SymPy: Symbolic Mathematics

Hendrik Speleers

SymPy: Symbolic Mathematics

Università di Roma
Tor Vergata

- Overview
  - A symbolic calculator
  - Algebraic manipulation
    - Substitution, simplification, factorization, ...
  - Calculus
    - Series, limits, differentiation and integration
  - Linear algebra
    - Matrix manipulation and decompositions
  - Solvers
    - Linear and nonlinear (system of) equations

SymPy: Symbolic Mathematics

Università di Roma
Tor Vergata

- SymPy
  - Symbolic Python
  - Python extension for symbolic mathematics
    - Performing algebraic manipulations on symbolic expressions
    - Evaluating expressions with arbitrary precision
    - Similar to: Mathematica, Maple
    - Keeping code as simple as possible and easily extensible
  - Import convention

```python
import sympy as sym
```

SymPy: Symbolic Mathematics

- A symbolic calculator
  - A simple demo

```
In [1]: x, y = sym.symbols('x y')
In [2]: expr = x + 3*y
In [3]: expr
Out[3]: x + 3*y
In [4]: expr + 1
Out[4]: x + 3*y + 1
In [5]: (expr + x)**2
Out[5]: (2*x + 3*y)**2
```

SymPy: Symbolic Mathematics

- A symbolic calculator
  - Symbolic variables need to be explicitly defined as symbols

```
In [1]: x = sym.symbols('x')
In [2]: x, y = sym.symbols('x y')
In [3]: x, y = sym.symbols('x,y')
In [4]: x, y, z = sym.symbols('x:z')
```

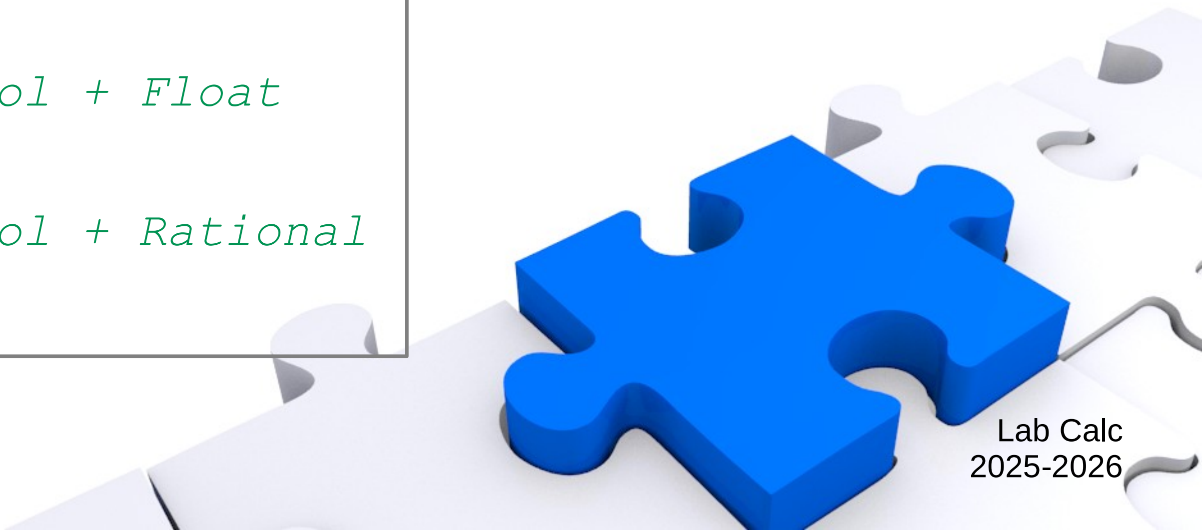
  - Specific properties can be assigned to symbols

```
In [5]: k = sym.symbols('k', integer=True)
In [6]: x, y, z = sym.symbols('x,y,z', real=True)
```

SymPy: Symbolic Mathematics

- A symbolic calculator
  - Data types
    - Symbolic: `sym.Symbol`
    - Numeric: `sym.Integer`, `sym.Rational`, `sym.Float`
  - Be careful when using integer fractions

