

```
Table[{i, j, i - j}, {i, 3}, {j, 4}]
Flatten[%, 1]
RandomInteger[{1, 5}, 10]
Partition[%, 3] // MatrixForm
Partition[%%, 3, 1] // MatrixForm
```

Array shaping, examples

- Show digits of π in 7 rows. – From a list of digits, create a matrix of 7 rows. Digits should be written in columns. Use **Partition**

```
Pi
First@RealDigits@N[%, 100](*list of digits*)
...
...
Grid[%, Frame -> All] (*show the matrix in a grid*)
```

3	6	9	2	2	8	9	1	7	2	4	0	8	3
1	5	3	6	7	4	3	0	4	3	0	8	0	4
4	3	2	4	9	1	9	5	9	0	6	9	3	2
1	5	3	3	5	9	9	8	4	7	2	9	4	1
5	8	8	3	0	7	3	2	4	8	8	8	8	1
9	9	4	8	2	1	7	0	5	1	6	6	2	7
2	7	6	3	8	6	5	9	9	6	2	2	5	0

- Do a block-wise permutation of a list of numbers. From a list of numbers, create a list of 2x3 matrices, transpose them and read by rows to a new list. Use **Partition** and **Flatten**.

```
Range[24] (*list of numbers*)
... (*transform list of numbers*)
... (*to list of matrices*)
MatrixForm /@ %
... (*tranpose matrices*)
MatrixForm /@ %
... (*read rows*)
```

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24}

$$\left\{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}, \begin{pmatrix} 13 & 14 & 15 \\ 16 & 17 & 18 \end{pmatrix}, \begin{pmatrix} 19 & 20 & 21 \\ 22 & 23 & 24 \end{pmatrix}\right\}$$

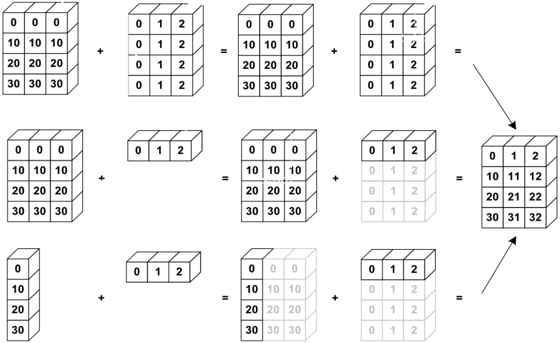
$$\left\{\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}, \begin{pmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{pmatrix}, \begin{pmatrix} 13 & 16 \\ 14 & 17 \\ 15 & 18 \end{pmatrix}, \begin{pmatrix} 19 & 22 \\ 20 & 23 \\ 21 & 24 \end{pmatrix}\right\}$$

{1, 4, 2, 5, 3, 6, 7, 10, 8, 11, 9, 12,
13, 16, 14, 17, 15, 18, 19, 22, 20, 23, 21, 24}

Exercise

Array/Matrix Broadcasting (Python/MATLAB vs MATHEMATICA)

- Schematic illustration of broadcasting arrays in Python (numpy)/MATLAB[4]:



[4]source: https://mathematica.stackexchange.com/questions/99171/how-to-implement-the-general-array-broadcasting-method-from-numpy

Array/Matrix Broadcasting (Python/MATLAB vs MATHEMATICA)

- Define `mat=Table[10 i,{i,0,3},3]; v1={0,1,2}; v2={0,1,2,3};`. In MATLAB, we define `mat=repmat(10*(0:3)',1,3), v1=0:2, v2=0:3`

MATLAB	MATHEMATICA
`v1 + mat`	`(# + v1) & /@ mat`
`v1 .* mat`	`(# v1) & /@ mat`
`v2'+ mat`	`v2 + mat`
`v2'.* mat`	`v2 * mat` (note the difference with `v2 . mat`)
`v1 + v2'`	`(# + v1) & /@ v2` or `Outer[Plus, v2, v1]`
`v1 .* v2'`	`(# v1) & /@ v2` or `Outer[Times, v2, v1]`

- or you can explicitly replicate array with help of `ConstantArray` and `Join`.

Logical Indexing and multiple assignment.

