

ISSAC26_tutorial

July 13, 2026

1 ISSAC 2026 tutorial

2 *Designing and exploiting fast algorithms for polynomial matrices*

3 Vincent Neiger

[237]: `%display latex`

3.1 high-precision fraud: a guess is not a proof

Take this example:

$$S := \sum_{n=1}^{\infty} \frac{\lfloor n \tanh(\pi) \rfloor}{10^n}$$

this is “Sum 9” from [Borwein-Borwein, *American Mathematical Monthly* 1992]

```
[247]: infinity = 10000
seq = [floor(n * tanh(pi)) for n in range(infinity)]
print("-- sequence terms 0...10:");
show(seq[0:11])
print("-- sequence terms 100...110:");
show(seq[100:111])
print("-- tanh(pi) is close to 1:");
show(tanh(pi).numerical_approx())
print("-- sequence terms 260...270:");
show(seq[260:271])
```

-- sequence terms 0...10:

[0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

-- sequence terms 100...110:

[99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

-- tanh(pi) is close to 1:

0.996272076220750

-- sequence terms 260...270:

[259, 260, 261, 262, 263, 264, 265, 266, 267, 268]

```
[242]: S = sum(seq[n] / 10^n for n in range(infinity))
Srat = 1/81
show(S.numerical_approx())
show(Srat.numerical_approx())
show((S - Srat).numerical_approx())
```

0.0123456790123457

0.0123456790123457

$-1.11111111111111 \times 10^{-269}$

the identity $S = \frac{1}{81}$ is *almost correct*: true up to 268 digits

from another viewpoint, it is also *completely wrong*: the sum S is transcendental

No proof of $S = \frac{1}{81}$. But the approximation may be useful anyway!

```
[250]: Series.<x> = PowerSeriesRing(QQ, default_prec=infinity)

for prec in [100, 200, 400, 800]:
    f = Series(seq) + O(x**prec)
    f_pade = f.pade(prec/2, prec/2-1)
    print("-- precision", prec)
    show(f_pade)

print("-- difference:")
show((f - x^2 / (x-1)^2).pade(prec/2, prec/2-1))
```

-- precision 100

$$\frac{x^2}{x^2 - 2x + 1}$$

-- precision 200

$$\frac{x^2}{x^2 - 2x + 1}$$

-- precision 400

$$\frac{x^2}{x^2 - 2x + 1}$$

-- precision 800

$$\frac{x^{268} + x^{267} + x^{266} + x^{265} + x^{264} + x^{263} + x^{262} + x^{261} + x^{260} + x^{259} + x^{258} + x^{257} + x^{256} + x^{255} + x^{254} + x^{253} + x^{252} + x^{251} + x^{250} + x^{249} + x^{248} + x^{247} + x^{246} + x^{245} + x^{244} + x^{243} + x^{242} + x^{241} + x^{240} + x^{239} + x^{238} + x^{237} + x^{236} + x^{235} + x^{234} + x^{233} + x^{232} + x^{231} + x^{230} + x^{229} + x^{228} + x^{227} + x^{226} + x^{225} + x^{224} + x^{223} + x^{222} + x^{221} + x^{220} + x^{219} + x^{218} + x^{217} + x^{216} + x^{215} + x^{214} + x^{213} + x^{212} + x^{211} + x^{210} + x^{209} + x^{208} + x^{207} + x^{206} + x^{205} + x^{204} + x^{203} + x^{202} + x^{201} + x^{200} + x^{199} + x^{198} + x^{197} + x^{196} + x^{195} + x^{194} + x^{193} + x^{192} + x^{191} + x^{190} + x^{189} + x^{188} + x^{187} + x^{186} + x^{185} + x^{184} + x^{183} + x^{182} + x^{181} + x^{180} + x^{179} + x^{178} + x^{177} + x^{176} + x^{175} + x^{174} + x^{173} + x^{172} + x^{171} + x^{170} + x^{169} + x^{168} + x^{167} + x^{166} + x^{165} + x^{164} + x^{163} + x^{162} + x^{161} + x^{160} + x^{159} + x^{158} + x^{157} + x^{156} + x^{155} + x^{154} + x^{153} + x^{152} + x^{151} + x^{150} + x^{149} + x^{148} + x^{147} + x^{146} + x^{145} + x^{144} + x^{143} + x^{142} + x^{141} + x^{140} + x^{139} + x^{138} + x^{137} + x^{136} + x^{135} + x^{134} + x^{133} + x^{132} + x^{131} + x^{130} + x^{129} + x^{128} + x^{127} + x^{126} + x^{125} + x^{124} + x^{123} + x^{122} + x^{121} + x^{120} + x^{119} + x^{118} + x^{117} + x^{116} + x^{115} + x^{114} + x^{113} + x^{112} + x^{111} + x^{110} + x^{109} + x^{108} + x^{107} + x^{106} + x^{105} + x^{104} + x^{103} + x^{102} + x^{101} + x^{100} + x^{99} + x^{98} + x^{97} + x^{96} + x^{95} + x^{94} + x^{93} + x^{92} + x^{91} + x^{90} + x^{89} + x^{88} + x^{87} + x^{86} + x^{85} + x^{84} + x^{83} + x^{82} + x^{81} + x^{80} + x^{79} + x^{78} + x^{77} + x^{76} + x^{75} + x^{74} + x^{73} + x^{72} + x^{71} + x^{70} + x^{69} + x^{68} + x^{67} + x^{66} + x^{65} + x^{64} + x^{63} + x^{62} + x^{61} + x^{60} + x^{59} + x^{58} + x^{57} + x^{56} + x^{55} + x^{54} + x^{53} + x^{52} + x^{51} + x^{50} + x^{49} + x^{48} + x^{47} + x^{46} + x^{45} + x^{44} + x^{43} + x^{42} + x^{41} + x^{40} + x^{39} + x^{38} + x^{37} + x^{36} + x^{35} + x^{34} + x^{33} + x^{32} + x^{31} + x^{30} + x^{29} + x^{28} + x^{27} + x^{26} + x^{25} + x^{24} + x^{23} + x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} + x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1}{x^{269} - x^{268} - x + 1}$$

-- difference:

$$\frac{-x^{269}}{x^{269} - x^{268} - x + 1}$$

We have “proved”

$$\sum_{n=1}^{\infty} \lfloor n \tanh(\pi) \rfloor x^n = \frac{x^2}{(x-1)^2} - \frac{x^{269}}{(1-x)(1-x^{268})} + \dots$$