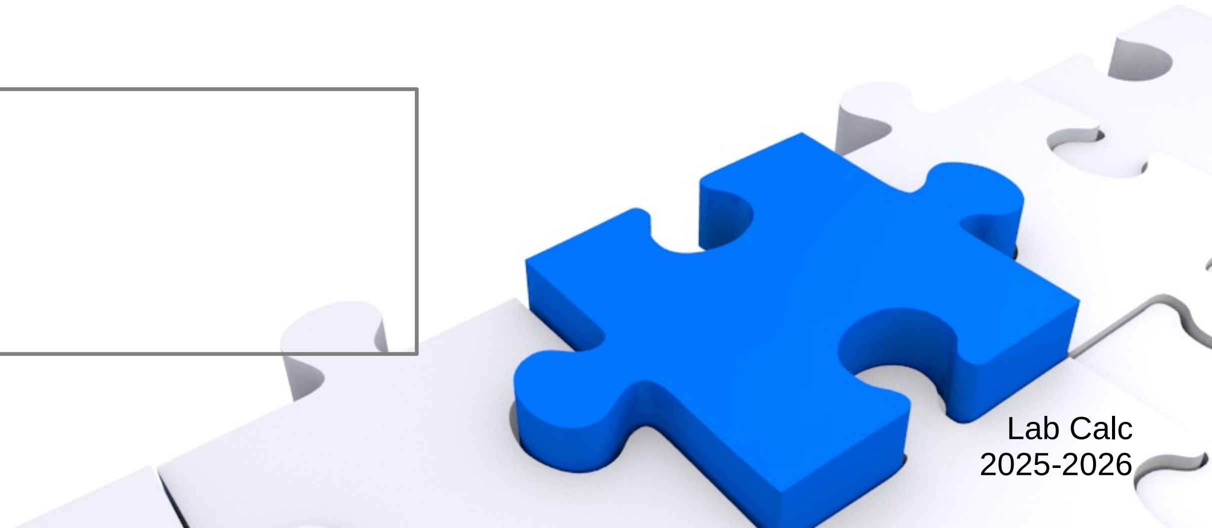
```
In [1]: x = sym.Symbol('x')
In [2]: x + 1/2                # Symbol + Float
Out[2]: x + 0.5
In [3]: x + sym.Rational(1, 2) # Symbol + Rational
Out[3]: x + 1/2
```



SymPy: Symbolic Mathematics

- A symbolic calculator
  - Symbolic constants: `sym.pi`, `sym.E`, `sym.I`, `sym.oo`
  - Symbolic functions:
    - Number-theory functions: `sym.factorial(n)`, `sym.binomial(n, k)`
    - Power functions: `sym.exp(x)`, `sym.log(x)`, `sym.sqrt(x)`, `sym.root(x, n)`
    - Trigonometric functions: `sym.cos(x)`, `sym.sin(x)`, `sym.tan(x)`
    - Many more...

```
In [1]: f = 4*sym.pi**2
In [2]: sym.sqrt(f)
Out[2]: 2*pi
```



SymPy: Symbolic Mathematics

- Basic manipulation
  - Substitution
    - Single variable: `expr.subs(old, new)`
    - Multiple variables: `expr.subs(iterable)`

```
In [1]: expr = 1 + x*y
In [2]: expr.subs(x, sym.pi)
Out[2]: pi*y + 1
In [3]: expr.subs({x:sym.pi, y:2})
Out[3]: 1 + 2*pi
In [4]: expr.subs([(x, y+2), (y, 2)])
Out[4]: 9
```

SymPy: Symbolic Mathematics

- Basic manipulation
  - Numerical evaluation
    - Arbitrary precision: `expr.evalf(n=15, subs=None)`