- Mathematica doesn't have logical indexing as Matlab.
- …but similar behaviour can be achieved by **ReplacePart** and **Position**[5]. Sometimes we can use a rule, *e.g.*

```
mat = RandomReal[{0, 1}, {5, 3}];
mat2 = ReplacePart[mat, Position[mat, pos_ /; pos < 0.7] -> -1];
mat3 = mat /. {x_ /; x < 0.7 -> -1};
MatrixForm /@ {mat, mat2, mat3}
```

  in Matlab it would be

```
mat=rand(5,3); mat2 = mat; mat2(mat<0.7)=-1;
mat,mat2
```

- Multiple assignment works, but the parts have to be of equal size, *e.g.*

```
mat = RandomInteger[5, {5, 3}]; mat // MatrixForm
mat[[1]] = Table[-1, 3];
mat[[2 ;; 5, 2]] = Table[10, 4];
{a1, a2} = {10, 11};
{a1, a2, mat // MatrixForm}
```

[5] https://mathematica.stackexchange.com/a/2823/74293

Manipulate

- Interactive manipulation with general expressions, *e.g.*

```
Manipulate[Plot[Sin[k x], {x, -10, 10}], {k, 0.1, 8.3}]
Manipulate[(x + y)^n // Expand, {{n, 2}, Range[10]}]
```

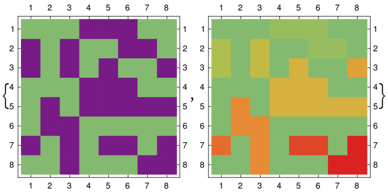
- *E.g.* electric field near two static charges:

```
1/Sqrt[(x + 2)^2 + y^2] - al 1/Sqrt[(x - 2)^2 + al y^2]
Grad[%, {x, y}]
Manipulate[VectorPlot[%, {x, -5, 5}, {y, -5, 5}, VectorScale -> {Tiny, 1, 1 &},
  VectorColorFunction -> "SunsetColors", VectorColorFunctionScaling -> False,
  FrameLabel -> {"x", "y"}],
  {{al, 1, "alpha"}, 0, 4, 0.1}]
```

Example of Code.

Create a square matrix of 0s and −1s – the playground. −1s are ground, 0s are water. An island is a set of places that meet by at least one side (of square). Find islands and denote each by a different number. Then, plot them.

```
n = 8; plgrnd = RandomInteger[{0, -1}, {n, n}];
listMeetQ[lst1:{{_, _} ..}, lst2:{{_,_} ..}] :=
  AnyTrue[Flatten[Outer[List,lst1,lst2,1],1],
   (Plus@@Abs[Subtract@@ #]) ==1 &];
rule={head___List,x:{{_, _}..},fill___List,y:{{_, _}..},
  tail___List} /; listMeetQ[x, y] :> {head, Join[x, y], fill, tail};
pos = ({#} & /@ Position[plgrnd, -1]) //. rule;
plgrnd2 = ReplacePart[plgrnd, MapThread[Rule, {pos, Range[Length[pos]]}]];
MatrixPlot[#, ColorFunction -> "Rainbow"] & /@ {plgrnd, plgrnd2}
```



Questions?

A8B17CAS
lukacjo1@fel.cvut.cz

December 16
Winter semester 2024/25

Voluntary homework

- Assume you have a matrix (a bitmap picture)
  `n = 5; mat = RandomInteger[{-2, 4}, {n, n}];` Read the documentation of the command `Partition` and divide the matrix to submatrices of the size 3 x 3 with offset 1 in both directions. Compute the sums of the created submatrices.