3.2 Interpreting $\mathbb{K}[x]^{m \times m}$ as $\mathbb{K}(x)^{m \times m}$ is to be avoided in algorithms

A too naive application of Gaussian elimination can be exponential-time.

```
[251]: reset()
field = GF(9001)
xring.<x> = PolynomialRing(field)

def one_step_naive(mat, k):
    # one step: put zeroes in first column of mat[k:, k:]
    piv = mat[k,k]
    assert piv != 0, f"non generic pivots {k}"

    for i in range(k+1, mat.nrows()):
        mat[i,k:] = piv * mat[i,k:] - mat[i,k] * mat[k,k:]

    return

def matrix_size(mat):
    return m*m + sum(mat[i,j].degree() for j in range(mat.ncols())
                      for i in range(mat.nrows()) if mat[i,j] != 0)

m = 15
d = 4

mat = matrix.random(xring, m, m, degree=d)
sz0 = matrix_size(mat)
sz = sz0

print("      | ----- NAIVE GAUSS ----- \t|")
print("step  |\tdeg\tsize\tratio\tratio0\t|")
print("-----|-----|")
print(f"input  |\t{mat.degree()}\t{sz}\tnan\t1.00\t|")

for k in range(m-1):
    one_step_naive(mat, k)
    #print(mat.degree_matrix())
    newsz = matrix_size(mat)
    print(f"{k:<6}|\t{mat.degree()}\t{newsz}\t{RR(newsz/sz):.2e}\t{RR(newsz/
↪sz0):.2e}\t|")
    sz = newsz
```

----- NAIVE GAUSS -----				
step	deg	size	ratio	ratio0
input	4	1125	nan	1.00
0	8	1853	1.65e+0	1.65e+0
1	16	3101	1.67e+0	2.76e+0
2	32	5213	1.68e+0	4.63e+0
3	64	8733	1.68e+0	7.76e+0
4	128	14493	1.66e+0	1.29e+1
5	256	23709	1.64e+0	2.11e+1
6	512	38045	1.60e+0	3.38e+1
7	1024	59549	1.57e+0	5.29e+1
8	2048	90269	1.52e+0	8.02e+1
9	4096	131229	1.45e+0	1.17e+2
10	8192	180381	1.37e+0	1.60e+2
11	16384	229533	1.27e+0	2.04e+2
12	32768	262301	1.14e+0	2.33e+2
13	65536	262301	1.00e+0	2.33e+2

[252]: `show(mat.degree_matrix())`

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
-1	8	8	8	8	8	8	8	8	8	8	8	8	8	8
-1	-1	16	16	16	16	16	16	16	16	16	16	16	16	16
-1	-1	-1	32	32	32	32	32	32	32	32	32	32	32	32
-1	-1	-1	-1	64	64	64	64	64	64	64	64	64	64	64
-1	-1	-1	-1	-1	128	128	128	128	128	128	128	128	128	128
-1	-1	-1	-1	-1	-1	256	256	256	256	256	256	256	256	256
-1	-1	-1	-1	-1	-1	-1	512	512	512	512	512	512	512	512
-1	-1	-1	-1	-1	-1	-1	-1	1024	1024	1024	1024	1024	1024	1024
-1	-1	-1	-1	-1	-1	-1	-1	-1	2048	2048	2048	2048	2048	2048
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	4096	4096	4096	4096	4096
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	8192	8192	8192	8192
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	16384	16384	16384
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	32768	32768
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	65536

3.3 polynomial matrices: 2-parameter complexity breaks some habits

[446]: `reset()
p = next_prime(2**59)
field = GF(p)
xring.<x> = PolynomialRing(field)`

3.3.1 multiplying matrices of large size and degree 1? Karatsuba!

Typical situation: characteristic polynomial $\det(xI_m - M)$

```
[259]: m = 400

A0, A1, B0, B1 = [matrix.random(field, m, m) for i in range(4)]
print("3 x 3 leading submatrix of A0:")
show(A0[:3,:3])
show(A0.parent())
print(type(A0), end="\n\n")

A = A0 + A1 * x
print("3 x 3 leading submatrix of A = A0 + A1 * x :")
show(A[:3,:3])
show(A.parent())
print(type(A))
```

3 x 3 leading submatrix of A0:

$$\begin{pmatrix} 354919409288517271 & 438386438904076799 & 29283386713267301 \\ 234111424421071536 & 221372218310176618 & 474458805488806805 \\ 465318241931580570 & 40727308044100888 & 268307888640744216 \end{pmatrix}$$

Mat_{400×400}($\mathbf{F}_{576460752303423619}$)

<class 'sage.matrix.matrix_modn_dense_flint.Matrix_modn_dense_flint'>

3 x 3 leading submatrix of A = A0 + A1 * x :

$$\begin{pmatrix} 90467362389861396x + 354919409288517271 & 271558641008273262x + 438386438904076799 & 286830833191 \\ 298666612735567626x + 234111424421071536 & 138424640407645185x + 221372218310176618 & 1417301035298 \\ 42608297526545244x + 465318241931580570 & 486695253702870707x + 40727308044100888 & 5501141433687 \end{pmatrix}$$

Mat_{400×400}($\mathbf{F}_{576460752303423619}[x]$)

<class 'sage.matrix.matrix_polynomial_dense.Matrix_polynomial_dense'>

```
[105]: %%timeit -n 7 -r 2
# naive multiplication
C0 = A0 * B0
C1 = A0 * B1 + A1 * B0
C2 = A1 * B1
```

96.1 ms ± 1.05 ms per loop (mean ± std. dev. of 2 runs, 7 loops each)

```
[106]: %%timeit -n 7 -r 2
# Karatsuba multiplication
D0 = A0 * B0
D2 = A1 * B1
D1 = (A0 + A1) * (B0 + B1) - D0 - D2
```

71.9 ms ± 462 s per loop (mean ± std. dev. of 2 runs, 7 loops each)

```
[59]: # let's see the impact of well-implemented matrix multiplication...
%%timeit -n 2 -r 2      A0.__mul__(B0)
%%timeit -n 1 -r 1      A0.multiply_classical(B0)
```

24.4 ms $\pm$ 694 s per loop (mean $\pm$ std. dev. of 2 runs, 2 loops each)

17.4 s $\pm$ 0 ns per loop (mean $\pm$ std. dev. of 1 run, 1 loop each)

3.3.2 multiplying matrices of large degree and size 2 x 2? Strassen!

typical situation: extended GCD algorithm $(a, b) \mapsto (g, u, v)$ with $g = \gcd(a, b) = au + bv$

```
[447]: d = 200
a00, a01, a10, a11, b00, b01, b10, b11 = [xring.random_element(degree=d) for i in range(8)]
show(a00 % x**5)
show(a00.parent())
print(type(a00))
```

$497227397496297429x^4 + 183161842033187952x^3 + 272007267521470225x^2 + 132225293342972203x + 209675301562104566$

$\mathbb{F}_{576460752303423619}[x]$

<class 'sage.rings.polynomial.polynomial_zmod_flint.Polynomial_zmod_flint'>

```
[448]: %%timeit -n 10 -r 1000
# naive multiplication
c00 = a00 * b00 + a01 * b10
c01 = a00 * b01 + a01 * b11
c10 = a10 * b00 + a11 * b10
c11 = a10 * b01 + a11 * b11
```