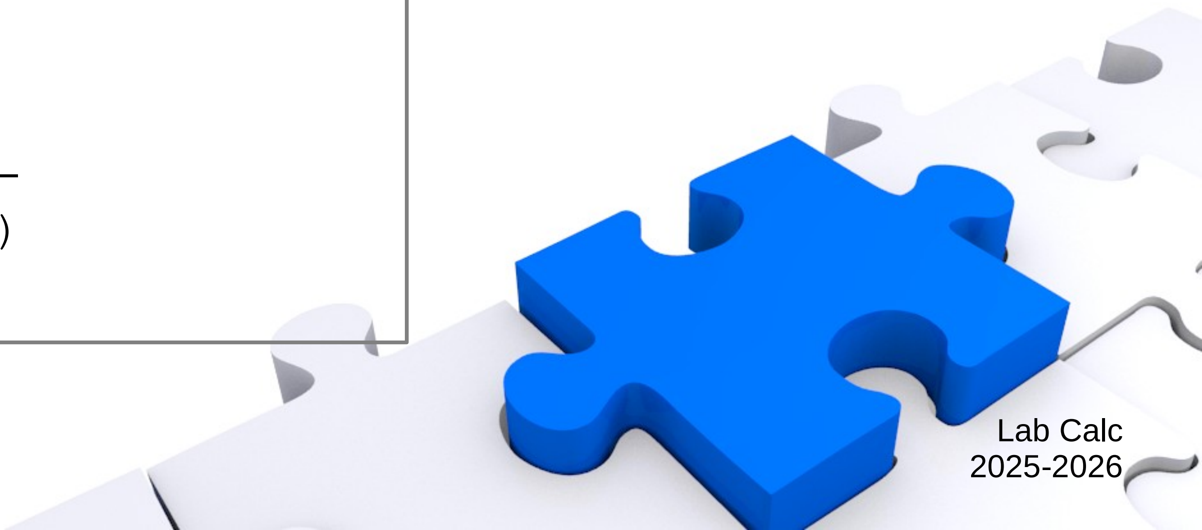
```
In [1]: expr = sym.sqrt(8)
In [2]: expr, expr.evalf()
Out[2]: (2*sqrt(2), 2.82842712474619)
In [3]: sym.pi.evalf(100)
Out[3]: 3.141592653589793238462643383279502884...
In [4]: sym.exp(x).evalf(subs={x:2})
Out[4]: 7.38905609893065
```

SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Basic manipulation
  - Numerical evaluation
    - Be careful when combining with `numpy`

```
In [1]: a = sym.exp(2).evalf()
In [2]: type(a)
Out[2]: sympy.core.numbers.Float
In [3]: a_array = np.array([a, a])
In [4]: np.sin(a_array)
--------------------------------------
AttributeError ----> 1 np.sin(a_array)
'Float' object has no attribute 'sin'
```



SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Basic manipulation
  - Numerical evaluation
    - Be careful when combining with `numpy`

```
In [1]: a = sym.exp(2).evalf()
In [2]: a_float = float(a)
In [3]: type(a_float)
Out[3]: float
In [4]: a_array = np.array([a_float, a_float])
In [5]: np.sin(a_array)
Out[5]: array([0.89385495, 0.89385495])
```

SymPy: Symbolic Mathematics

- Algebraic manipulation
  - Simplification
    - General purpose: `sym.simplify(expr)` or `expr.simplify()`
    - Several more specific functions (`trigsimp`, `powsimp`, `combsimp`, `...`)

```
In [1]: expr = sym.sin(x)**2 + sym.cos(x)**2
In [2]: expr.simplify()
Out[2]: 1
In [3]: num = x**3 + x**2 - x - 1
   ...: denom = x**2 + 2*x + 1
In [4]: sym.simplify(num/denom)
Out[4]: x - 1
```

SymPy: Symbolic Mathematics

- Algebraic manipulation
  - Polynomial manipulation
    - Expansion: `sym.expand(expr)` or `expr.expand()`
    - Factorisation: `sym.factor(expr)` or `expr.factor()`
    - Collect powers: `sym.collect(expr, syms)` or `expr.collect(syms)`

