BMATRIX_EXPLAINER

by Matias I. Bofarull Oddo

Department of Computer Science
The University of British Columbia
<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Multi-paradigm: structured, imperative (procedural, object-oriented), generic, array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designed by</td>
<td>John Backus</td>
</tr>
<tr>
<td>Developer</td>
<td>John Backus and IBM</td>
</tr>
<tr>
<td>First appeared</td>
<td>1957; 65 years ago</td>
</tr>
<tr>
<td>Stable release</td>
<td>Fortran 2018 (ISO/IEC 1539-1:2018) / 28 November 2018; 3 years ago</td>
</tr>
<tr>
<td>Typing discipline</td>
<td>Strong, static, manifest</td>
</tr>
<tr>
<td>Filename extensions</td>
<td>f90, .f, .for, .f90</td>
</tr>
<tr>
<td>Website</td>
<td>fortran-lang.org</td>
</tr>
</tbody>
</table>

### C++

- **Logo endorsement by the C++ standards committee:**
- **Paradigm:** Multi-paradigm: procedural, imperative, functional, object-oriented, generic, modular
- **Family:** C
- **Designed by:** Bjane Stroustrup
- **Developer:** ISO/IEC JTC 1 (Joint Technical Committee 1) / SC 22 (Subcommittee 22) / WG 21 (Working Group 21)
- **First appeared:** 1983; 37 years ago
- **Stable release:** C++20 (ISO/IEC 14882-2020) / 15 December 2020; 21 months ago
- **Preview release:** C++17 / 11 March 2022; 6 months ago
- **Typing discipline:** Static, nominative, partially inferred
- **OS:** Cross-platform
- **Filename extensions:** .c, .cc, .cpp, .cxx, .h, .hh, .hpp, .hxx, .hpp, .hh+
- **Website:** isocpp.org

### MATLAB (programming language)

- **Paradigm:** multi-paradigm: functional, imperative, procedural, object-oriented, array
- **Designed by:** Cleve Moler
- **Developer:** MathWorks
- **First appeared:** late 1970s
- **Stable release:** R2022b / 15 September 2022; 4 days ago
- **Typing discipline:** dynamic, weak
- **File name extensions:** .m, .p, .mp, .max, .mat, .fig, .mex, .matlab, .m, .mlapp, .mldl, .mlddp, .matlapp, .mil, .mip, .mldn, .mipdata, .mipx, .mipxref, .mipxref
- **Website:** mathworks.com

### Python

- **Paradigm:** multi-paradigm: object-oriented, procedural (imperative), functional, structured, reflective
- **Designed by:** Guido van Rossum
- **Developer:** Python Software Foundation
- **First appeared:** 1991; 31 years ago
- **Stable release:** 3.10 / 7 September 2022; 7 days ago
- **Preview release:** 3.11.0rc0 / 12 September 2022; 2 days ago
- **Typing discipline:** duck, dynamic, strong typing
- **Framework:** Python, PyPi, CPython, Pypy, PyPy, Jython, IronPython, Jython
- **OS:** Windows, macOS, Linux, UNIX, Android and more
- **License:** Python Software Foundation License
- **Website:** python.org

### MATLAB (software)

- **L-shaped membrane logo**
- **Developer(s):** MathWorks
- **Initial release:** 1984; 36 years ago
- **Stable release:** R2022b / 15 September 2022; 4 days ago
- **Written in:** C/C++, MATLAB
- **Operating system:** Windows, macOS, and Linux
- **Platform:** IA-32, x64-64
- **Type:** Numerical computing
- **License:** Proprietary commercial software
- **Website:** mathworks.com

### Julia

- **Paradigm:** multi-paradigm: multiple dispatch (primary paradigm), procedural, functional, meta, multilanguage
- **Designed by:** Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah, and others
- **Developer:** Julia lang foundation
- **First appeared:** 2012; 10 years ago
- **Stable release:** 1.8.7 / 9 September 2020; 2 days ago and 1.7.3 / 15 July 2020; 2 months ago
- **Preview release:** Working on 1.8.8 and 1.9.0-DEV with daily updates
- **Typing discipline:** Dynamic, strong or value
- **Implementation language:** LLVM
- **Platform:** Tier 1: x86-64, IA-32, CUDA 10.0+ and cuDNN (for Linux and Windows), Tier 2: 64-bit ARM (e.g. Apple M1 Macs), while they also have tier 1 support using Rosetta2 32-bit Windows (as of 2020) Tier 3: 32-bit Arm, PowerPC, and ROCm (OpenGL) GPUs. Also supports most CUDA GPUs and Google's TPUs and has web browser support via JavaScript and WebAssembly and can work in Android. For more details see supported platforms.
- **OS:** Linux, macOS, Windows, and FreeBSD
- **License:** MIT license (copy at will) and can be used in Android. For more details see supported languages.
- **Website:** julia-lang.org