113 s $\pm$ 6.08 s per loop (mean $\pm$ std. dev. of 1000 runs, 10 loops each)

```
[449]: %%timeit -n 10 -r 1000
# Strassen multiplication
m1 = (a00 + a11) * (b00 + b11)
m2 = (a10 + a11) * b00
m3 = a00 * (b01 - b11)
m4 = a11 * (b10 - b00)
m5 = (a00 + a01) * b11
m6 = (a10 - a00) * (b00 + b01)
m7 = (a01 - a11) * (b10 + b11)

d00 = m1 + m4 - m5 + m7
d01 = m3 + m5
d10 = m2 + m4
d11 = m1 - m2 + m3 + m6
```

98.8 s $\pm$ 10.1 s per loop (mean $\pm$ std. dev. of 1000 runs, 10 loops each)

```
[450]: print(98.8/113, 7/8.)
```

```
0.874336283185841 0.8750000000000000
```

3.4 choice of data representation matters

```
[269]: reset()
```

```
def generate_mat(field, m, d):  
    # polynomial matrix: matrix filled with polynomial entries  
    xring.<x> = PolynomialRing(field)  
    poly_mat = matrix.random(xring, m, m, degree=d-1)  
    # matrix polynomial: polynomial whose coefficients are matrices  
    matspace.<y> = PolynomialRing(MatrixSpace(field, m, m))  
    mat_poly = matspace.random_element(degree=d-1)  
    return poly_mat, mat_poly
```

```
[270]: poly_mat, mat_poly = generate_mat(GF(97), m=3, d=4)  
show(poly_mat.parent())  
show(mat_poly.parent())
```

$\text{Mat}_{3 \times 3}(\mathbf{F}_{97}[x])$

$\text{Mat}_{3 \times 3}(\mathbf{F}_{97})[y]$

```
[173]: poly_mat1, mat_poly1 = generate_mat(GF(97), m=2, d=1000)  
poly_mat2, mat_poly2 = generate_mat(GF(97), m=2, d=1000)  
%timeit poly_mat1 * poly_mat2  
%timeit mat_poly1 * mat_poly2
```

```
155 s ± 1.67 s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)  
133 ms ± 1.45 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
[174]: poly_mat1, mat_poly1 = generate_mat(GF(97), m=100, d=2)  
poly_mat2, mat_poly2 = generate_mat(GF(97), m=100, d=2)  
%timeit poly_mat1 * poly_mat2  
%timeit mat_poly1 * mat_poly2
```

```
160 ms ± 594 s per loop (mean ± std. dev. of 7 runs, 10 loops each)  
124 s ± 1.5 s per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Conclusions - choosing the right data structure matters - array of matrices is generally best for large size / low degree - array of polynomials is generally best for small size / large degree - more heterogeneous degrees $\Rightarrow$ more options to consider...

Note: same conclusions (although with more reasonable running time discrepancies) are drawn from more low-level and optimized implementations

3.5 division with remainder for polynomial matrices

```
[272]: reset()
xring.<x> = PolynomialRing(GF(997))
m = 8
d = 100
B = matrix.random(xring, m, m, degree=d)
while not B.leading_matrix().is_invertible():
    B = matrix.random(xring, m, m, degree=d)
```

```
[274]: u = matrix.random(xring, 1, m, degree=d*d)
v = matrix.random(xring, 1, m, degree=d*d) * B
w = v + matrix([[x+1, 0, 3*x^2 + x, 5*x, 3, x**10, 7*x**2 + 2, x]])

print("degrees in B:")
show(B.degree_matrix())

for vec in [u, v, w]:
    (Q, R) = vec.right_quo_rem(B)
    print(f"vec belongs to <B>? --> {R == 0}")
    print(f"degrees in vec: {vec.degree_matrix()}")
    print(f"degrees in vec mod B: {R.degree_matrix()}")
    print()

show(R)
```

degrees in B:

$$\begin{pmatrix} 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \end{pmatrix}$$

vec belongs to ? --> False

degrees in vec: [10000 10000 10000 10000 10000 10000 10000 10000]

degrees in vec mod B: [99 99 99 99 99 99 99 99]

vec belongs to ? --> True

degrees in vec: [10100 10100 10100 10100 10100 10100 10100 10100]

degrees in vec mod B: [-1 -1 -1 -1 -1 -1 -1 -1]

vec belongs to ? --> False

degrees in vec: [10100 10100 10100 10100 10100 10100 10100 10100]

degrees in vec mod B: [1 -1 2 1 0 10 2 1]

$$\begin{pmatrix} x+1 & 0 & 3x^2+x & 5x & 3 & x^{10} & 7x^2+2 & x \end{pmatrix}$$

```
[276]: B = matrix([[x**2, 0], [0, x**3 - 1]])
v = matrix.random(xring, 1, 2, degree=5)
print("input matrix:")
show(B)
print("input vector:")
show(v)
(Q,R) = v.right_quo_rem(B) # !! works !!
print("quotient:")
show(Q)
print("remainder:")
show(R)
```

input matrix:

$$\begin{pmatrix} x^2 & 0 \\ 0 & x^3 + 996 \end{pmatrix}$$

input vector:

$$\begin{pmatrix} 125x^5 + 85x^4 + 30x^3 + 519x^2 + 394x + 837 & 31x^5 + 441x^4 + 529x^3 + 469x^2 + 871x + 146 \end{pmatrix}$$

quotient:

$$\begin{pmatrix} 125x^3 + 85x^2 + 30x + 519 & 31x^2 + 441x + 529 \end{pmatrix}$$

remainder:

$$\begin{pmatrix} 394x + 837 & 500x^2 + 315x + 675 \end{pmatrix}$$

this is a case where matrix division with remainder is well-defined, yet this is not covered by the algorithm in the slides, due to the assumption “ $\text{lc}(B)$ is invertible”...

3.6 Computing an s-minimal basis of vector approximants/interpolants

Input: polynomials $f_1, \dots, f_m \in \mathbb{K}[x]_{<d}$

Input: polynomial $M = (x - \alpha_1) \cdots (x - \alpha_d)$

Input: integer shifts $s = (s_1, \dots, s_m)$

Output: s-minimal basis of $\{[p_1 \ \cdots \ p_m] \in \mathbb{K}[x]^{1 \times m} \mid p_1 f_1 + \cdots + p_m f_m = 0 \bmod M\}$

Documentation: - [Matrix_polynomial_dense.minimal_approximant_basis](#) - [Matrix_polynomial_dense.minimal_interpolant_basis](#) - [Matrix_polynomial_dense.minimal_relation_basis](#)