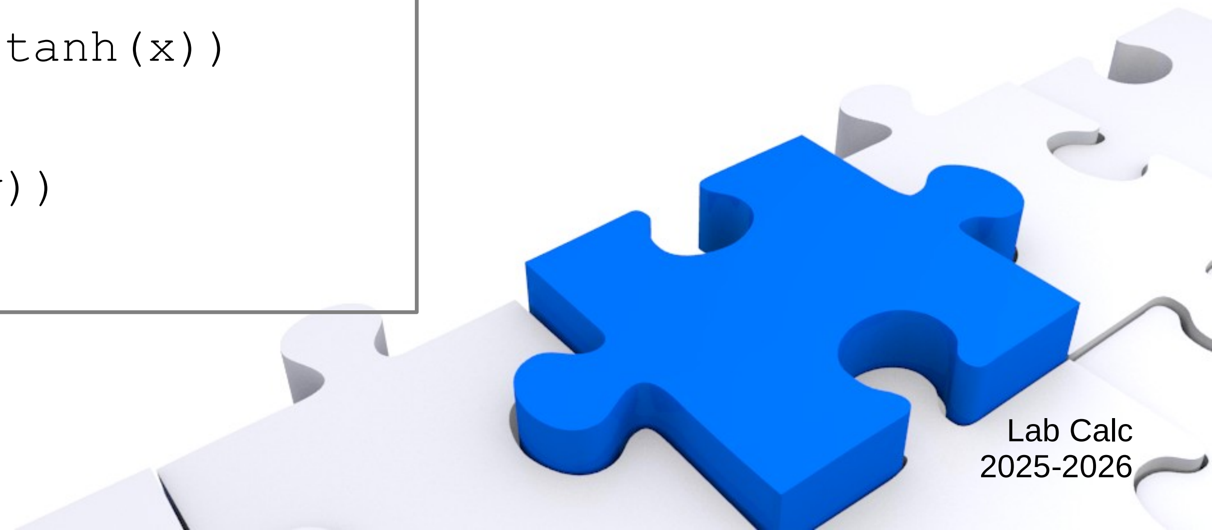
```
In [1]: sym.expand((x + 2)*(x - 3))
Out[1]: x**2 - x - 6
In [2]: sym.factor(x**3 - x**2 + x - 1)
Out[2]: (x - 1)*(x**2 + 1)
In [3]: sym.collect(2*x**2 + x*y + x - z*x**2, x)
Out[3]: x**2*(2 - z) + x*(y + 1)
```

SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Algebraic manipulation
  - Trigonometric manipulation
    - Simplification: `sym.trigsimp(expr)` or `expr.trigsimp()`
    - Expansion: `sym.expand_trig(expr)`

```
In [1]: sym.trigsimp(sym.sin(x)/sym.sec(x))
Out[1]: sin(2*x)/2
In [2]: sym.trigsimp(sym.sinh(x)/sym.tanh(x))
Out[2]: cosh(x)
In [3]: sym.expand_trig(sym.sin(x + y))
Out[3]: sin(x)*cos(y) + sin(y)*cos(x)
```



SymPy: Symbolic Mathematics

Università di Roma
Tor Vergata

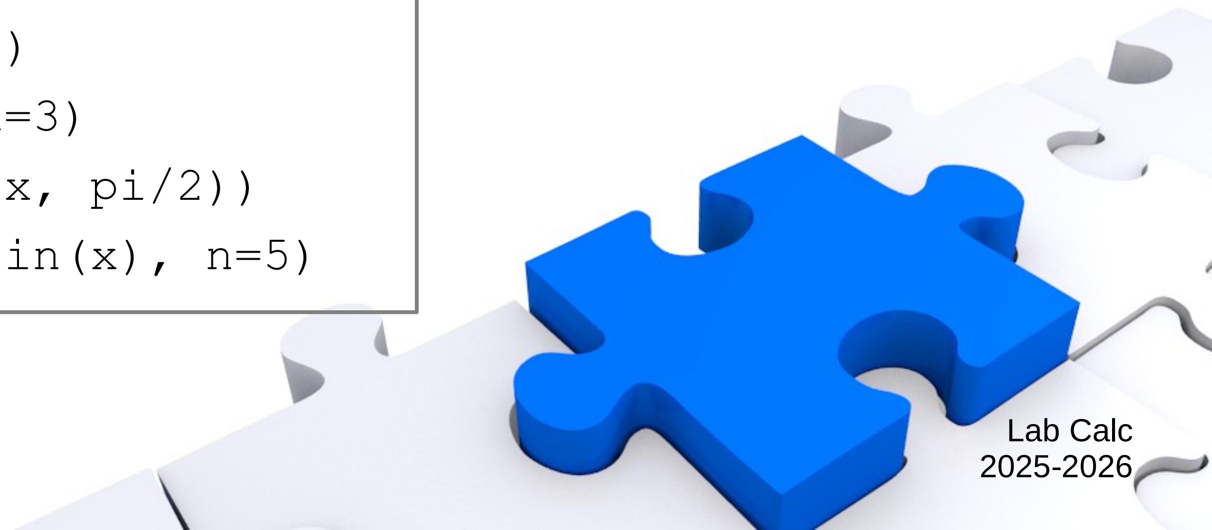
- Calculus
  - Sums and products
    - Syntax: `sym.summation(expr, *args), sym.product(expr, *args)`
    - The range represented by tuples (index, a, b) is collected in `*args`

```
In [1]: sym.summation(1/(2**k), (k, 0, sym.oo))
Out[1]: 2
In [2]: sym.summation((k+1)*(x+k), (k, 0, 4))
Out[2]: 15*x + 40
In [3]: sym.product(2*k+x, (k, 0, 4))
Out[3]: x*(x + 2)*(x + 4)*(x + 6)*(x + 8)
```