### Influences

- **Fortran:**
  - Speedcoding
  - ALGOL 58, BASIC, C, Chapel, CMS-2, DOPE, Fortress, PL/I, PACT I, MUMPS, IDL, Ratfor

- **C++**
  - Ada, ALGOL 68, BCP, C, CLU, ML, Mesa, Modula-2, Simula, Smalltalk
  - Ada 95, C#, C++, Chapel, Clojure, D, Java, JS, Swift, Lua, Nim, Objective-C++, Perl, PHP, Python, Rust, Seed7

- **MATLAB (software)**
  - Ada, ALGOL 68, BCP, C, CLU, ML, Mesa, Modula-2, Simula, Smalltalk
  - Ada 95, C#, C++, Chapel, Clojure, D, Java, JS, Swift, Lua, Nim, Objective-C++, Perl, PHP, Python, Rust, Seed7

- **Python**
  - Ada, ALGOL 68, BCP, C, CLU, ML, Mesa, Modula-2, Simula, Smalltalk
  - Ada 95, C#, C++, Chapel, Clojure, D, Java, JS, Swift, Lua, Nim, Objective-C++, Perl, PHP, Python, Rust, Seed7

- **Julia**
  - Ada, ALGOL 68, BCP, C, CLU, ML, Mesa, Modula-2, Simula, Smalltalk
  - Ada 95, C#, C++, Chapel, Clojure, D, Java, JS, Swift, Lua, Nim, Objective-C++, Perl, PHP, Python, Rust, Seed7
Long-story short, I made a Depth-First Search recursive scraper for Wikimedia API to extract knowledge networks hidden in semantically rich infobox fields.

My goal was to interlink these networks to fill information gaps, and then create a human-in-the-loop vis tool for Wikipedia editors.

As you can guess, it didn't go as planned . . .
Oh no, automated myself out of a job
Fortran

From Wikipedia, the free encyclopedia

Fortran (/ˈfɔːrtən/; formerly FORTRAN) is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing. Fortran was originally developed by IBM[1] in the 1950s for scientific and engineering applications, and subsequently came to dominate scientific computing. It has been in use for over six decades in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing[2] and is used for programs that benchmark and rank the world's fastest supercomputers.[3][4][5]

Fortran has had numerous versions, each of which has added extensions while largely retaining compatibility with preceding versions. Successive versions have added support for structured programming and processing of character-based data (FORTRAN 77), array programming, modular programming and generic programming (Fortran 90), High Performance Fortran (Fortran 95), object-oriented programming (Fortran 2003), concurrent programming (Fortran 2008), and native parallel computing capabilities (Coarray Fortran 2008/2018).

Fortran's design was the basis for many other programming languages. Among the better-known is BASIC, which is based on FORTRAN II with a number of syntax cleanups, notably better logical structures,[6] and other changes to work more easily in an interactive environment.[7]

Since August 2021 Fortran has ranked among the top 15 languages in the TIOBE index, a measure of the popularity of programming languages.[8]

### Contents

1 Naming
2 Origins
   2.1 FORTRAN
      2.1.1 Fixed layout and punched cards
3 Evolution
   3.1 FORTRAN II
      3.1.1 Simple FORTRAN II program
   3.2 FORTRAN III
   3.3 IBM 1401 FORTRAN
   3.4 FORTRAN IV
   3.5 FORTRAN 66
   3.6 FORTRAN 77
   3.7 Transition to ANSI Standard Fortran
   3.8 Fortran 90
      3.8.1 Obsolescence and deprecations
   3.8.2 "Hello, World!" example
   3.9 Fortran 95
      3.9.1 Conditional compilation and varying length strings
4 Modern Fortran
   4.1 Fortran 2003
   4.2 Fortran 2008
   4.3 Fortran 2018
5 Language features
6 Science and engineering
7 Portability
8 Obsolete variants
   8.1 Fortran-based languages
9 Code examples
Fortran

From Wikipedia, the free encyclopedia

Fortran (/ˈfɔːrtrən/; formerly FORTRAN) is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing.