```
[457]: reset()
field = GF(97)
xring.<x> = PolynomialRing(field)

d = 8
m = 4
```

```

points = [24, 31, 15, 32, 83, 27, 20, 59]
M = prod(x - pt for pt in points)
print("Modulus polynomial:")
show(M)

values = [80, 73, 73, 35, 66, 46, 91, 64]
f = xring.lagrange_polynomial(zip(points, values))
f1 = xring.one()
f2 = f
f3 = f**2 % M
f4 = f**3 % M

print("Input polynomials:")
show(f1)
show(f2)
show(f3)
show(f4)

```

Modulus polynomial:

$$x^8 + 84x^6 + 11x^5 + 67x^4 + 71x^3 + 26x^2 + 24x + 89$$

Input polynomials:

1

$$15x^7 + 24x^6 + 96x^5 + 59x^4 + x^3 + 43x^2 + 17x + 84$$

$$38x^7 + 65x^6 + 60x^5 + 10x^4 + 23x^3 + 44x^2 + 84x + 49$$

$$x^7 + 13x^6 + 89x^5 + 27x^4 + 24x^3 + 57x^2 + 54x + 75$$

```

[458]: F = matrix([[1], [f], [f**2 % M], [f**3 % M]])
P = F.minimal_interpolant_basis(points)
print("minimal basis of M-Padé approximants / vector interpolants:")
show(P)
print("minimal      -->", P.is_reduced())
print("Popov        -->", P.is_popov())
print("P*F = 0 mod M -->", (P * F) % M == 0)
print("generating set -->", P.determinant() == M)

```

minimal basis of M-Padé approximants / vector interpolants:

$$\begin{pmatrix} x^2 + x + 38 & 83x + 50 & 94x + 25 & 41x + 44 \\ 11x^2 + 72x & x^2 + 30x + 10 & 24x + 53 & 11x + 69 \\ 9x^2 + 9x + 14 & 64x^2 + 62x + 45 & x^2 + 87x + 64 & 57x + 95 \\ 92x^2 + 61x + 58 & 28x^2 + 40x + 81 & 89x^2 + 12x + 52 & x^2 + 64x + 22 \end{pmatrix}$$

```

minimal      --> True
Popov        --> False
P*F = 0 mod M --> True
generating set --> True

```

```
[459]: # this provides a low x-degree bivariate interpolant
xring.<Y> = PolynomialRing(xring)
Q = P[0,0] + P[0,1] * Y + P[0,2] * Y**2 + P[0,3] * Y**3
show(Q)
evals = [Q(values[i], points[i]) for i in range(d)]
show(evals)
```

$$(41x + 44)Y^3 + (94x + 25)Y^2 + (83x + 50)Y + x^2 + x + 38$$

[0, 0, 0, 0, 0, 0, 0]

```
[460]: s = [0, 2, 4, 6]
P = F.minimal_interpolant_basis(points, shifts=s)
print(f"{s}-minimal basis of M-Pad  approximants / vector interpolants:")
show(P)
print("minimal      -->", P.is_reduced())
print("s-minimal    -->", P.is_reduced(shifts=s))
print("s-Popov      -->", P.is_popov(shifts=s))
print("P*F = 0 mod M -->", (P * F) % M == 0)
print("generating set -->", P.determinant() == M)
```

[0, 2, 4, 6]-minimal basis of M-Pad  approximants / vector interpolants:

$$\begin{pmatrix} x^5 + 12x^4 + 10x^3 + 34x^2 + 65x + 2 & 60x^2 + 43x + 67 & 0 & 0 \\ 48x^5 + 50x^4 + 59x^3 + 62x^2 + 32x + 39 & x^3 + 55x^2 + 16x + 90 & 0 & 0 \\ 2x^4 + 56x^3 + 42x^2 + 48x + 15 & 72x^2 + 12x + 30 & 1 & 0 \\ 40x^4 + 19x^3 + 14x^2 + 40x + 49 & 53x^2 + 79x + 74 & 0 & 1 \end{pmatrix}$$

```
minimal      --> False
s-minimal    --> True
s-Popov      --> False
P*F = 0 mod M --> True
generating set --> True
```

3.7 Base case vector M-Pad : one point

This is essentially Gaussian elimination on the constant matrix $F(\alpha) \in \mathbb{K}^{m \times 1}$.

```
[461]: show(F.minimal_interpolant_basis([points[0]]))
show(F.minimal_interpolant_basis([points[0]], shifts=[2, 1, 0, 2]))
```

$$\begin{pmatrix} x + 73 & 0 & 0 & 0 \\ 17 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 63 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 49 & 0 \\ 0 & 1 & 40 & 0 \\ 0 & 0 & x + 73 & 0 \\ 0 & 0 & 17 & 1 \end{pmatrix}$$

3.8 Beckermann and Labahn's divide and conquer approach

1. Run two recursive calls with half as many points
2. Combine result via polynomial matrix multiplication

second call depends on the output of the first! (through the “residual”)
notorious obstacle for parallelization

```
[462]: # split into subproblems
d1 = floor(d / 2)
points1 = points[0:d1]
points2 = points[d1:d]

# first recursive call
P1 = F.minimal_interpolant_basis(points1, shifts=s)
# residual (simplified)
G = P1 * F
# updated shift
t = P1.row_degrees(shifts=s)
# second recursive call
P2 = G.minimal_interpolant_basis(points2, shifts=t)
# multiply
P = P2 * P1
```

```
[463]: print("s-minimal      -->", P.is_reduced(shifts=s))
print("s-Popov        -->", P.is_popov(shifts=s))
print("P*F = 0 mod M  -->", (P * F) % M == 0)
print("generating set -->", P.determinant() == M)
```

```
s-minimal      --> True
s-Popov        --> False
P*F = 0 mod M  --> True
generating set --> True
```

```
[464]: P_test = F.minimal_interpolant_basis(points, shifts=s)
print("exact same basis  -->", P == P_test)
print("confirm same module -->", P.popov_form() == P_test.popov_form())
```