SymPy: Symbolic Mathematics

- Calculus
  - Taylor series
    - Syntax: `sym.series(expr, *args)` or `expr.series(*args)`
    - The variable, expansion point and order are collected in `*args`

```
In [1]: expr = sym.cos(x)
In [2]: expr.series()
Out[2]: 1 - x**2/2 + x**4/24 + O(x**6)
In [3]: expr.series(x, x0=sym.pi/2, n=3)
Out[3]: pi/2 - x + O((x - pi/2)**3, (x, pi/2))
In [4]: taylor_sin = sym.series(sym.sin(x), n=5)
```



SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Calculus
  - Limits
    - Syntax: `sym.limit(expr, *args)` or `expr.limit(*args)`
    - The variable and limit point are collected in `*args`

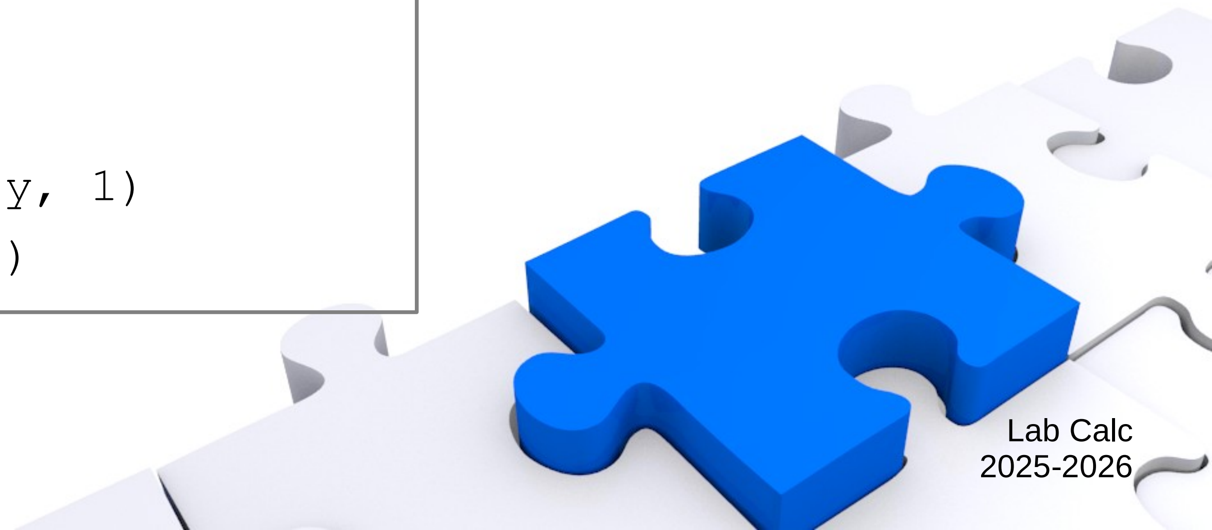
```
In [1]: expr = x**2/sym.exp(x)
In [2]: expr.subs(x, sym.oo)
Out[2]: nan
In [3]: expr.limit(x, sym.oo)
Out[3]: 0
In [4]: lim_sinc = sym.limit(sym.sin(x)/x, x, 0)
```

SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Calculus
  - Differentiation
    - Syntax: `sym.diff(expr, *args)` or `expr.diff(*args)`
    - Variables and/or the derivative order are collected in `*args`

```
In [1]: sym.sin(x).diff()
Out[1]: cos(x)
In [2]: sym.diff(sym.sin(x), x, 3)
Out[2]: -cos(x)
In [3]: sym.diff(sym.sin(x*y), x, 2, y, 1)
Out[3]: -y*(x*y*cos(x*y) + 2*sin(x*y))
```



SymPy: Symbolic Mathematics

Università di Roma
Tor Vergata

- Calculus
  - Integration
    - Syntax: `sym.integrate(expr, *args)` or `expr.integrate(*args)`
    - Indefinite: variables are collected in `*args`
    - Definite: tuples (var, a, b) are collected in `*args`

```
In [1]: sym.cos(x).integrate()
Out[1]: sin(x)
In [2]: sym.integrate(sym.cos(x), x)
Out[2]: sin(x)
In [3]: sym.integrate(sym.exp(-x), (x, 0, sym.oo))
Out[3]: 1
```

SymPy: Symbolic Mathematics

- Calculus
  - Numerical evaluation
    - For integrals/sums/products: combine `evalf` with `sym.Integral`/`sym.Sum`/`sym.Product`
    - Can be useful when symbolic computation is very hard

```
In [1]: expr = sym.exp(sym.sin(x))
In [2]: sym.integrate(expr, (x, 0, 4))
Out[2]: Integral(exp(sin(x)), (x, 0, 4))
In [3]: sym.Integral(expr, (x, 0, 4)).evalf()
Out[3]: 6.79647242142120
In [4]: sym.Sum(1/k**2, (k, 1, sym.oo)).evalf()
Out[4]: 1.64493406684823
```

SymPy: Symbolic Mathematics

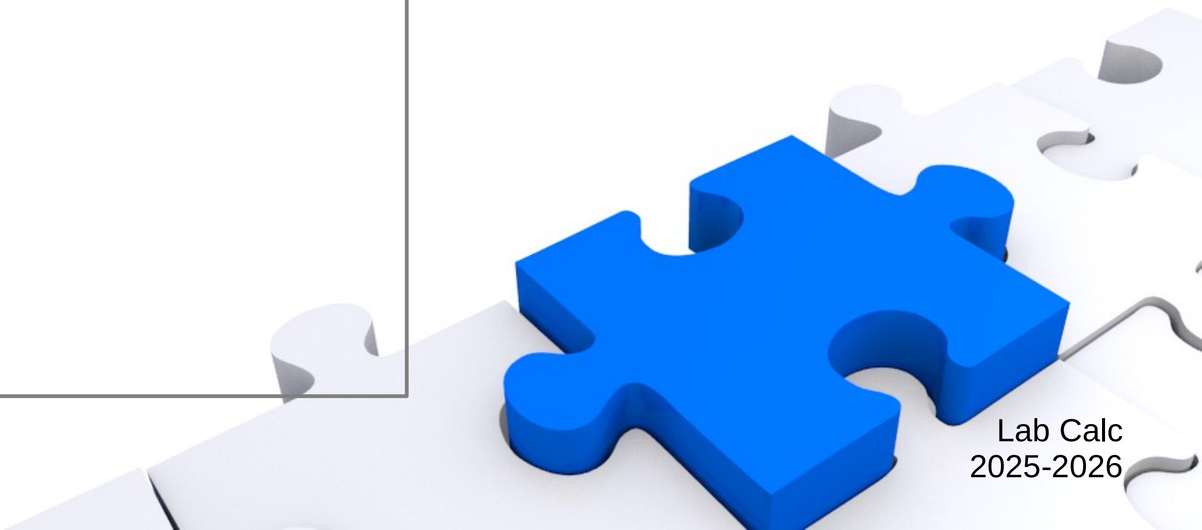
- Linear algebra
  - Matrix data type
  - Matrix creation via `sym.Matrix`
    - List of lists, flattened list, or function as input

```
In [1]: M1 = sym.Matrix([[0, 1, 2], [3, 4, 5]])
In [2]: M2 = sym.Matrix(2, 3, [0, 1, 2, 3, 4, 5])
In [3]: def f(i,j): return 3*i+j
In [4]: M3 = sym.Matrix(2, 3, f)
In [5]: M3
Out[5]: Matrix([[0, 1, 2],
   ...:         [3, 4, 5]])
```