Fortran was originally developed by IBM[2] in the 1950s for scientific and engineering applications, and subsequently came to dominate scientific computing. It has been in use for over six decades in computationally intensive areas such as numerical weather prediction, finite element analysis, computational fluid dynamics, geophysics, computational physics, crystallography and computational chemistry. It is a popular language for high-performance computing[3] and is used for programs that benchmark and rank the world's fastest supercomputers.[4][5]

Fortran has had numerous versions, each of which has added extensions while largely retaining compatibility with preceding versions. Successive versions have added support for structured programming and processing of character-based data (FORTRAN 77), array programming, modular programming and generic programming (Fortran 90), High Performance Fortran (Fortran 95), object-oriented programming (Fortran 2003), concurrent programming (Fortran 2008), and native parallel computing capabilities (Coarray Fortran 2008/2018).

Fortran's design was the basis for many other programming languages. Among the better-known is BASIC, which is based on FORTRAN II with a number of syntax cleanups, notably better logical structures,[6] and other changes to work more easily in an interactive environment.[7]

Since August 2021 Fortran has ranked among the top 15 languages in the TIOBE index, a measure of the popularity of programming languages.[8]
LESSON
Software that gap-fills Wikipedia is **NOT** an InfoVis project.
What are other ways to understand and better work with information dense networks?
Fig. 1 Example networks and their portraits. The random network is an Erdős-Rényi graph while the real network is the NCAA Division-I football network (Park and Newman 2005). Colors denote the entries of the portrait matrix $B$ (Eq. (2); white indicates $B_{\ell,k} = 0$)

Fig. 1 Example networks and their portraits. The random network is an Erdős-Rényi graph while the real network is the NCAA Division-I football network (Park and Newman 2005). Colors denote the entries of the portrait matrix $B$ (Eq. (2); white indicates $B_{\ell,k} = 0$).

Fig. 1 Example networks and their portraits. The random network is an Erdős-Rényi graph while the real network is the NCAA Division-I football network (Park and Newman 2005). Colors denote the entries of the portrait matrix $B$ (Eq. (2); white indicates $B_{\ell,k} = 0$)

FIG. 12. (Color online) Evolution of a $B$ matrix portrait when using the order-parameter performance measure with $\hat{\sigma}=0.6$. Initial topology was a periodic ring lattice with nearest and next-nearest-neighbor coupling. $B$ matrices were taken at iterations (a) 1, (b) 4000, (c) 8000, (d) 14 000, (e) 20 000, and (f) 180 000. Colors represent the number of nodes at a given index within the $B$ matrix and are plotted on a log scale $[\log(b_{ik})]$. 

Fig. 2: (Color online) (a) A $B$-Matrix with a logarithmic color scale (the white background indicates zero elements of $B$). The degree distribution is slightly visible in the first row. The “turning point” about row 4 represents finite-size effects. Shown is the network of the 10% most connected actors on IMDB [2]. (b) The same matrix with a logarithmic horizontal axis. The degree distribution is now clearly visible.

FUN FACT
This is the first published network portrait.
How exactly do we get a graph's B-Matrix? How do we interpret a network portrait? That's exactly what BMatrix_Explainer is all about.

github.com/dirediredock/BMatrix_Explainer

To get started with BMATRIX_EXPLAINER, consider this small graph as an explainer example:

- It has 7 nodes and 14 edges
- Edgelist:  
  1-2  
  1-3  
  2-4  
  2-5  
  2-6  
  2-3  
  3-4  
  3-5  
  3-6  
  4-5  
  4-6  
  5-7  
  5-6  
  6-7
We start by picking a node, and initialize an empty matrix.
Rows are for number of hops away from this starting node, and columns are for node counts.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
We only have one node at hand (no hops yet), so we **flip the bit** at zeroth row and first column.
Then at first hop, there are two nodes - so we flip the bit at the first row and second column.
At second hop there are three nodes, so we flip the bit at the second row and third column.
At third hop there is one node (last one), so we flip the bit at the third row and first column.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This completes the bit matrix of node 1 (of 7).