```
exact same basis  --> False
confirm same module --> True
```

```
[465]: show(P)
print()
show(P_test)
P.add_multiple_of_row(1, 0, 48 - 82)
P == P_test
```

$$\begin{pmatrix} x^5 + 12x^4 + 10x^3 + 34x^2 + 65x + 2 & 60x^2 + 43x + 67 & 0 & 0 \\ 82x^5 + 70x^4 + 11x^3 + 54x^2 + 11x + 10 & x^3 + 58x^2 + 23x + 40 & 0 & 0 \\ 2x^4 + 56x^3 + 42x^2 + 48x + 15 & 72x^2 + 12x + 30 & 1 & 0 \\ 40x^4 + 19x^3 + 14x^2 + 40x + 49 & 53x^2 + 79x + 74 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x^5 + 12x^4 + 10x^3 + 34x^2 + 65x + 2 & 60x^2 + 43x + 67 & 0 & 0 \\ 48x^5 + 50x^4 + 59x^3 + 62x^2 + 32x + 39 & x^3 + 55x^2 + 16x + 90 & 0 & 0 \\ 2x^4 + 56x^3 + 42x^2 + 48x + 15 & 72x^2 + 12x + 30 & 1 & 0 \\ 40x^4 + 19x^3 + 14x^2 + 40x + 49 & 53x^2 + 79x + 74 & 0 & 1 \end{pmatrix}$$

[465]: True

3.9 Worst-case size: why targeting Popov bases matters

We take a random polynomial $R \in \mathbb{K}[x]_{<d}$ and consider Hermite-Padé approximation with specially chosen polynomials. - The obtained basis has very unbalanced row degrees, even though we start from the uniform shift $s = (0, \dots, 0)$ - Augmenting the example, we get an instance with output size $O(m^2d)$ - Yet, the canonical s -Popov basis always has size $\leq md$

```
[466]: reset()
field = GF(97)
xring.<x> = PolynomialRing(field)

m = 4
d = 128
f = xring.random_element(degree=d-1)
f1 = f
f2 = ((1+x) * f).truncate(d)
f3 = (x * (1+x) * f).truncate(d)
f4 = (x * x * (1+x) * f).truncate(d)

print("Input polynomials:")
show(f1)
show(f2)
show(f3)
show(f4)
```

Input polynomials:

```
44x127 + 73x126 + 60x125 + 51x124 + 23x123 + 75x122 + 35x121 + 50x120 + 17x119 + 53x118 + 7x117 +
14x116 + 23x115 + 7x114 + 12x113 + 55x112 + 59x111 + 8x110 + 71x109 + 61x108 + 61x107 + 43x106 +
19x105 + 77x104 + 59x103 + 88x102 + 5x101 + 95x100 + 25x99 + 51x98 + 51x97 + 43x96 + 46x95 + 76x94 +
21x93 + 74x92 + 60x91 + 38x90 + 37x89 + 83x88 + 10x87 + 33x86 + 90x85 + 89x84 + 30x83 + 60x82 + 53x81 +
12x80 + 43x79 + 54x78 + 70x77 + 73x76 + 40x75 + 71x74 + x72 + 21x71 + 46x70 + 8x69 + 11x68 + 47x67 +
28x66 + 74x65 + 29x64 + 62x63 + 89x62 + 31x61 + 9x60 + 35x59 + 65x58 + 63x57 + 15x56 + 68x55 + 93x54 +
59x53 + 35x52 + 95x51 + 84x50 + 90x49 + 90x48 + 48x47 + 8x46 + 71x45 + 15x44 + 5x43 + 13x42 + 86x41 +
39x40 + 4x39 + 39x38 + 91x37 + 41x36 + 57x35 + 45x34 + 64x33 + 28x32 + 28x31 + 25x30 + 47x29 + 89x28 +
76x27 + 13x26 + 70x25 + 88x24 + 64x23 + 17x22 + 53x21 + 83x20 + 55x19 + 40x18 + 40x17 + 50x16 +
```

$$\begin{aligned}
&60x^{15} + 7x^{14} + 12x^{13} + 36x^{12} + 18x^{11} + 11x^{10} + 57x^9 + 23x^8 + 67x^7 + 81x^6 + 15x^5 + x^4 + 69x^3 + 14x^2 + 79 \\
&20x^{127} + 36x^{126} + 14x^{125} + 74x^{124} + x^{123} + 13x^{122} + 85x^{121} + 67x^{120} + 70x^{119} + 60x^{118} + 21x^{117} + \\
&37x^{116} + 30x^{115} + 19x^{114} + 67x^{113} + 17x^{112} + 67x^{111} + 79x^{110} + 35x^{109} + 25x^{108} + 7x^{107} + 62x^{106} + \\
&96x^{105} + 39x^{104} + 50x^{103} + 93x^{102} + 3x^{101} + 23x^{100} + 76x^{99} + 5x^{98} + 94x^{97} + 89x^{96} + 25x^{95} + 95x^{94} + \\
&37x^{93} + x^{92} + 75x^{91} + 23x^{90} + 93x^{89} + 43x^{88} + 26x^{87} + 82x^{86} + 22x^{85} + 90x^{84} + 16x^{83} + 65x^{82} + 55x^{81} + 27x^{80} + \\
&27x^{78} + 46x^{77} + 16x^{76} + 14x^{75} + 71x^{74} + x^{73} + 22x^{72} + 67x^{71} + 54x^{70} + 19x^{69} + 58x^{68} + 75x^{67} + 5x^{66} + \\
&6x^{65} + 91x^{64} + 54x^{63} + 23x^{62} + 40x^{61} + 44x^{60} + 3x^{59} + 31x^{58} + 78x^{57} + 83x^{56} + 64x^{55} + 55x^{54} + 94x^{53} + \\
&33x^{52} + 82x^{51} + 77x^{50} + 83x^{49} + 41x^{48} + 56x^{47} + 79x^{46} + 86x^{45} + 20x^{44} + 18x^{43} + 2x^{42} + 28x^{41} + 43x^{40} + \\
&43x^{39} + 33x^{38} + 35x^{37} + x^{36} + 5x^{35} + 12x^{34} + 92x^{33} + 56x^{32} + 53x^{31} + 72x^{30} + 39x^{29} + 68x^{28} + 89x^{27} + \\
&83x^{26} + 61x^{25} + 55x^{24} + 81x^{23} + 70x^{22} + 39x^{21} + 41x^{20} + 95x^{19} + 80x^{18} + 90x^{17} + 13x^{16} + 67x^{15} + 19x^{14} + \\
&48x^{13} + 54x^{12} + 29x^{11} + 68x^{10} + 80x^9 + 90x^8 + 51x^7 + 96x^6 + 16x^5 + 70x^4 + 83x^3 + 14x^2 + 79x + 79 \\
&36x^{127} + 14x^{126} + 74x^{125} + x^{124} + 13x^{123} + 85x^{122} + 67x^{121} + 70x^{120} + 60x^{119} + 21x^{118} + 37x^{117} + \\
&30x^{116} + 19x^{115} + 67x^{114} + 17x^{113} + 67x^{112} + 79x^{111} + 35x^{110} + 25x^{109} + 7x^{108} + 62x^{107} + 96x^{106} + \\
&39x^{105} + 50x^{104} + 93x^{103} + 3x^{102} + 23x^{101} + 76x^{100} + 5x^{99} + 94x^{98} + 89x^{97} + 25x^{96} + 95x^{94} + 37x^{93} + \\
&x^{92} + 75x^{91} + 23x^{90} + 93x^{89} + 43x^{88} + 26x^{87} + 82x^{86} + 22x^{85} + 90x^{84} + 16x^{83} + 65x^{82} + 55x^{81} + 27x^{79} + \\
&46x^{78} + 16x^{77} + 14x^{76} + 71x^{75} + x^{74} + 22x^{73} + 67x^{72} + 54x^{71} + 19x^{70} + 58x^{69} + 75x^{68} + 5x^{67} + 6x^{66} + \\
&91x^{65} + 54x^{64} + 23x^{63} + 40x^{62} + 44x^{61} + 3x^{60} + 31x^{59} + 78x^{58} + 83x^{57} + 64x^{56} + 55x^{55} + 94x^{54} + 33x^{53} + \\
&82x^{52} + 77x^{51} + 83x^{50} + 41x^{49} + 56x^{48} + 79x^{47} + 86x^{46} + 20x^{45} + 18x^{44} + 2x^{43} + 28x^{42} + 43x^{41} + 43x^{40} + \\
&33x^{39} + 35x^{38} + x^{37} + 5x^{36} + 12x^{35} + 92x^{34} + 56x^{33} + 53x^{32} + 72x^{31} + 39x^{30} + 68x^{29} + 89x^{28} + 83x^{27} + \\
&61x^{26} + 55x^{25} + 81x^{24} + 70x^{23} + 39x^{22} + 41x^{21} + 95x^{20} + 80x^{19} + 90x^{18} + 13x^{17} + 67x^{16} + 19x^{15} + \\
&48x^{14} + 54x^{13} + 29x^{12} + 68x^{11} + 80x^{10} + 90x^9 + 51x^8 + 96x^7 + 16x^6 + 70x^5 + 83x^4 + 14x^3 + 79x^2 + 79x \\
&14x^{127} + 74x^{126} + x^{125} + 13x^{124} + 85x^{123} + 67x^{122} + 70x^{121} + 60x^{120} + 21x^{119} + 37x^{118} + 30x^{117} + \\
&19x^{116} + 67x^{115} + 17x^{114} + 67x^{113} + 79x^{112} + 35x^{111} + 25x^{110} + 7x^{109} + 62x^{108} + 96x^{107} + 39x^{106} + \\
&50x^{105} + 93x^{104} + 3x^{103} + 23x^{102} + 76x^{101} + 5x^{100} + 94x^{99} + 89x^{98} + 25x^{97} + 95x^{95} + 37x^{94} + x^{93} + \\
&75x^{92} + 23x^{91} + 93x^{90} + 43x^{89} + 26x^{88} + 82x^{87} + 22x^{86} + 90x^{85} + 16x^{84} + 65x^{83} + 55x^{82} + 27x^{80} + \\
&46x^{79} + 16x^{78} + 14x^{77} + 71x^{76} + x^{75} + 22x^{74} + 67x^{73} + 54x^{72} + 19x^{71} + 58x^{70} + 75x^{69} + 5x^{68} + 6x^{67} + \\
&91x^{66} + 54x^{65} + 23x^{64} + 40x^{63} + 44x^{62} + 3x^{61} + 31x^{60} + 78x^{59} + 83x^{58} + 64x^{57} + 55x^{56} + 94x^{55} + 33x^{54} + \\
&82x^{53} + 77x^{52} + 83x^{51} + 41x^{50} + 56x^{49} + 79x^{48} + 86x^{47} + 20x^{46} + 18x^{45} + 2x^{44} + 28x^{43} + 43x^{42} + 43x^{41} + \\
&33x^{40} + 35x^{39} + x^{38} + 5x^{37} + 12x^{36} + 92x^{35} + 56x^{34} + 53x^{33} + 72x^{32} + 39x^{31} + 68x^{30} + 89x^{29} + 83x^{28} + \\
&61x^{27} + 55x^{26} + 81x^{25} + 70x^{24} + 39x^{23} + 41x^{22} + 95x^{21} + 80x^{20} + 90x^{19} + 13x^{18} + 67x^{17} + 19x^{16} + \\
&48x^{15} + 54x^{14} + 29x^{13} + 68x^{12} + 80x^{11} + 90x^{10} + 51x^9 + 96x^8 + 16x^7 + 70x^6 + 83x^5 + 14x^4 + 79x^3 + 79x^2
\end{aligned}$$