SymPy: Symbolic Mathematics

- Linear algebra
  - Special matrices
    - Common: `sym.eye(rows, cols=None), sym.ones(...), sym.zeros(...)`
    - Diagonal: `sym.diag(*values)`

```
In [1]: A = sym.eye(2)
In [2]: B = sym.ones(2, 3)
In [3]: C = sym.diag(3, B, 2)
In [4]: C
Out[4]: Matrix([[3, 0, 0, 0, 0],
   ...:         [0, 1, 1, 1, 0],
   ...:         [0, 1, 1, 1, 0],
   ...:         [0, 0, 0, 0, 2]])
```



SymPy: Symbolic Mathematics

Università di Roma
Tor Vergata

- Linear algebra
  - Matrix manipulation
    - Indexing and slicing (pass by value)

```
In [1]: M = sym.Matrix([[0, 1, 2], [3, 4, 5]])
In [2]: M[1, 2] = x
In [3]: M[-1, :]
Out[3]: Matrix([[3, 4, x]])
In [4]: M2 = M[:, :]
In [5]: M2[0, 0] = 100
In [6]: M[0, 0] == 100
Out[6]: False
```

SymPy: Symbolic Mathematics

- Linear algebra
  - Matrix manipulation
    - Access: `M.row(i)`, `M.col(i)`
    - Deletion (in-place): `M.row_del(i)`, `M.col_del(i)`
    - Insertion (copy): `M.row_insert(i, matrix)`, `M.col_insert(i, matrix)`

```
In [7]: M.row(1)
Out[7]: Matrix([[3, 4, x]])
In [8]: M.row_insert(0, sym.Matrix([[1, -2, 3]]))
Out[8]: Matrix([[1, -2, 3],
   ...:         [0,  1, 2],
   ...:         [3,  4, x]])
```

SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Linear algebra
  - Matrix operations
    - Basic arithmetic operations (+, −, *, **)

```
In [1]: M = sym.Matrix([[-1, 1], [2, x]])
In [2]: b = sym.Matrix([4, 5])
In [3]: M * b
Out[3]: Matrix([[      1],
   ...:         [5*x + 8]])
In [4]: M**2
Out[4]: Matrix([[      3,      x - 1],
   ...:         [2*x - 2, x**2 + 2]])
```

Be careful: these are linear algebra operations: * and ** not elementwise!

SymPy: Symbolic Mathematics

Università di Roma Tor Vergata

- Linear algebra
  - Matrix operations
    - Transpose: `M.T`
    - Determinant: `M.det()`
    - Inverse: `M.inv()`
    - Nullspace: `M.nullspace()`
    - Columnspace: `M.columnspace()`
    - Eigenvalues: `M.eigenvals()`, `M.eigenvects()`

```
In [5]: M.det()
Out[5]: -x - 2
```

SymPy: Symbolic Mathematics

- Linear algebra
  - Matrix decompositions
    - Diag: `P, D = M.diagonalize()`
    - LU: `L, U, perm = M.LUdecomposition()`
    - Cholesky: `C = M.cholesky()`
    - QR: `Q, R = M.QRdecomposition()`
  - And solvers of $M * x = b$ based on decompositions
    - Diag: `x = M.diagonal_solve(b)`
    - LU: `x = M.LUsolve(b)`
    - Cholesky: `x = M.cholesky_solve(b)`
    - QR: `x = M.QRsolve(b)`