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
We repeat and get a bit matrix for node 2 (of 7), and we store the already completed matrix.
This is the bit matrix of node 3 (of 7), and we save the two already completed matrices.
The bit matrix of node 4 (of 7), and we save the three already completed matrices.
The bit matrix of node 5 (of 7), and we save the four already completed matrices.
The bit matrix of node 6 (of 7), and we save the five already completed matrices.
Finally, the bit matrix of node 7 (of 7), and we save the six already completed matrices.
We add these seven bit matrices element-wise into a single matrix. This ends the algorithm.
We’re done!

The B-Matrix of the graph.

And it has a bunch of properties.
Also each row adds up to the total number of nodes.
The first row marks the number of times different node counts happened at exactly one hop away from the starting node.

The node degree is how many edges a node has, so this row is effectively a record of frequency of node degrees.

<table>
<thead>
<tr>
<th>Node Degree</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-degree</td>
<td>7</td>
</tr>
<tr>
<td>1-degree</td>
<td>0</td>
</tr>
<tr>
<td>2-degree</td>
<td>0</td>
</tr>
<tr>
<td>3-degree</td>
<td>0</td>
</tr>
<tr>
<td>4-degree</td>
<td>0</td>
</tr>
<tr>
<td>5-degree</td>
<td>0</td>
</tr>
</tbody>
</table>

1-degree

3-degree

5-degree

2-degree

4-degree
We can visualize the distribution of node degrees with a bar chart of counts (histogram).

Frequency of node degrees
Two nodes of degree 2

Frequency of node degrees

```
0 7 0 0 0 0 0
0 0 2 0 1 4
0 4 1 2 0 0
5 2 0 0 0 0
```
[Row 1, Column 4] One node of degree 4

<table>
<thead>
<tr>
<th>Degree</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
[Row 1, Column 5] Four nodes of degree 5

Frequency of node degrees
Finally, the last row is the maximum number of hops the algorithm got through before running out of nodes.

In other words, the final hop number is equivalent to the **diameter** of the graph, *L-shell* of 3 in this case.

![Frequency of node 3-shell degrees](image)
The graph diameter is the length of the shortest path between the two most distanced nodes.
In contrast to this explainer example, the B-Matrix of a real-world graph can be very large.

It is not practical to show this data abstraction directly with numbers, we need a visual encoding.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>7</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In literature this is solved by mapping the B-Matrix node count to a colormap range.
And the result is a heatmap.
However, we can do one more edit to increase information resolution in this heatmap idiom.

Notice that the zeroth row always has the highest value (which sets the colormap extreme).

In large networks this value can be so high that the colormap must be log-transformed.
We can safely remove the zeroth row.

This is fine because this row only contains redundant data (total node count).
Then we can get higher fidelity by rescaling the colormap.
That’s it!
This figure is the network portrait of the graph.
Now let's explore real-world networks with

**BMatrix_Explainer**

a Python-based B-Matrix visualization GitHub repo.
Recall that each row of the B-Matrix can be visually encoded as stacked bars for count data.
BMATRIX_Explainer fully features this visualization with bars encoding node counts, or histograms.
And that is not all! To further support interpretation tasks, in BMatrix_Explainer both color and bar encodings can have per-row normalization to enhance information discovery within hop level.
<table>
<thead>
<tr>
<th>Graph</th>
<th>Fall 2000 College Football Games personal.umich.edu/~mejn/netdata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>115</td>
</tr>
<tr>
<td>Edges</td>
<td>613</td>
</tr>
</tbody>
</table>

A small “small-world” network.
Knowledge network from *Fortran* infobox.
Knowledge network from Carl Jung infobox.
For future work, I would like to build a **B-Matrix reverse-highlight visualization system**. This can help network exploration tasks such as understanding nodes with special properties, where these located, and in relation to what global network features.

Utah Teapot has nodes at degree-40.
Thank You!

To check out code, data, and more figures

https://github.com/dirediredock/BMatrix_Explainer
https://github.com/dirediredock/infobox_interlinker