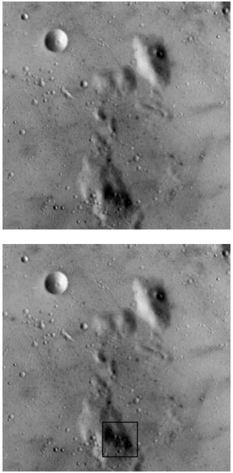
$$\begin{pmatrix} -2 & 3 & -1 & 2 & 4 \\ 4 & 3 & 2 & 0 & 4 \\ 0 & 2 & 3 & 4 & -1 \\ 0 & -2 & -1 & 2 & 1 \\ -1 & 2 & 0 & 4 & 4 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} -2 & 3 & -1 \\ 4 & 3 & 2 \\ 0 & 2 & 3 \end{pmatrix} & \begin{pmatrix} 3 & -1 & 2 \\ 3 & 2 & 0 \\ 2 & 3 & 4 \end{pmatrix} & \begin{pmatrix} -1 & 2 & 4 \\ 2 & 0 & 4 \\ 3 & 4 & -1 \end{pmatrix} \\ \begin{pmatrix} 4 & 3 & 2 \\ 0 & 2 & 3 \\ 0 & -2 & -1 \end{pmatrix} & \begin{pmatrix} 3 & 2 & 0 \\ 2 & 3 & 4 \\ -2 & -1 & 2 \end{pmatrix} & \begin{pmatrix} 2 & 0 & 4 \\ 3 & 4 & -1 \\ -1 & 2 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 2 & 3 \\ 0 & -2 & -1 \\ -1 & 2 & 0 \end{pmatrix} & \begin{pmatrix} 2 & 3 & 4 \\ -2 & -1 & 2 \\ 2 & 0 & 4 \end{pmatrix} & \begin{pmatrix} 3 & 4 & -1 \\ -1 & 2 & 1 \\ 0 & 4 & 4 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 14 & 18 & 17 \\ 11 & 13 & 14 \\ 3 & 14 & 16 \end{pmatrix}$$

Example of the matrix, partitions/submatrices, and sums of submatrices.

Voluntary homework

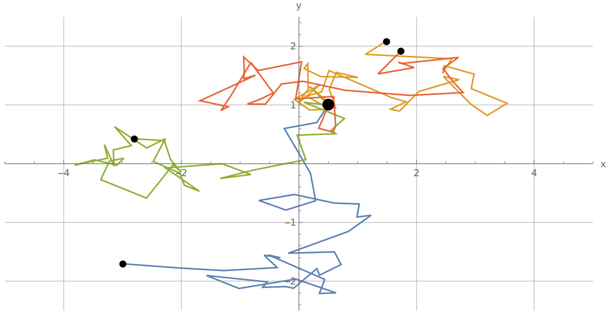
Your code can be used to detect a part of an image. *E.g.* To detect a darkest p x p block.

```
p = 40; (*partition size*)
SetDirectory[NotebookDirectory[] <> "data"];
mat = Import["im04_surface.tiff", "Data"]/255.;
n = Length@mat;
(*your code *)
(*goes here*)
ijFlat = First@Ordering[Flatten@%, 1]; (*find index of minimum*)
ij = With[{m=n-p+1}, {Quotient[#-1,m]+1, Mod[#-1,m]+1}]& @ ijFlat;
idc = Module[{i, j, iRng, jRng},
   i = First@%;
   j = Last@%;
   jRng = Range[j, j + p - 1];
   iRng = Range[i + 1, i + p - 2];
   Join[{i, #} & /@ jRng, {i + p - 1, #} & /@ jRng , {#, j} & /@
     iRng, {#, j + p - 1} & /@ iRng]
   ];
mat2 = ReplacePart[mat, idc -> 0];
GraphicsGrid[{{Image[mat], Image[mat2]}}, ImageSize -> Medium]
```



Voluntary homework (2) – Random walk

Generate a random walk[7] in 2 dimensions ($\mathbb{R}^2$). Assume the step in the x direction from $\mathcal{N}(0, 0.5)$; the step in the y direction from $\mathcal{N}(0, 0.25)$[8] Start from the point $[0.5, 1]$ and generate a list of 2-D points (2-element lists) – the path. Try to use `NestList` even though using `Accumulate` could be faster. To generate more paths, you can use `Table[(*code for one path*),{nPaths}]`. Plot the path(s) using `ListLinePlot`.



[7] https://en.wikipedia.org/wiki/Random_walk

[8] $\mathcal{N}(\mu, \sigma)$ denotes a random number from normal distribution with mean μ and standard deviation σ.