SymPy: Symbolic Mathematics

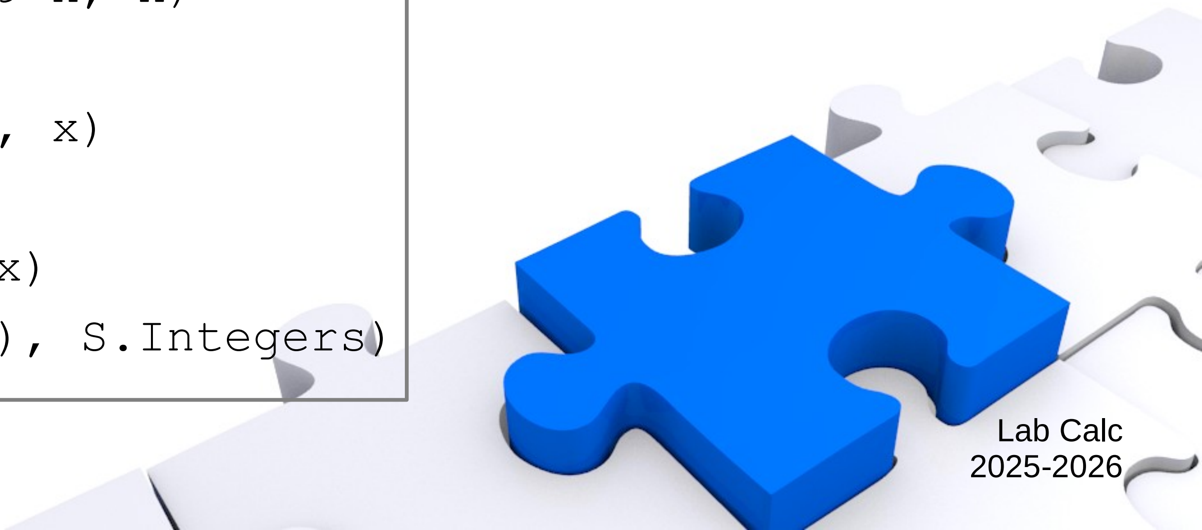
- Linear algebra
  - Example

```
In [1]: M = sym.Matrix([[-1, 0], [2, x]])
   ...: B = sym.Matrix([[x+1, 3], [5, 1]])
In [2]: M.inv() * B
Out[2]: Matrix([[             -x - 1,  -3],
   ...:         [2*(x + 1)/x + 5/x, 7/x]])
In [3]: M.LUsolve(B)
Out[3]: Matrix([[      -x - 1,  -3],
   ...:         [(2*x + 7)/x, 7/x]])
In [4]: sym.simplify(Out[2]-Out[3]).is_zero
Out[4]: True
```

SymPy: Symbolic Mathematics

- Solvers
  - Univariate equation
    - general: `sym.solveset(equation, symbol=None)`
    - polynomial: `sym.roots(equation, symbol=None)`
    - Equation assumed to equal zero

```
In [1]: sym.solveset(x**3 - 6*x**2 + 9*x, x)
Out[1]: {0, 3}
In [2]: sym.roots(x**3 - 6*x**2 + 9*x, x)
Out[2]: {0: 1, 3: 2}
In [3]: sym.solveset(sym.exp(x) - 1, x)
Out[3]: ImageSet(Lambda(_n, 2*_n*I*pi), S.Integers)
```



SymPy: Symbolic Mathematics

- Solvers
  - System of equations
    - System of linear eqns.: `sym.linsolve(eqns, *symbols)`
    - System of nonlinear eqns.: `sym.nonlinsolve(eqns, *symbols)`
    - Equations assumed to equal zero

```
In [1]: eqns = [x + y + z - 1, x + y + 2*z - 3]
In [2]: sym.linsolve(eqns, x, y, z)
Out[2]: {(-y - 1, y, 2)}
In [3]: sym.linsolve(eqns, y, z)
Out[3]: {(-x - 1, 2)}
In [4]: sym.nonlinsolve([x*y - 1, x - 2], x, y)
Out[4]: {(2, 1/2)}
```

SymPy: Symbolic Mathematics

- Pretty printing
  - Default printing is often string-based

```
In [1]: sym.Integral(sym.sqrt(1/x), x)
Out[1]: Integral(sqrt(1/x), x)
```

  - Pretty printing can be activated

```
In [1]: sym.init_printing(pretty_print=True)
In [2]: sym.Integral(sym.sqrt(1/x), x)
Out[2]:
```

$$\int \sqrt{\frac{1}{x}}\, dx$$