```

[467]: F = matrix([[f1], [f2], [f3], [f4]])
P = F.minimal_approximant_basis(d)
P_popov = F.minimal_approximant_basis(d, normal_form=True)
print("minimal basis of Hermite-Pad  approximants,"
      " and the corresponding Popov basis:")
show(P)
show(P_popov)

print("\n", "their degree matrices:")
show(P.degree_matrix(), P_popov.degree_matrix())

```

minimal basis of Hermite-Pad  approximants, and the corresponding Popov basis:

$$\begin{pmatrix} x+1 & 96 & 0 & 0 \\ 96x+96 & x+1 & 96 & 0 \\ x+1 & 96x+96 & x+1 & 96 \\ 96x^{125} & x^{125} & 96x^{125} & x^{125} \end{pmatrix}$$

$$\begin{pmatrix} x+1 & 96 & 0 \\ 0 & x & 96 \\ 0 & 0 & x \\ 1 & 96 & 1 & x^{125} + 96x^{124} + x^{123} + 96x^{122} + x^{121} + 96x^{120} + x^{119} + 96x^{118} + x^{117} + 96x^{116} + x^{115} + 96x^{114} \end{pmatrix}$$

their degree matrices:

$$\begin{pmatrix} 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \\ 125 & 125 & 125 & 125 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 & -1 \\ -1 & 1 & 0 & -1 \\ -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 125 \end{pmatrix}$$

```
[468]: m = 8

f5 = xring.random_element(degree=d-1)
f6 = xring.random_element(degree=d-1)
f7 = xring.random_element(degree=d-1)
f8 = xring.random_element(degree=d-1)
F = matrix([[f1], [f2], [f3], [f4], [f5], [f6], [f7], [f8]])

s = [0,0,0,0,d,d,d,d]
P = F.minimal_approximant_basis(d, shifts=s)
P_popov = F.minimal_approximant_basis(d, shifts=s, normal_form=True)