Voluntary homework (3) – CORDIC algorithm

- In CORDIC algorithm for computation of sine and cosine[9] there is a sequence $K(n)$ that can be defined iteratively as $K(n+1) = K(n)\frac{1}{\sqrt{1+2^{-2n}}}, \quad K(0) = 1$. Numerically compute the limit of the sequence (**Limit** will not help). Define a function that takes a two-element vector/list $\{K(n), n\}$ and returns also two-element vector $\{K(n+1), n+1\}$. Use the function with **Nest** or **NestList**. Then evaluate the result using **N** to 40 digits.
- Implement CORDIC algorithm for computation of sine and cosine of an angle α. Define a function for one-iteration $f(\{x_n, y_n, \theta_n, n, \alpha\}, \gamma)$ that returns $\{x_n - \sigma y_n 2^{-n}, \sigma 2^{-n} x_n + y_n, \theta_n + \sigma_n \gamma, n+1, \alpha\}$, where $\sigma = \mathrm{sign}(\alpha - \theta_n)$. Values of θ_n are precomputed in a list. $x_0 = 1.0, y_0 = 0.0, \gamma_0 = 0.0$, start from $n = 0$. Use **Fold** or **FoldList**; **With** may help. Compute the error w.r.t. the correct values. Result needs to be scaled by K (**kN**).

```
nIters = 20; alpha = 0.8; (*angle to compute sine and cosine of*)
gammaList = ArcTan[2^#, 1] &@Range[0, nIters - 1] // N;(*{x,y,theta,n,alpha}*)
f[{x_, y_, theta_, n_, alpha_}, gamma_] ...
(*f[{1.0,0.0,0.0,0,alpha},gammaList//First] -- run for a single gamma value*)
kN = 0.6072529350088812561694467525049282 6;
... (*Fold or FoldList with f and gammaList*)
(kN*(*result: {x_n, y_n}*)) - ({Cos@#, Sin@#} &@alpha) (*error*)
```

[9] https://en.wikipedia.org/wiki/CORDIC#Modes_of_operation

Voluntary homework (4) – Discrete-time system (DtS)

- Compute output sequence/vector – a response of a DtS – to a given input vector. The DtS is given by the input-output relation $y[n] = b_0 x[n] + b_1 x[n-1] + a_1 \sin\left(\frac{2\pi n}{N_{\max}}\right) y[n-1] + a_2 y[n-2]$. Coefficients are $\boldsymbol{a} = \{a_1, a_2\} = \{0.5, -0,2\}$, $\boldsymbol{b} = \{b_1, b_2\} = \{1, -0,4\}$, $N_{\max} = 35$. Define a function $f[\{y_{n-1}, y_{n-2}\}, \{x_n, x_{n-1}, n\}]$ that returns the vector $\{y_n, y_{n-1}\}$. Assume a list of x-values; partition it to pairs, with offset one. Add a third column – a range of numbers $\{\{0\}, \{1\}, ...\}$ (use **Join**) and use together with f in **FoldList**. You will get a list of pairs $\{y_i, y_{i-1}\}$; extract the first column. Compute such an ouput vector $\boldsymbol{y}_1$ for input $\boldsymbol{x}_1$, and $\boldsymbol{y_2}$ for input $\boldsymbol{x}_2$.

```
alpha = -2.5; nOnes = 100; nZeros = 30;
xx1 = Table[0, 20]~Join~Table[1, nOnes]~Join~Table[0, nZeros];
xx2 = Table[0, 20]~Join~Exp[alpha/nOnes Range[nOnes]]~Join~Table[0, nZeros];
nMax = 35; aVec = {0.5, -0.2}; bVec = {1, -0.4}; (*{a1,a2}, {b1,b2}*)
f[{y1_, y2_}, {x0_, x1_, n_}] ...
yy1 ...
yy2 ...
ListLinePlot[{xx1, xx2, yy1, yy2}, PlotRange -> All, ImageSize -> Full,
 GridLines -> Automatic, AxesLabel -> {"x[n]", "y[n]"},
 PlotStyle -> {Full, Dashed, Dotted, DotDashed}, AspectRatio -> 0.4,
 PlotLegends -> Placed[{"x1[n]", "x2[n]", "y1[n]", "y2[n]"}, {Right, Center}]]
```