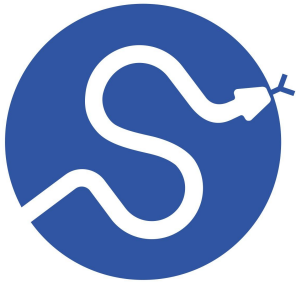
SciPy Sparse:



At SciPy 2024

Sparse Arrays in scipy.sparse

Dan Schult, Colgate University - Scipy & NetworkX

Funding from:

Chan Zuckerberg Initiative

COLGATE UNIVERSITY

Thanks to: SciPy Developers

Especially CJ Carey & Stéfan van der Walt

Sparse Matrix:

Dense

0	0	5	0
0	-2	0	0
7	0	0	0

Sparse

Coords,	Data
(0, 2)	5
(1, 1)	-2
(2, 0)	7

Axis1:	0	2	5
Axis2:	1	1	-2
Data:	2	0	7

Sparse Matrix:

(MxN)

Dense

N cols

0	0	5	0
0	-2	0	0
7	0	0	0

M rows

Sparse

Coords,	Data
(0, 2)	5
(1, 1)	-2
(2, 0)	7

Axis1:
Axis2:
Data:

0	2	5
1	1	-2
2	0	7

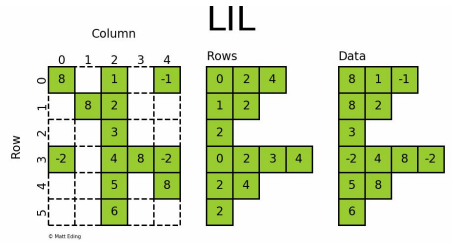
Notation: “nnz”,“COO”
nnz=3 values.
COO format.

We lose position information, so coord info is needed. But for a sparse matrix we still save memory and computation time.

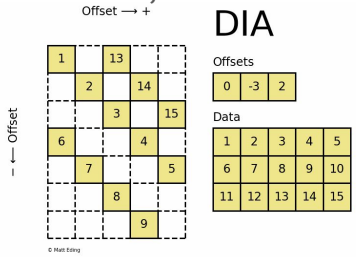
Sparse means “not dense”. So there are many flavors.

DOK: (Dict-Of-Keys)
{(0, 2): 5, (1, 1): -2, (2, 0): 7}

LIL: (List of Lists)
Each row stores 2 arrays: column numb, data

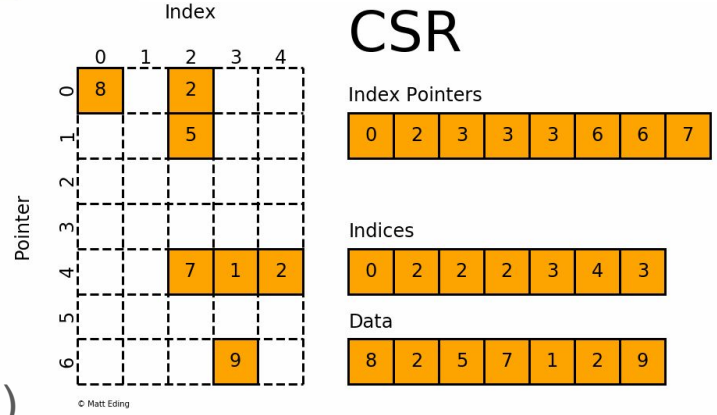


DIA: (non-empty diagonals)
Diagonals stored with offset



Animations courtesy of
Matt Eding’s blog post on Sparse Matrices

CSR/CSC: (Compressed row/column)
Sort by rows
A.data[k] data for k^{th} value
A.indices[k] column number for k^{th} value
A.indptr[r] 1st data/indices position for r^{th} row
(only need M entries for NNZ values)



Ideal: Drop-in replacement for NumPy array

NumPy **A[2,:] = (B*D - (C1 == C2) *F).sum()**

Ideally **scipy.sparse** would result in the **same code** for sparse!

Long Ago Choice: Drop-in for **numpy.matrix** or **numpy.array?**

Answer: **Numpy.matrix!**

- Closer to matrix algebra
- Always 2D matrix

A[2, :] -> [[0, 2, 3, 0]], not **[0, 2, 3, 0]**
A[:, 3] -> [[0], [0], [0]] not **[0, 0, 0]**

- **A*B** means **matrix multiply**
- **A.multiply(B)** means **element**-wise multiply
- **A**3** means **matrix power**
- No element-wise power

NumPy wants to **get rid of np.matrix** interface

SciPy wants a **scipy.sparse array** interface

(2008) NumPy discussions first considered deprecating **np.matrix**
(2015) Python introduces **@** operator to represent matrix multiplication
(2017) NumPy starts to officially discourage use of **np.matrix**

(2022) SciPy introduces a sparse array interface: **csr_array**, etc.
(2023) **scipy.sparse** construction functions **eye_array** introduced
(2024) **scipy.sparse** supports 1D arrays. 2D has full features except indexing.

Future: full featured 1D and 2D sparse arrays in v1.15 (currently in `main`)

More details at Stack Overflow

Sparse Array Interface

operators:

@ - matrix multiply (dot product)

* - element-wise multiply

:can still use **A.multiply(B)**

** - element-wise power

Matrix power can use **sp.sparse.linalg.matrix.power(A, n)**

---

shapes:

1D input creates 1D arrays: **A = sp.sparse.csr_array([1, 0, 0, 3])**

Indexing resulting shape:

scalars:	**A[3, 4]**	→ **7.3**
1D arrays:	**A[2, 3:5]**	→ **[5.1, 0]**
2D arrays:	**A[1:3, 3:5]**	→ **[[2.3, 0], [0, 4.1]]**

Sparse Constructors

Classes:	**csr_matrix**	→	**csr_array**
	dok_matrix	→	**dok_array**
	coo_matrix	→	**coo_array**
	…		…

Construction functions:

eye	→	**eye_array**
random	→	**random_array**
diags	→	**diags_array**
bmat	→	**block**

Manipulation functions:

vstack, hstack, kron, block, triu, tril

If any input is a sparse array, the result is a sparse array.

Migration from spmatrix to sparray

First pass:

0) **Make sure you have good tests**

1) **Update all sparse code to use latest spmatrix interface**

- **A@B, sp.sparse.linalg.matrix_power(A,3)**
- Use 2D input to class constructors
- Replace removed methods like **A.getnnz → A.nnz, A.A → A.toarray()**
- Change **isspmatrix_csr** functions to **issparse(A) and A.format == 'csr'**

Test before the change

Migration from spmatrix to sparray

Second pass:

2) Switch spmatrix to sparray:

- Convert all class constructors
- Convert constructor functions
- Convert any code that reduces shape of arrays (indices, reduction methods)

  A[4], **A.sum()**, etc
- Iterations over sparse_arrays yield 1D sparrays
- Replace code that uses np.matrix features.
- Reread your code. Test your code again!

Managing both spmatrix and sparray

If your library needs to accept spmatrix input, convert to sparray upon input.

If you want to have code to work with each type, you can reduce duplication by avoiding **A*B** except for scalar multiplication **3*A** or **4*9**
Use **A.multiply(B)** and/or **A@B** instead of **A*B**

Use **issparse(A)** to check if any **scipy.sparse** type.
Use **isspmatrix(A)** or **isinstance(A, sparray)** to check spmatrix vs sparray
Use **A.format** attribute to check if “coo”, “csr”, “csc”, “dia”, “dok”, “lil”, “bsr”.
Use array indexing e.g. **A[[3], :]** to ensure 2D return shape.

Bigger Picture Migration Path
(How has SciPy released the changes?)

1) Introduce new classes: sparray alongside spmatrix
- Mostly the same code. Use if statements where not the same.
- As API diverged (e.g. A.getnnz() → A.nnz),
  methods got moved to an spmatrix “Mixin” class.

2) Rewrite helper functions:
- Split functions into two groups: constructors and manipulators.
  - Constructors were given new function names, e.g. **eye_array()**
  - Manipulators use the same name with output based on input e.g. **vstack()**

3) Add support for 1D arrays, **, other new functionality

Release and Deprecation Timetable

1) Announce sparse arrays once full functionality obtained
   - provide migration support
   - wait 1-2 release cycles

2) Announce deprecation of spmatrix
   - provide migration support
   - wait 2+ release cycles

3) Remove spmatrix (turn off old function names)
   (possibly make old function names aliases for array style names:
   e.g. alias **eye(5)** to match array-api)

Decisions about API changes and releases

1) Allow both old and new **side-by-side** *vs* transition in **one release**
   - We chose side-by-side. And in general chose least backward issues.
     - This made transition take much longer (good and bad aspects).
     - Cleanup of existing API warts happened piecemeal over time.
     - New API warts introduced by need to separate class handling.

2) Build **two namespaces** that duplicate names *vs* **use new names**
   - 2 namespaces: migration could be as easy as changing an import statement.
     - In practice, only true if new API matches old exactly. (and it doesn't)

3) **New package** *vs* **updating** the old package.
   - SciPy has a useful history if we maintain it.
   - Other folks are introducing new packages.

Thanks for listening

Recap for **scipy.sparse**:

- Sparse arrays are here in 1D and 2D
- Fully functional (add indexing) on github main and coming in v1.15

- Migration guide and developer support for migrating is now a major focus

- API has changed slightly – but carefully: hope to reduce difficulty in switching.