print("minimal basis of Hermite-Pad  approximants,"
      " and the corresponding Popov basis:")
show(P)
show(P_popov)
print("\n", "their degree matrices:")
show(P.degree_matrix(), P_popov.degree_matrix())
```

minimal basis of Hermite-Pad  approximants, and the corresponding Popov basis:

$$\begin{pmatrix} 80x^{124} + 33x^{123} + 14x^{122} + 20x^{121} + 77x^{120} + 34x^{119} + 25x^{118} + 74x^{117} + 26x^{116} + 15x^{115} + 84x^{114} + 5x^{113} + \dots \\ 20x^{124} + 86x^{123} + 60x^{122} + 64x^{121} + 66x^{120} + 73x^{119} + 21x^{118} + 8x^{117} + 23x^{116} + 45x^{115} + 20x^{114} + 78x^{113} + \dots \\ 73x^{124} + 68x^{123} + 8x^{122} + 33x^{121} + 27x^{120} + 67x^{119} + 92x^{118} + 29x^{117} + 82x^{116} + 61x^{115} + 83x^{114} + \dots \\ 8x^{124} + 57x^{123} + 93x^{122} + 55x^{121} + 40x^{120} + 35x^{119} + 25x^{118} + 63x^{117} + 71x^{116} + 76x^{115} + 92x^{114} + \dots \end{pmatrix}$$

$$\begin{pmatrix} x+1 & 96 & 0 \\ 0 & x & 96 \\ 0 & 0 & x \\ 1 & 96 & 1 \\ 25 & 45 & 70 & 17x^{124} + 47x^{123} + 36x^{122} + 41x^{121} + 76x^{120} + 84x^{119} + 85x^{118} + 35x^{117} + 36x^{116} + 46x^{115} + 64x^{114} + 47x^{113} + 36x^{112} + 41x^{111} + 76x^{110} + 84x^{109} + 85x^{108} + 35x^{107} + 36x^{106} + 46x^{105} + 64x^{104} + 47x^{103} + 36x^{102} + 41x^{101} + 76x^{100} + 84x^{99} + 85x^{98} + 35x^{97} + 36x^{96} + 46x^{95} + 64x^{94} + 47x^{93} + 36x^{92} + 41x^{91} + 76x^{90} + 84x^{89} + 85x^{88} + 35x^{87} + 36x^{86} + 46x^{85} + 64x^{84} + 47x^{83} + 36x^{82} + 41x^{81} + 76x^{80} + 84x^{79} + 85x^{78} + 35x^{77} + 36x^{76} + 46x^{75} + 64x^{74} + 47x^{73} + 36x^{72} + 41x^{71} + 76x^{70} + 84x^{69} + 85x^{68} + 35x^{67} + 36x^{66} + 46x^{65} + 64x^{64} + 47x^{63} + 36x^{62} + 41x^{61} + 76x^{60} + 84x^{59} + 85x^{58} + 35x^{57} + 36x^{56} + 46x^{55} + 64x^{54} + 47x^{53} + 36x^{52} + 41x^{51} + 76x^{50} + 84x^{49} + 85x^{48} + 35x^{47} + 36x^{46} + 46x^{45} + 64x^{44} + 47x^{43} + 36x^{42} + 41x^{41} + 76x^{40} + 84x^{39} + 85x^{38} + 35x^{37} + 36x^{36} + 46x^{35} + 64x^{34} + 47x^{33} + 36x^{32} + 41x^{31} + 76x^{30} + 84x^{29} + 85x^{28} + 35x^{27} + 36x^{26} + 46x^{25} + 64x^{24} + 47x^{23} + 36x^{22} + 41x^{21} + 76x^{20} + 84x^{19} + 85x^{18} + 35x^{17} + 36x^{16} + 46x^{15} + 64x^{14} + 47x^{13} + 36x^{12} + 41x^{11} + 76x^{10} + 84x^9 + 85x^8 + 35x^7 + 36x^6 + 46x^5 + 64x^4 + 47x^3 + 36x^2 + 41x + 76 \\ 51 & 10 & 95 & 77x^{124} + 31x^{123} + 6x^{122} + 27x^{121} + 4x^{120} + 20x^{119} + 56x^{118} + 33x^{117} + 41x^{116} + 64x^{115} + 47x^{114} + 36x^{113} + 41x^{112} + 76x^{111} + 84x^{110} + 85x^{109} + 35x^{108} + 36x^{107} + 46x^{106} + 64x^{105} + 47x^{104} + 36x^{103} + 41x^{102} + 76x^{101} + 84x^{100} + 85x^{99} + 35x^{98} + 36x^{97} + 46x^{96} + 64x^{95} + 47x^{94} + 36x^{93} + 41x^{92} + 76x^{91} + 84x^{90} + 85x^{89} + 35x^{88} + 36x^{87} + 46x^{86} + 64x^{85} + 47x^{84} + 36x^{83} + 41x^{82} + 76x^{81} + 84x^{80} + 85x^{79} + 35x^{78} + 36x^{77} + 46x^{76} + 64x^{75} + 47x^{74} + 36x^{73} + 41x^{72} + 76x^{71} + 84x^{70} + 85x^{69} + 35x^{68} + 36x^{67} + 46x^{66} + 64x^{65} + 47x^{64} + 36x^{63} + 41x^{62} + 76x^{61} + 84x^{60} + 85x^{59} + 35x^{58} + 36x^{57} + 46x^{56} + 64x^{55} + 47x^{54} + 36x^{53} + 41x^{52} + 76x^{51} + 84x^{50} + 85x^{49} + 35x^{48} + 36x^{47} + 46x^{46} + 64x^{45} + 47x^{44} + 36x^{43} + 41x^{42} + 76x^{41} + 84x^{40} + 85x^{39} + 35x^{38} + 36x^{37} + 46x^{36} + 64x^{35} + 47x^{34} + 36x^{33} + 41x^{32} + 76x^{31} + 84x^{30} + 85x^{29} + 35x^{28} + 36x^{27} + 46x^{26} + 64x^{25} + 47x^{24} + 36x^{23} + 41x^{22} + 76x^{21} + 84x^{20} + 85x^{19} + 35x^{18} + 36x^{17} + 46x^{16} + 64x^{15} + 47x^{14} + 36x^{13} + 41x^{12} + 76x^{11} + 84x^{10} + 85x^9 + 35x^8 + 36x^7 + 46x^6 + 64x^5 + 47x^4 + 36x^3 + 41x^2 + 76x + 85 \\ 43 & 47 & 53 & 24x^{124} + 5x^{123} + 84x^{122} + 77x^{121} + 90x^{120} + 37x^{119} + 65x^{118} + 3x^{117} + 12x^{116} + 24x^{115} + 47x^{114} + 36x^{113} + 41x^{112} + 76x^{111} + 84x^{110} + 85x^{109} + 35x^{108} + 36x^{107} + 46x^{106} + 64x^{105} + 47x^{104} + 36x^{103} + 41x^{102} + 76x^{101} + 84x^{100} + 85x^{99} + 35x^{98} + 36x^{97} + 46x^{96} + 64x^{95} + 47x^{94} + 36x^{93} + 41x^{92} + 76x^{91} + 84x^{90} + 85x^{89} + 35x^{88} + 36x^{87} + 46x^{86} + 64x^{85} + 47x^{84} + 36x^{83} + 41x^{82} + 76x^{81} + 84x^{80} + 85x^{79} + 35x^{78} + 36x^{77} + 46x^{76} + 64x^{75} + 47x^{74} + 36x^{73} + 41x^{72} + 76x^{71} + 84x^{70} + 85x^{69} + 35x^{68} + 36x^{67} + 46x^{66} + 64x^{65} + 47x^{64} + 36x^{63} + 41x^{62} + 76x^{61} + 84x^{60} + 85x^{59} + 35x^{58} + 36x^{57} + 46x^{56} + 64x^{55} + 47x^{54} + 36x^{53} + 41x^{52} + 76x^{51} + 84x^{50} + 85x^{49} + 35x^{48} + 36x^{47} + 46x^{46} + 64x^{45} + 47x^{44} + 36x^{43} + 41x^{42} + 76x^{41} + 84x^{40} + 85x^{39} + 35x^{38} + 36x^{37} + 46x^{36} + 64x^{35} + 47x^{34} + 36x^{33} + 41x^{32} + 76x^{31} + 84x^{30} + 85x^{29} + 35x^{28} + 36x^{27} + 46x^{26} + 64x^{25} + 47x^{24} + 36x^{23} + 41x^{22} + 76x^{21} + 84x^{20} + 85x^{19} + 35x^{18} + 36x^{17} + 46x^{16} + 64x^{15} + 47x^{14} + 36x^{13} + 41x^{12} + 76x^{11} + 84x^{10} + 85x^9 + 35x^8 + 36x^7 + 46x^6 + 64x^5 + 47x^4 + 36x^3 + 41x^2 + 76x + 85 \\ 23 & 73 & 21 & 89x^{124} + 48x^{123} + 53x^{122} + 86x^{121} + 68x^{120} + 91x^{119} + 78x^{118} + 53x^{117} + 64x^{116} + 47x^{115} + 36x^{114} + 41x^{113} + 76x^{112} + 84x^{111} + 85x^{110} + 35x^{109} + 36x^{108} + 46x^{107} + 64x^{106} + 47x^{105} + 36x^{104} + 41x^{103} + 76x^{102} + 84x^{101} + 85x^{100} + 35x^{99} + 36x^{98} + 46x^{97} + 64x^{96} + 47x^{95} + 36x^{94} + 41x^{93} + 76x^{92} + 84x^{91} + 85x^{90} + 35x^{89} + 36x^{88} + 46x^{87} + 64x^{86} + 47x^{85} + 36x^{84} + 41x^{83} + 76x^{82} + 84x^{81} + 85x^{80} + 35x^{79} + 36x^{78} + 46x^{77} + 64x^{76} + 47x^{75} + 36x^{74} + 41x^{73} + 76x^{72} + 84x^{71} + 85x^{70} + 35x^{69} + 36x^{68} + 46x^{67} + 64x^{66} + 47x^{65} + 36x^{64} + 41x^{63} + 76x^{62} + 84x^{61} + 85x^{60} + 35x^{59} + 36x^{58} + 46x^{57} + 64x^{56} + 47x^{55} + 36x^{54} + 41x^{53} + 76x^{52} + 84x^{51} + 85x^{50} + 35x^{49} + 36x^{48} + 46x^{47} + 64x^{46} + 47x^{45} + 36x^{44} + 41x^{43} + 76x^{42} + 84x^{41} + 85x^{40} + 35x^{39} + 36x^{38} + 46x^{37} + 64x^{36} + 47x^{35} + 36x^{34} + 41x^{33} + 76x^{32} + 84x^{31} + 85x^{30} + 35x^{29} + 36x^{28} + 46x^{27} + 64x^{26} + 47x^{25} + 36x^{24} + 41x^{23} + 76x^{22} + 84x^{21} + 85x^{20} + 35x^{19} + 36x^{18} + 46x^{17} + 64x^{16} + 47x^{15} + 36x^{14} + 41x^{13} + 76x^{12} + 84x^{11} + 85x^{10} + 35x^9 + 36x^8 + 46x^7 + 64x^6 + 47x^5 + 36x^4 + 41x^3 + 76x^2 + 84x + 85 \end{pmatrix}$$

their degree matrices:

$$\begin{pmatrix} 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 & -1 \\ 125 & 125 & 125 & 125 & -1 & -1 & -1 & -1 \\ 124 & 124 & 124 & 124 & 0 & -1 & -1 & -1 \\ 124 & 124 & 124 & 124 & -1 & 0 & -1 & -1 \\ 124 & 124 & 124 & 124 & -1 & -1 & 0 & -1 \\ 124 & 124 & 124 & 124 & -1 & -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 125 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & -1 & 0 \end{pmatrix}$$

3.10 Exploiting the knowledge of pivot degrees

If the pivot degrees $\delta = (\delta_1, \dots, \delta_m)$ of the s -Popov basis are known: - one can forget about the shift s - rather use the shift $-\delta$ - **any** $-\delta$ -minimal basis has **small size** and is **almost s -Popov**

```
[469]: best_s = [-P[i,i].degree() for i in range(m)] # [1, 1, 1, 125, 0, 0, 0, 0]
B = F.minimal_approximant_basis(d, shifts=best_s)

print("degrees of minimal basis for pivot-degree-based shift:")
show(B.degree_matrix())
print("minimal for modified shift -->", B.is_reduced(shifts=best_s))
print("Popov for modified shift -->", B.is_popov(shifts=best_s))
```

degrees of minimal basis for pivot-degree-based shift:

$$\begin{pmatrix} 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 125 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & -1 & 0 \end{pmatrix}$$

```
minimal for modified shift --> True
Popov for modified shift --> False
```

```
[470]: lmB = B.leading_matrix(shifts=best_s)
```

```

print("leading matrix:")
show(lmB, "\n")

B_popov = lmB.inverse() * B #  $O(m^{w-1} d)$  via  $K$ -linear algebra
show(LatexExpr(r"\text{degrees, after constant transformation }"
               r"\mathrm{lm}_{-\boldsymbol{\delta}}(B)^{-1} \cdot B,"
               r"\text{ at a cost } O(m^{\omega - 1} d):"))
show(B_popov.degree_matrix())

```

leading matrix:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 96 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 96 & 1 & 0 & 0 & 0 & 0 & 0 \\ 96 & 1 & 96 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

degrees, after constant transformation $\mathrm{lm}_{\delta}(B)^{-1} \cdot B$, at a cost $O(m^{\omega-1}d)$:

$$\begin{pmatrix} 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 125 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 124 & -1 & -1 & -1 & 0 \end{pmatrix}$$

```

[471]: print("minimal for modified shift -->", B_popov.is_reduced(shifts=best_s))
print("Popov   for modified shift -->", B_popov.is_popov(shifts=best_s))
print("Popov   for original shift -->", B_popov.is_popov(shifts=s), "and",
      ↪B_popov == P_popov)

```

```

minimal for modified shift --> True
Popov   for modified shift --> True
Popov   for original shift --> True and True

```

[]: