

NZMATH User Manual

(for version 1.1)

Contents

1	Overview	20
1.1	Introduction	20
1.1.1	Philosophy – Advantages over Other Systems	20
1.1.1.1	Open Source Software	20
1.1.1.2	Speed of Development	21
1.1.1.3	Bridging the Gap between Users And Developers	21
1.1.1.4	Link with Other Softwares	21
1.1.2	Information	21
1.1.3	Installation	22
1.1.3.1	Basic Installation	22
1.1.3.2	Installation for Windows Users	23
1.1.4	Tutorial	23
1.1.4.1	Sample Session	23
1.1.5	Note on the Document	25
2	Basic Utilities	26
2.1	config – setting features	26
2.1.1	Default Settings	26
2.1.1.1	Dependencies	26
2.1.1.2	Plug-ins	26
2.1.1.3	Assumptions	27
2.1.1.4	Files	27
2.1.2	Automatic Configuration	27
2.1.2.1	Checks	27
2.1.3	User Settings	28
2.2	bigrandom – random numbers	28
2.2.1	random – random number generator	28
2.2.2	randrange – random integer generator	29
2.2.3	map_choice – choice from image of mapping	29
2.3	bigrange – range-like generator functions	30
2.3.1	count – count up	30
2.3.2	range – range-like iterator	30
2.3.3	arithmetic_progression – arithmetic progression iterator	30
2.3.4	geometric_progression – geometric progression iterator	31

2.3.5	multirange – multiple range iterator	31
2.3.6	multirange_restrictions – multiple range iterator with restrictions	31
2.4	compatibility – Keep compatibility between Python versions	33
2.4.1	set, frozenset	33
2.4.2	card(virtualset)	33
3	Functions	34
3.1	algorithm – basic number theoretic algorithms	34
3.1.1	digital_method – univariate polynomial evaluation	34
3.1.2	digital_method_func – function of univariate polynomial evaluation	34
3.1.3	rl_binary_powering – right-left powering	35
3.1.4	lr_binary_powering – left-right powering	35
3.1.5	window_powering – window powering	35
3.1.6	powering_func – function of powering	36
3.2	arith1 - miscellaneous arithmetic functions	37
3.2.1	floorsqrt – floor of square root	37
3.2.2	floorpowerroot – floor of some power root	37
3.2.3	legendre - Legendre(Jacobi) Symbol	37
3.2.4	modsqrt – square root of a for modulo p	37
3.2.5	expand – p -adic expansion	37
3.2.6	inverse – inverse	37
3.2.7	CRT – Chinese Remainder Theorem	38
3.2.8	AGM – Arithmetic Geometric Mean	38
3.2.9	vp – p -adic valuation	38
3.2.10	issquare - Is it square?	38
3.2.11	log – integer part of logarithm	38
3.2.12	product – product of some numbers	39
3.3	arygcd – binary-like gcd algorithms	39
3.3.1	bit_num – the number of bits	39
3.3.2	binarygcd – gcd by the binary algorithm	40
3.3.3	arygcd_i – gcd over gauss-integer	40
3.3.4	arygcd_w – gcd over Eisenstein-integer	40
3.4	combinatorial – combinatorial functions	41
3.4.1	binomial – binomial coefficient	41
3.4.2	combinationIndexGenerator – iterator for combinations	41
3.4.3	factorial – factorial	41
3.4.4	permutationGenerator – iterator for permutation	41
3.4.5	fallingfactorial – the falling factorial	42
3.4.6	risingfactorial – the rising factorial	42
3.4.7	multinomial – the multinomial coefficient	42
3.4.8	bernoulli – the Bernoulli number	42
3.4.9	catalan – the Catalan number	42
3.4.10	euler – the Euler number	42
3.4.11	bell – the Bell number	43

3.4.12	stirling1 – Stirling number of the first kind	43
3.4.13	stirling2 – Stirling number of the second kind	43
3.4.14	partition_number – the number of partitions	44
3.4.15	partitionGenerator – iterator for partition	44
3.4.16	partition_conjugate – the conjugate of partition	44
3.5	cubic_root – cubic root, residue, and so on	46
3.5.1	c_root_p – cubic root mod p	46
3.5.2	c_residue – cubic residue mod p	46
3.5.3	c_symbol – cubic residue symbol for Eisenstein-integers	46
3.5.4	decompose_p – decomposition to Eisenstein-integers	46
3.5.5	cornacchia – solve $x^2 + dy^2 = p$	47
3.6	ecpp – elliptic curve primality proving	48
3.6.1	ecpp – elliptic curve primality proving	48
3.6.2	hilbert – Hilbert class polynomial	48
3.6.3	dedekind – Dedekind’s eta function	48
3.6.4	cmm – CM method	49
3.6.5	cmm_order – CM method with order	49
3.6.6	cornacchiamodify – Modified cornacchia algorithm	49
3.7	equation – solving equations, congruences	50
3.7.1	e1 – solve equation with degree 1	50
3.7.2	e1_ZnZ – solve congruent equation modulo n with degree 1	50
3.7.3	e2 – solve equation with degree 2	50
3.7.4	e2_Fp – solve congruent equation modulo p with degree 2	51
3.7.5	e3 – solve equation with degree 3	51
3.7.6	e3_Fp – solve congruent equation modulo p with degree 3	51
3.7.7	Newton – solve equation using Newton’s method	51
3.7.8	SimMethod – find all roots simultaneously	52
3.7.9	root_Fp – solve congruent equation modulo p	52
3.7.10	allroots_Fp – solve congruent equation modulo p	52
3.8	gcd – gcd algorithm	54
3.8.1	gcd – the greatest common divisor	54
3.8.2	binarygcd – binary gcd algorithm	54
3.8.3	extgcd – extended gcd algorithm	54
3.8.4	lcm – the least common multiple	54
3.8.5	gcd_of_list – gcd of many integers	55
3.8.6	coprime – coprime check	55
3.8.7	pairwise_coprime – coprime check of many integers	55
3.9	multiplicative – 乗法的数論関数	57
3.9.1	euler – オイラーのファイ関数	57
3.9.2	moebius – メビウス関数	57
3.9.3	sigma – 約数の冪の合計	57
3.10	prime – 素数判定, 素数生成	59
3.10.1	trialDivision – 試し割り算	59
3.10.2	spsp – 強擬素数テスト	59
3.10.3	smallSpsp – 小さい数に対する強擬素数テスト	59
3.10.4	miller – Miller の素数判定	59

3.10.5	millerRabin – Miller-Rabin の素数判定	59
3.10.6	lpsp – Lucas テスト	60
3.10.7	fpsp – Frobenius テスト	60
3.10.8	by_primitive_root – Lehmer’s test	60
3.10.9	full_euler – Brillhart & Selfridge’s test	60
3.10.10	apr – Jacobi 和テスト	61
3.10.11	primeq – 自動的な素数判定	61
3.10.12	prime – n 番目の素数	61
3.10.13	nextPrime – 次の素数を生成	61
3.10.14	randPrime – ランダムに素数を生成	61
3.10.15	generator – 素数生成	62
3.10.16	generator_eratosthenes – Eratosthenes の篩を使っている 素数生成	62
3.10.17	primorial – 素数の積	62
3.10.18	properDivisors – 真の約数	62
3.10.19	primitive_root – 平方根	62
3.10.20	Lucas_chain – Lucas 数列	63
3.11	prime_decomp – 素イデアル分解	64
3.11.1	prime_decomp – 素イデアル分解	64
3.12	quad – 虚二次体	65
3.12.1	ReducedQuadraticForm – 簡約二次形式クラス	65
3.12.1.1	inverse	66
3.12.1.2	disc	66
3.12.2	ClassGroup – 類群クラス	66
3.12.3	class_formula	67
3.12.4	class_number	67
3.12.5	class_group	67
3.12.6	class_number_bsgs	67
3.12.7	class_group_bsgs	68
3.13	round2 – the round 2 method	69
3.13.1	ModuleWithDenominator – bases of \mathbb{Z} -module with de- nominator.	70
3.13.1.1	get_rationals – get the bases as a list of rationals	71
3.13.1.2	get_polynomials – get the bases as a list of poly- nomials	71
3.13.1.3	determinant – determinant of the bases	71
3.13.2	round2(function)	72
3.13.3	Dedekind(function)	72
3.14	squarefree – Squarefreeness tests	73
3.14.1	Definition	73
3.14.2	lenstra – Lenstra’s condition	73
3.14.3	trial_division – trial division	73
3.14.4	trivial_test – trivial tests	74
3.14.5	viafactor – via factorization	74
3.14.6	viadecomposition – via partial factorization	74
3.14.7	lenstra_ternary – Lenstra’s condition, ternary version	74

3.14.8	trivial_test_ternary – trivial tests, ternary version	75
3.14.9	trial_division_ternary – trial division, ternary version . .	75
3.14.10	viafactor_ternary – via factorization, ternary version . . .	75
4	Classes	76
4.1	algfield – Algebraic Number Field	76
4.1.1	NumberField – number field	76
4.1.1.1	getConj – roots of polynomial	78
4.1.1.2	disc – polynomial discriminant	78
4.1.1.3	integer_ring – integer ring	78
4.1.1.4	field_discriminant – discriminant	78
4.1.1.5	basis – standard basis	78
4.1.1.6	signature – signature	79
4.1.1.7	POLRED – polynomial reduction	79
4.1.1.8	isIntBasis – check integral basis	79
4.1.1.9	isGaloisField – check Galois field	79
4.1.1.10	isFieldElement – check field element	79
4.1.1.11	getCharacteristic – characteristic	80
4.1.1.12	createElement – create an element	80
4.1.2	BasicAlgNumber – Algebraic Number Class by standard basis	81
4.1.2.1	inverse – inverse	83
4.1.2.2	getConj – roots of polynomial	83
4.1.2.3	getApprox – approximate conjugates	83
4.1.2.4	getCharPoly – characteristic polynomial	83
4.1.2.5	getRing – the field	83
4.1.2.6	trace – trace	83
4.1.2.7	norm – norm	84
4.1.2.8	isAlgInteger – check (algebraic) integer	84
4.1.2.9	ch_matrix – obtain MatAlgNumber object . . .	84
4.1.3	MatAlgNumber – Algebraic Number Class by matrix rep- resentation	85
4.1.3.1	inverse – inverse	87
4.1.3.2	getRing – the field	87
4.1.3.3	trace – trace	87
4.1.3.4	norm – norm	87
4.1.3.5	ch_basic – obtain BasicAlgNumber object . . .	87
4.1.4	changetype(function) – obtain BasicAlgNumber object . .	89
4.1.5	disc(function) – discriminant	89
4.1.6	fppoly(function) – polynomial over finite prime field . . .	89
4.1.7	qpoly(function) – polynomial over rational field	89
4.1.8	zpoly(function) – polynomial over integer ring	90
4.2	elliptic – elliptic class object	91
4.2.1	†ECCGeneric – generic elliptic curve class	92
4.2.1.1	simple – simplify the curve coefficient	94

4.2.1.2	changeCurve – change the curve by coordinate change	94
4.2.1.3	changePoint – change coordinate of point on the curve	94
4.2.1.4	coordinateY – Y-coordinate from X-coordinate	94
4.2.1.5	whetherOn – Check point is on curve	95
4.2.1.6	add – Point addition on the curve	95
4.2.1.7	sub – Point subtraction on the curve	95
4.2.1.8	mul – Scalar point multiplication on the curve	95
4.2.1.9	divPoly – division polynomial	95
4.2.2	ECoverQ – elliptic curve over rational field	96
4.2.2.1	point – obtain random point on curve	97
4.2.3	ECoverGF – elliptic curve over finite field	98
4.2.3.1	point – find random point on curve	99
4.2.3.2	naive – Frobenius trace by naive method	99
4.2.3.3	Shanks_Mestre – Frobenius trace by Shanks and Mestre method	99
4.2.3.4	Schoof – Frobenius trace by Schoof’s method	99
4.2.3.5	trace – Frobenius trace	100
4.2.3.6	order – order of group of rational points on the curve	100
4.2.3.7	pointorder – order of point on the curve	100
4.2.3.8	TatePairing – Tate Pairing	101
4.2.3.9	TatePairing_Extend – Tate Pairing with final exponentiation	101
4.2.3.10	WeilPairing – Weil Pairing	101
4.2.3.11	BSGS – point order by Baby-Step and Giant-Step	101
4.2.3.12	DLP_BSGS – solve Discrete Logarithm Problem by Baby-Step and Giant-Step	102
4.2.3.13	structure – structure of group of rational points	102
4.2.3.14	issupersingular – check supersingular curve	102
4.2.4	EC(function)	104
4.3	finitefield – Finite Field	105
4.3.1	†FiniteField – finite field, abstract	106
4.3.2	†FiniteFieldElement – element in finite field, abstract	107
4.3.3	FinitePrimeField – finite prime field	108
4.3.3.1	createElement – create element of finite prime field	109
4.3.3.2	getCharacteristic – get characteristic	109
4.3.3.3	issubring – subring test	109
4.3.3.4	issuperring – superring test	109
4.3.4	FinitePrimeFieldElement – element of finite prime field	110
4.3.4.1	getRing – get ring object	111
4.3.4.2	order – order of multiplicative group	111
4.3.5	ExtendedField – extended field of finite field	112
4.3.5.1	createElement – create element of extended field	113

	4.3.5.2	getCharacteristic – get characteristic	113
	4.3.5.3	issubring – subring test	113
	4.3.5.4	issuperring – superring test	113
	4.3.5.5	primitive_element – generator of multiplicative group	113
	4.3.6	ExtendedFieldElement – element of finite field	114
	4.3.6.1	getRing – get ring object	115
	4.3.6.2	inverse – inverse element	115
4.4	group	– algorithms for finite groups	116
	4.4.1	†Group – group structure	117
	4.4.1.1	setOperation – change operation	119
	4.4.1.2	†createElement – generate a GroupElement instance	119
	4.4.1.3	†identity – identity element	119
	4.4.1.4	grouporder – order of the group	119
	4.4.2	GroupElement – elements of group structure	121
	4.4.2.1	setOperation – change operation	123
	4.4.2.2	†getGroup – generate a Group instance	123
	4.4.2.3	order – order by factorization method	123
	4.4.2.4	t_order – order by baby-step giant-step	123
	4.4.3	†GenerateGroup – group structure with generator	125
	4.4.4	AbelianGenerate – abelian group structure with generator	126
	4.4.4.1	relationLattice – relation between generators	126
	4.4.4.2	computeStructure – abelian group structure	126
4.5	imaginary	– complex numbers and its functions	127
	4.5.1	ComplexField – field of complex numbers	128
	4.5.1.1	createElement – create Imaginary object	129
	4.5.1.2	getCharacteristic – get characteristic	129
	4.5.1.3	issubring – subring test	129
	4.5.1.4	issuperring – superring test	129
	4.5.2	Complex – a complex number	130
	4.5.2.1	getRing – get ring object	131
	4.5.2.2	arg – argument of complex	131
	4.5.2.3	conjugate – complex conjugate	131
	4.5.2.4	copy – copied number	131
	4.5.2.5	inverse – complex inverse	131
	4.5.3	ExponentialPowerSeries – exponential power series	132
	4.5.4	AbsoluteError – absolute error	132
	4.5.5	RelativeError – relative error	132
	4.5.6	exp(function) – exponential value	132
	4.5.7	expi(function) – imaginary exponential value	132
	4.5.8	log(function) – logarithm	132
	4.5.9	sin(function) – sine function	132
	4.5.10	cos(function) – cosine function	132
	4.5.11	tan(function) – tangent function	132
	4.5.12	sinh(function) – hyperbolic sine function	132

4.5.13	cosh(function) – hyperbolic cosine function	132
4.5.14	tanh(function) – hyperbolic tangent function	132
4.5.15	atanh(function) – hyperbolic arc tangent function	133
4.5.16	sqrt(function) – square root	133
4.6	intresidue – integer residue	134
4.6.1	IntegerResidueClass – integer residue class	135
4.6.1.1	getRing – get ring object	136
4.6.1.2	getResidue – get residue	136
4.6.1.3	getModulus – get modulus	136
4.6.1.4	inverse – inverse element	136
4.6.1.5	minimumAbsolute – minimum absolute repre- sentative	136
4.6.1.6	minimumNonNegative – smallest non-negative representative	136
4.6.2	IntegerResidueClassRing – ring of integer residue	137
4.6.2.1	createElement – create IntegerResidueClass object	138
4.6.2.2	isfield – field test	138
4.6.2.3	getInstance – get instance of IntegerResidueClass- Ring	138
4.7	lattice – Lattice	139
4.7.1	Lattice – lattice	139
4.7.1.1	createElement – create element	140
4.7.1.2	bilinearForm – bilinear form	140
4.7.1.3	isCyclic – Check whether cyclic lattice or not	140
4.7.1.4	isIdeal – Check whether ideal lattice or not	140
4.7.2	LatticeElement – element of lattice	141
4.7.2.1	getLattice – Find lattice belongs to	142
4.7.3	LLL(function) – LLL reduction	143
4.8	matrix – 行列	144
4.8.1	Matrix – 行列	145
4.8.1.1	map – 各成分に関数を適用	147
4.8.1.2	reduce – 繰り返し関数を適用	147
4.8.1.3	copy – コピー作成	147
4.8.1.4	set – 成分を設定	147
4.8.1.5	setRow – m 行目に行ベクトルを設定	148
4.8.1.6	setColumn – n 列目に列ベクトルを設定	148
4.8.1.7	getRow – i 行目の行ベクトルを返す	148
4.8.1.8	getColumn – j 列目の列ベクトルを返す	148
4.8.1.9	swapRow – 二つの行ベクトルを交換	148
4.8.1.10	swapColumn – 二つの列ベクトルを交換	149
4.8.1.11	insertRow – 行ベクトルを挿入	149
4.8.1.12	insertColumn – 列ベクトル挿入	149
4.8.1.13	extendRow – 行ベクトルを伸張	149
4.8.1.14	extendColumn – 列ベクトルを伸張	149
4.8.1.15	deleteRow – 行ベクトルを削除	150
4.8.1.16	deleteColumn – 列ベクトルを削除	150

4.8.1.17	transpose – 転置行列	150
4.8.1.18	getBlock – ブロック行列	150
4.8.1.19	subMatrix – 部分行列	150
4.8.2	SquareMatrix – 正方行列	152
4.8.2.1	isUpperTriangularMatrix – 上三角行列かどうか	153
4.8.2.2	isLowerTriangularMatrix – 下三角行列かどうか	153
4.8.2.3	isDiagonalMatrix – 対角行列かどうか	153
4.8.2.4	isScalarMatrix – スカラー行列かどうか	153
4.8.2.5	isSymmetricMatrix – 対称行列かどうか	153
4.8.3	RingMatrix – 成分が環に属する行列	155
4.8.3.1	getCoefficientRing – 係数環を返す	156
4.8.3.2	toFieldMatrix – 係数環として体を設定	156
4.8.3.3	toSubspace – ベクトル空間としてみなす	156
4.8.3.4	hermiteNormalForm (HNF) – Hermite 正規形	156
4.8.3.5	exthermiteNormalForm (extHNF) – 拡張 Hermite 正規形アルゴリズム	156
4.8.3.6	kernelAsModule – \mathbb{Z} 加群としての核	157
4.8.4	RingSquareMatrix – 各成分が環に属する正方行列	158
4.8.4.1	getRing – 行列の環を返す	159
4.8.4.2	isOrthogonalMatrix – 直交行列かどうか	159
4.8.4.3	isAlternatingMatrix (isAntiSymmetricMatrix, isSkewSymmetricMatrix) – 交代行列かどうか	159
4.8.4.4	isSingular – 特異行列かどうか	159
4.8.4.5	trace – トレース	159
4.8.4.6	determinant – 行列式	160
4.8.4.7	cofactor – 余因子	160
4.8.4.8	commutator – 交換子	160
4.8.4.9	characteristicMatrix – 特性行列	160
4.8.4.10	adjugateMatrix – 随伴行列	160
4.8.4.11	cofactorMatrix (cofactors) – 余因子行列	160
4.8.4.12	smithNormalForm (SNF, elementary_divisor) – Smith 正規形 (SNF)	161
4.8.4.13	extsmithNormalForm (extSNF) – Smith 正規形 (SNF)	161
4.8.5	FieldMatrix – 各成分が体に属する行列	162
4.8.5.1	kernel – 核	163
4.8.5.2	image – 像	163
4.8.5.3	rank – 階数	163
4.8.5.4	inverseImage – 逆像: 一次方程式の基底解	163
4.8.5.5	solve – 一次方程式の解	163
4.8.5.6	columnEchelonForm – 列階段行列	164
4.8.6	FieldSquareMatrix – 各成分が体に属する正方行列	165
4.8.6.1	triangulate – 行基本変形による三角化	166
4.8.6.2	inverse – 逆行列	166
4.8.6.3	hessenbergForm – Hessenberg 行列	166
4.8.6.4	LUdecomposition – LU 分解	166

4.8.7	†MatrixRing – 行列の環	167
4.8.7.1	unitMatrix – 単位行列	168
4.8.7.2	zeroMatrix – 零行列	168
4.8.7.3	getInstance(class function) – キャッシュされたインスタンスを返す	169
4.8.8	Subspace – 有限次元ベクトル空間の部分空間	170
4.8.8.1	issubspace – 部分空間かどうか	171
4.8.8.2	toBasis – 基底を選択	171
4.8.8.3	supplementBasis – 最大階数にする	171
4.8.8.4	sumOfSubspaces – 部分空間の和	171
4.8.8.5	intersectionOfSubspaces – 部分空間の共通部分	171
4.8.8.6	fromMatrix(class function) – 部分空間を作成	173
4.8.9	createMatrix[function] – インスタンスを作成	174
4.8.10	identityMatrix(unitMatrix)[function] – 単位行列	174
4.8.11	zeroMatrix[function] – 零行列	174
4.9	module – HNF による加群/イデアル	176
4.9.1	Submodule – 行列表現としての部分加群	177
4.9.1.1	getGenerators – 加群の生成元	178
4.9.1.2	isSubmodule – 部分加群かどうか	178
4.9.1.3	isEqual – self と other が同じ加群かどうか	178
4.9.1.4	isContain – other が self に含まれているかどうか	178
4.9.1.5	toHNF – HNF に変換	179
4.9.1.6	sumOfSubmodules – 部分加群の和	179
4.9.1.7	intersectionOfSubmodules – 部分加群の共通部分	179
4.9.1.8	represent_element – 一次結合として成分を表す	179
4.9.1.9	linear_combination – 一次結合を計算	179
4.9.2	fromMatrix(class function) – 部分加群を作成	181
4.9.3	Module – 数体上の加群	182
4.9.3.1	toHNF – Hermite 正規形 (HNF) に変換	184
4.9.3.2	copy – コピーを作成	184
4.9.3.3	intersect – 共通部分を返す	184
4.9.3.4	issubmodule – 部分加群かどうか	184
4.9.3.5	issupermodule – 部分加群かどうか	184
4.9.3.6	represent_element – 一次結合として表す	184
4.9.3.7	change_base_module – 基底変換	185
4.9.3.8	index – 加群のサイズ	185
4.9.3.9	smallest_rational – 有理数体上の \mathbb{Z} 生成元	185
4.9.4	Ideal – 数体上のイデアル	187
4.9.4.1	inverse – 逆元	188
4.9.4.2	issubideal – 部分イデアルかどうか	188
4.9.4.3	issuperideal – 部分加群かどうか	188
4.9.4.4	gcd – 最大公約数	188
4.9.4.5	lcm – 最小公倍数	188
4.9.4.6	norm – ノルム	189
4.9.4.7	isIntegral – 整イデアルかどうか	189
4.9.5	Ideal_with_generator – 生成元によるイデアル	190

4.9.5.1	copy - コピーを作成	192
4.9.5.2	to_HNFRepresentation - HNF イdealに変換	192
4.9.5.3	twoElementRepresentation - 二つの要素で表す	192
4.9.5.4	smallest_rational - 有理数体上の \mathbb{Z} 生成元	192
4.9.5.5	inverse - 逆元	192
4.9.5.6	norm - ノルム	193
4.9.5.7	intersect - 共通部分	193
4.9.5.8	issubideal - 部分イdealかどうか	193
4.9.5.9	issuperideal - 部分イdealかどうか	193
4.10	permute - 置換 (対称) 群	195
4.10.1	Permute - 置換群の元	196
4.10.1.1	setKey - key を変換	198
4.10.1.2	getValue - “value” を得る	198
4.10.1.3	getGroup - PermGroup を得る	198
4.10.1.4	numbering - インデックスを与える	198
4.10.1.5	order - 元の位数	198
4.10.1.6	ToTranspose - 互換の積として表す	199
4.10.1.7	ToCyclic - ExPermute の元に対応する	199
4.10.1.8	sgn - 置換記号	199
4.10.1.9	types - 巡回置換の形式	199
4.10.1.10	ToMatrix - 置換行列	199
4.10.2	ExPermute - 巡回表現としての置換群の元	201
4.10.2.1	setKey - key を変換	203
4.10.2.2	getValue - “value” を得る	203
4.10.2.3	getGroup - PermGroup を得る	203
4.10.2.4	order - 元の位数	203
4.10.2.5	ToNormal - 普通の表現	203
4.10.2.6	simplify - 単純な値を使用	204
4.10.2.7	sgn - 置換符号	204
4.10.3	PermGroup - 置換群	205
4.10.3.1	createElement - シードから元を作成	206
4.10.3.2	identity - 単位元	206
4.10.3.3	identity_c - 巡回表現の単位元	206
4.10.3.4	grouporder - 群の位数	206
4.10.3.5	randElement - 無作為に元を選ぶ	206
4.11	rational - 整数と有理数	208
4.11.1	Integer - 整数	209
4.11.1.1	getRing - ring オブジェクトを得る	210
4.11.1.2	actAdditive - 2 進の加法鎖の加法	210
4.11.1.3	actMultiplicative - 2 進の加法鎖の乗法	210
4.11.2	IntegerRing - 整数環	211
4.11.2.1	createElement - Integer オブジェクトを作成	212
4.11.2.2	gcd - 最大公約数	212
4.11.2.3	extgcd - 拡張 GCD	212
4.11.2.4	lcm - 最小公倍数	212
4.11.2.5	getQuotientField - 有理数体オブジェクトを得る	212

4.11.2.6	issubring – 部分環かどうか判定	212
4.11.2.7	issuperring – 含んでいるかどうか判定	213
4.11.3	Rational – 有理数	214
4.11.3.1	getRing – ring オブジェクトを得る	215
4.11.3.2	decimalString – 小数を表す	215
4.11.3.3	expand – 連分数による表現	215
4.11.4	RationalField – 有理数体	216
4.11.4.1	createElement – Rational オブジェクトを返す	217
4.11.4.2	classNumber – 類数を得る	217
4.11.4.3	getQuotientField – 有理数体オブジェクトを返す	217
4.11.4.4	issubring – 部分環かどうか判定	217
4.11.4.5	issuperring – 含んでいるかどうか判定	217
4.12	real – real numbers and its functions	218
4.12.1	RealField – field of real numbers	220
4.12.1.1	getCharacteristic – get characteristic	221
4.12.1.2	issubring – subring test	221
4.12.1.3	issuperring – superring test	221
4.12.2	Real – a Real number	222
4.12.2.1	getRing – get ring object	223
4.12.3	Constant – real number with error correction	224
4.12.4	ExponentialPowerSeries – exponential power series	224
4.12.5	AbsoluteError – absolute error	224
4.12.6	RelativeError – relative error	224
4.12.7	exp(function) – exponential value	224
4.12.8	sqrt(function) – square root	224
4.12.9	log(function) – logarithm	224
4.12.10	log1piter(function) – iterator of log(1+x)	224
4.12.11	piGaussLegendre(function) – pi by Gauss-Legendre	224
4.12.12	eContinuedFraction(function) – Napier's Constant by continued fraction expansion	224
4.12.13	floor(function) – floor the number	225
4.12.14	ceil(function) – ceil the number	225
4.12.15	trunc(function) – round-off the number	225
4.12.16	sin(function) – sine function	225
4.12.17	cos(function) – cosine function	225
4.12.18	tan(function) – tangent function	225
4.12.19	sinh(function) – hyperbolic sine function	225
4.12.20	cosh(function) – hyperbolic cosine function	225
4.12.21	tanh(function) – hyperbolic tangent function	225
4.12.22	asin(function) – arc sine function	226
4.12.23	acos(function) – arc cosine function	226
4.12.24	atan(function) – arc tangent function	226
4.12.25	atan2(function) – arc tangent function	226
4.12.26	hypot(function) – Euclidean distance function	226
4.12.27	pow(function) – power function	226
4.12.28	degrees(function) – convert angle to degree	226

4.12.29	radians(function) – convert angle to radian	226
4.12.30	fabs(function) – absolute value	226
4.12.31	fmod(function) – modulo function over real	226
4.12.32	frexp(function) – expression with base and binary exponent	227
4.12.33	ldexp(function) – construct number from base and binary exponent	227
4.12.34	EulerTransform(function) – iterator yields terms of Euler transform	227
4.13	ring – for ring object	228
4.13.1	†Ring – abstract ring	229
4.13.1.1	createElement – create an element	230
4.13.1.2	getCharacteristic – characteristic as ring	230
4.13.1.3	issubring – check subring	230
4.13.1.4	issuperring – check superring	230
4.13.1.5	getCommonSuperring – get common ring	231
4.13.2	†CommutativeRing – abstract commutative ring	232
4.13.2.1	getQuotientField – create quotient field	233
4.13.2.2	isdomain – check domain	233
4.13.2.3	isnoetherian – check Noetherian domain	233
4.13.2.4	isufd – check UFD	233
4.13.2.5	ispid – check PID	233
4.13.2.6	iseuclidean – check Euclidean domain	234
4.13.2.7	isfield – check field	234
4.13.2.8	registerModuleAction – register action as ring	234
4.13.2.9	hasaction – check if the action has	234
4.13.2.10	getaction – get the registered action	234
4.13.3	†Field – abstract field	236
4.13.3.1	gcd – gcd	237
4.13.4	†QuotientField – abstract quotient field	238
4.13.5	†RingElement – abstract element of ring	239
4.13.5.1	getRing – getRing	240
4.13.6	†CommutativeRingElement – abstract element of commutative ring	241
4.13.6.1	mul_module_action – apply a module action	242
4.13.6.2	exact_division – division exactly	242
4.13.7	†FieldElement – abstract element of field	243
4.13.8	†QuotientFieldElement – abstract element of quotient field	244
4.13.9	†Ideal – abstract ideal	245
4.13.9.1	issubset – check subset	246
4.13.9.2	issuperset – check superset	246
4.13.9.3	reduce – reduction with the ideal	246
4.13.10	†ResidueClassRing – abstract residue class ring	247
4.13.11	†ResidueClass – abstract an element of residue class ring	248
4.13.12	†CommutativeRingProperties – properties for CommutativeRingProperties	249
4.13.12.1	isfield – check field	250

4.13.12.2	setIsfield – set field	250
4.13.12.3	iseuclidean – check euclidean	250
4.13.12.4	setIseuclidean – set euclidean	250
4.13.12.5	ispid – check PID	251
4.13.12.6	setIspid – set PID	251
4.13.12.7	isufd – check UFD	251
4.13.12.8	setIsufd – set UFD	251
4.13.12.9	isnoetherian – check Noetherian	252
4.13.12.10	setIsnoetherian – set Noetherian	252
4.13.12.11	isdomain – check domain	252
4.13.12.12	setIsdomain – set domain	252
4.13.13	getRingInstance(function)	254
4.13.14	getRing(function)	254
4.13.15	inverse(function)	254
4.13.16	exact_division(function)	254
4.14	vector – ベクトルオブジェクトとその計算	256
4.14.1	Vector – ベクトルクラス	257
4.14.1.1	copy – 自身のコピー	259
4.14.1.2	set – 他の compo を設定	259
4.14.1.3	indexOfNoneZero – 0 でない最初の位置	259
4.14.1.4	toMatrix – Matrix オブジェクトに変換	259
4.14.2	innerProduct(function) – 内積	261
4.15	factor.ecm – ECM factorization	262
4.15.1	ecm – elliptic curve method	262
4.16	factor.find – find a factor	263
4.16.1	trialDivision – trial division	263
4.16.2	pmom – $p - 1$ method	263
4.16.3	rhomethod – ρ method	263
4.17	factor.methods – factoring methods	265
4.17.1	factor – easiest way to factor	265
4.17.2	ecm – elliptic curve method	265
4.17.3	mpqs – multi-polynomial quadratic sieve method	266
4.17.4	pmom – $p - 1$ method	266
4.17.5	rhomethod – ρ method	266
4.17.6	trialDivision – trial division	266
4.18	factor.misc – miscellaneous functions related factoring	268
4.18.1	allDivisors – all divisors	268
4.18.2	primeDivisors – prime divisors	268
4.18.3	primePowerTest – prime power test	268
4.18.4	squarePart – square part	268
4.18.5	FactoredInteger – integer with its factorization	270
4.18.5.1	is_divisible_by	271
4.18.5.2	exact_division	271
4.18.5.3	divisors	271
4.18.5.4	proper_divisors	271
4.18.5.5	prime_divisors	271

4.18.5.6	square_part	271
4.18.5.7	squarefree_part	272
4.18.5.8	copy	272
4.19	factor.mpqqs – MPQS	272
4.19.1	mpqsfind	272
4.19.2	mpqs	272
4.19.3	eratosthenes	273
4.20	factor.util – utilities for factorization	274
4.20.1	FactoringInteger – keeping track of factorization	275
4.20.1.1	getNextTarget – next target	276
4.20.1.2	getResult – result of factorization	276
4.20.1.3	register – register a new factor	276
4.20.1.4	sortFactors – sort factors	276
4.20.2	FactoringMethod – method of factorization	278
4.20.2.1	factor – do factorization	279
4.20.2.2	†continue_factor – continue factorization	279
4.20.2.3	†find – find a factor	279
4.20.2.4	†generate – generate prime factors	279
4.21	poly.factor – 多項式の因数分解	281
4.21.1	brute_force_search – 総当たりで因数分解を探す	281
4.21.2	divisibility_test – 可除性テスト	281
4.21.3	minimum_absolute_injection – 係数を絶対値最小表現に渡す	281
4.21.4	padic_factorization – p 進分解	281
4.21.5	upper_bound_of_coefficient – Landau-Mignotte の係数の上界	282
4.21.6	zassenhaus – Zassenhaus 法による平方因子のない整数係数多項式の因数分解	282
4.21.7	integerpolynomialfactorization – 整数多項式の因数分解	282
4.22	poly.formalsum – 形式和	283
4.22.1	FormalSumContainerInterface – インターフェースクラス	284
4.22.1.1	construct_with_default – コピーを構成	285
4.22.1.2	iterterms – 項のイテレータ	285
4.22.1.3	itercoefficients – 係数のイテレータ	285
4.22.1.4	iterbases – 基数のイテレータ	285
4.22.1.5	terms – 項のリスト	285
4.22.1.6	coefficients – 係数のリスト	285
4.22.1.7	bases – 基数のリスト	286
4.22.1.8	terms_map – 項に写像を施す	286
4.22.1.9	coefficients_map – 係数に写像を施す	286
4.22.1.10	bases_map – 基数に写像を施す	286
4.22.2	DictFormalSum – 辞書で実装された形式和	287
4.22.3	ListFormalSum – リストで実装された形式和	287
4.23	poly.groebner – グレブナー基底	288
4.23.1	buchberger – グレブナー基底を得るための素朴なアルゴリズム	288

4.23.2	normal_strategy – グレブナー基底を得る普通のアルゴリズム	288
4.23.3	reduce_groebner – 簡約グレブナー基底	288
4.23.4	s_polynomial – S-polynomial	289
4.24	poly.hensel – ヘンゼルリフト	289
4.24.1	HenselLiftPair – ヘンゼルリフトの組	291
4.24.1.1	lift – 一段階引き上げる	292
4.24.1.2	lift_factors – a1 と a2 を引き上げる	292
4.24.1.3	lift_ladder – u1 と u2 を引き上げる	292
4.24.2	HenselLiftMulti – 複数多項式に対するヘンゼルリフト	292
4.24.2.1	lift – 一段階引き上げる	294
4.24.2.2	lift_factors – 因数を引き上げる	294
4.24.2.3	lift_ladder – u1 と u2 を引き上げる	294
4.24.3	HenselLiftSimultaneously	295
4.24.3.1	lift – 一段階引き上げる	296
4.24.3.2	first_lift – 最初のステップ	296
4.24.3.3	general_lift – 次のステップ	296
4.24.4	lift_upto – main 関数	296
4.25	poly.multiutil – 多変数多項式に対するユーティリティ	297
4.25.1	RingPolynomial	298
4.25.1.1	getRing	299
4.25.1.2	getCoefficientRing	299
4.25.1.3	leading_variable	299
4.25.1.4	nest	299
4.25.1.5	unnest	299
4.25.2	DomainPolynomial	299
4.25.2.1	pseudo_divmod	301
4.25.2.2	pseudo_floordiv	301
4.25.2.3	pseudo_mod	301
4.25.2.4	exact_division	301
4.25.3	UniqueFactorizationDomainPolynomial	302
4.25.3.1	gcd	303
4.25.3.2	resultant	303
4.25.4	polynomial – さまざまな多項式に対するファクトリ関数	303
4.25.5	prepare_indeterminates – 不定元連立宣言	303
4.26	poly.multivar – 多変数多項式	304
4.26.1	PolynomialInterface – 全ての多変数多項式の基底クラス	305
4.26.2	BasicPolynomial – 多項式の基本的な実装	305
4.26.3	TermIndices – 多変数多項式の項のインデックス	305
4.26.3.1	pop	306
4.26.3.2	gcd	306
4.26.3.3	lcm	306
4.27	poly.ratfunc – 有理関数	307
4.27.1	RationalFunction – 有理関数クラス	308
4.27.1.1	getRing – 有理関数体を得る	309
4.28	poly.ring – 多項式環	310

4.28.1	PolynomialRing – 多項式環	311
4.28.1.1	getInstance – クラスメソッド	312
4.28.1.2	getCoefficientRing	312
4.28.1.3	getQuotientField	312
4.28.1.4	issubring	312
4.28.1.5	issuperring	312
4.28.1.6	getCharacteristic	312
4.28.1.7	createElement	312
4.28.1.8	gcd	312
4.28.1.9	isdomain	313
4.28.1.10	iseuclidean	313
4.28.1.11	isnoetherian	313
4.28.1.12	ispid	313
4.28.1.13	isufd	313
4.28.2	RationalFunctionField – 有理関数体	313
4.28.2.1	getInstance – クラスメソッド	314
4.28.2.2	createElement	314
4.28.2.3	getQuotientField	314
4.28.2.4	issubring	314
4.28.2.5	issuperring	314
4.28.2.6	unnest	314
4.28.2.7	gcd	314
4.28.2.8	isdomain	315
4.28.2.9	iseuclidean	315
4.28.2.10	isnoetherian	315
4.28.2.11	ispid	315
4.28.2.12	isufd	315
4.28.3	PolynomialIdeal – 多項式環のイデアル	315
4.28.3.1	reduce	316
4.28.3.2	issubset	316
4.28.3.3	issuperset	316
4.29	poly.termorder – 項順序	317
4.29.1	TermOrderInterface – 項順序のインターフェース	318
4.29.1.1	cmp	319
4.29.1.2	format	319
4.29.1.3	leading_coefficient	319
4.29.1.4	leading_term	319
4.29.2	UnivarTermOrder – 一変数多項式に対する項順序	319
4.29.2.1	format	320
4.29.2.2	degree	320
4.29.2.3	tail_degree	320
4.29.3	MultivarTermOrder – 多変数多項式に対する項順序	320
4.29.3.1	format	321
4.29.4	weight_order – 重み付き順序付け	321
4.30	poly.uniutil – 一変数多項式のためのユーティリティ	322
4.30.1	RingPolynomial – 可換環上の多項式	323

4.30.1.1	getRing	324
4.30.1.2	getCoefficientRing	324
4.30.1.3	shift_degree_to	324
4.30.1.4	split_at	324
4.30.2	DomainPolynomial – 整域上の多項式	324
4.30.2.1	pseudo_divmod	326
4.30.2.2	pseudo_floordiv	326
4.30.2.3	pseudo_mod	326
4.30.2.4	exact_division	326
4.30.2.5	scalar_exact_division	326
4.30.2.6	discriminant	327
4.30.2.7	to_field_polynomial	327
4.30.3	UniqueFactorizationDomainPolynomial – UFD 上の多項式	327
4.30.3.1	content	327
4.30.3.2	primitive_part	327
4.30.3.3	subresultant_gcd	328
4.30.3.4	subresultant_extgcd	328
4.30.3.5	resultant	328
4.30.4	IntegerPolynomial – 有理整数環上の多項式	328
4.30.5	FieldPolynomial – 体上の多項式	329
4.30.5.1	content	330
4.30.5.2	primitive_part	330
4.30.5.3	mod	330
4.30.5.4	scalar_exact_division	330
4.30.5.5	gcd	330
4.30.5.6	extgcd	330
4.30.6	FinitePrimeFieldPolynomial – 有限素体上の多項式	331
4.30.6.1	mod_pow – モジュロとべき乗	332
4.30.6.2	pthroot	332
4.30.6.3	squarefree_decomposition	332
4.30.6.4	distinct_degree_decomposition	332
4.30.6.5	split_same_degrees	333
4.30.6.6	factor	333
4.30.6.7	isirreducible	333
4.30.7	polynomial – さまざまな多項式に対するファクトリ関数	333
4.31	poly.univar – 一変数多項式	334
4.31.1	PolynomialInterface – 全ての一変数多項式に対する基底ク ラス	335
4.31.1.1	differentiate – 形式微分	336
4.31.1.2	downshift_degree – 多項式の次数を下げる	336
4.31.1.3	upshift_degree – 多項式の次数を上げる	336
4.31.1.4	ring_mul – 環上の乗法	336
4.31.1.5	scalar_mul – スカラーの乗法	336
4.31.1.6	term_mul – 項の乗法	336
4.31.1.7	square – 自身との乗法	337
4.31.2	BasicPolynomial – 多項式の基本的実装	337

4.31.3 SortedPolynomial – 項がソートされたままの状態に維持する多項式	337
4.31.3.1 degree – 次数	338
4.31.3.2 leading_coefficient – 主係数	338
4.31.3.3 leading_term – 主項	338
4.31.3.4 ring_mul_karatsuba – Karatsuba 法による乗算	338

Chapter 1

Overview

1.1 Introduction

NZMATH[8] is a number theory oriented calculation system mainly developed by the Nakamura laboratory at Tokyo Metropolitan University. NZMATH system provides you mathematical, especially number-theoretic computational power. It is freely available and distributed under the BSD license. The most distinctive feature of NZMATH is that it is written entirely using a scripting language called Python.

If you want to learn how to start using NZMATH, see Installation (section 1.1.3) and Tutorial (section 1.1.4).

1.1.1 Philosophy – Advantages over Other Systems

In this section, we discuss philosophy of NZMATH, that is, the advantages of NZMATH compared to other similar systems.

1.1.1.1 Open Source Software

Many computational algebra systems, such as Maple[4], Mathematica[5], and Magma[3] are fare-paying systems. These non-free systems are not distributed with source codes. Then, users cannot modify such systems easily. It narrows these system's potentials for users not to take part in developing them. NZMATH, on the other hand, is an open-source software and the source codes are openly available. Furthermore, NZMATH is distributed under the BSD license. BSD license claims as-is and redistribution or commercial use are permitted provided that these packages retain the copyright notice. NZMATH users can develop it just as they like.

1.1.1.2 Speed of Development

We took over developing of SIMATH[10], which was developed under the leadership of Prof. Zimmer at Saarlandes University in Germany. However, it costs a lot of time and efforts to develop these system. Almost all systems including SIMATH are implemented in C or C++ for execution speed, but we have to take the time to work memory management, construction of an interactive interpreter, preparation for multiple precision package and so on. In this regard, we chose Python which is a modern programming language. Python provides automatic memory management, a sophisticated interpreter and many useful packages. We can concentrate on development of mathematical matters by using Python.

1.1.1.3 Bridging the Gap between Users And Developers

KANT/KASH[2] and PARI/GP[9] are similar systems to NZMATH. But programming languages for modifying these systems are different between users and developers. We think the gap makes evolution speed of these systems slow. On the other hand, NZMATH has been developed with Python for bridging this gap. Python grammar is easy to understand and users can read easily codes written by Python. And NZMATH, which is one of Python libraries, works on very wide platform including UNIX/Linux, Macintosh, Windows, and so forth. Users can modify the programs and feedback to developers with a light heart. So developers can absorb their thinking. Then NZMATH will progress to more flexible user-friendly system.

1.1.1.4 Link with Other Softwares

NZMATH distributed as a Python library enables us to link other Python packages with it. For example, NZMATH can be used with IPython[1], which is a comfortable interactive interpreter. And it can be linked with matplotlib[6], which is a powerful graphic software. Also mpmath[7], which is a module for floating-point operation, can improve efficiency of NZMATH. In fact, the module `ecpp` improves performance with mpmath. There are many softwares implemented in Python. Many of these packages are freely available. Users can use NZMATH with these packages and create an unthinkable powerful system.

1.1.2 Information

NZMATH has more than 25 modules. These modules cover a lot of territory including elementary number theoretic methods, combinatorial theoretic methods, solving equations, primality, factorization, multiplicative number theoretic functions, matrix, vector, polynomial, rational field, finite field, elliptic curve, and so on. NZMATH manual for users is at:

<http://tnt.math.se.tmu.ac.jp/nzmth/manual/>

If you are interested in NZMATH, please visit the official website below to obtain more information about it.

<http://tnt.math.se.tmu.ac.jp/nzmeth/>

Note that NZMATH can be used even if users do not have any experience of writing programs in Python.

1.1.3 Installation

In this section, we explain how to install NZMATH. If you use Windows (Windows XP, Windows Vista, Windows 7 etc.) as an operating system (OS), then see 1.1.3.2 “Install for Windows Users”.

1.1.3.1 Basic Installation

There are three steps for installation of NZMATH.

First, check whether Python is installed in the computer. Python 2.5 or a higher version is needed for NZMATH. If you do not have a copy of Python, please install it first. Python is available from <http://www.python.org/>.

Second, download a NZMATH package and expand it. It is distributed at official web site:

<http://tnt.math.se.tmu.ac.jp/nzmeth/download>

or at sourceforge.net:

http://sourceforge.net/project/showfiles.php?group_id=171032

The package can be easily extracted, depending on the operating system. For systems with recent GNU tar, type a single command below:

```
% tar xf NZMATH-*.tar.gz
```

where, % is a command line prompt. With standard tar, type

```
% gzip -cd NZMATH-*.tar.gz | tar xf -
```

. Please read *.* as the version number of which you downloaded the package. For example, if the latest version is 1.0.0, then type the following command.

```
% tar xf NZMATH-1.0.0.tar.gz
```

Then, a subdirectory named NZMATH-*.* is created.

Finally, install NZMATH to the standard python path. Usually, this can be translated into writing files somewhere under /usr/lib or /usr/local/lib. So the appropriate write permission may be required at this step. Typically, type commands below:

```
% cd NZMATH-*.*
% su
# python setup.py install
```

1.1.3.2 Installation for Windows Users

We also distribute installation packages for specific platforms. Especially, we started distributing the installer for Windows in 2007.

Please download the installer (NZMATH-*.*.win32Install.exe) from

<http://tnt.math.se.tmu.ac.jp/nzmeth/download>

or at sourceforge.net:

http://sourceforge.net/project/showfiles.php?group_id=171032

Here, we explain a way of installing NZMATH with the installer. First please open the installer. If you use Windows Vista or higher version, UAC (User Account Control) may ask if you run the program. click "Allow". Then the setup window will open. Following the steps in the setup wizard, you can install NZMATH with only three clicks.

1.1.4 Tutorial

In this section, we describe how to use NZMATH.

1.1.4.1 Sample Session

Start your Python interpreter. That is, open your command interpreter such as Terminal for MacOS or bash/csh for linux, type the strings "python" and press the key Enter.

Examples

```
% python
Python 2.6.1 (r261:67515, Jan 14 2009, 10:59:13)
[GCC 4.1.2 20071124 (Red Hat 4.1.2-42)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

For windows users, it normally means opening IDLE (Python GUI), which is a Python software.

Examples

```
Python 2.6.1 (r261:67517, Dec 4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
```

```
*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
```



```
interface and no data is sent to or received from the Internet.  
*****
```

```
IDLE 2.6.1  
>>>
```

Here, '>>>' is a Python prompt, which means that the system waits you to input commands.

Then, type:

Examples

```
>>> from nzmeth import *  
>>>
```

This command enables you to use all NZMATH features. If you use only a specific module (the term “module” is explained later), for example, prime, type as the following:

Examples

```
>>> from nzmeth import prime  
>>>
```

You are ready to use NZMATH. For example, type the string “prime.nextPrime(1000)”, then you obtain ‘1009’ as the smallest prime among numbers greater than 1000.

Examples

```
>>> prime.nextPrime(1000)  
1009  
>>>
```

“prime” is a name of a module, which is a NZMATH file including Python codes. “nextPrime” is a name of a function, which outputs values after the system executes some processes for inputs. NZMATH has various functions for mathematical or algorithmic computations. See [3 Functions](#).

Also, we can create some mathematical objects. For example, you may use the module “matrix”. If you want to define the matrix

$$\begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix}$$

and compute the square, then type as the following:

Examples

```
>>> A = matrix.Matrix(2, 2, [1, 2]+[5, 6])
>>> print A
1 2
5 6
>>> print A ** 2
11 14
35 46
>>>
```

“Matrix” is a name of a class, which is a template of mathematical objects. See [4](#) Classes for using NZMATH classes.

The command “print” enables us to represent outputs with good-looking forms. The data structure such as “[a, b, c, ...]” is called list. Also, we use various Python data structures like tuple “(a, b, c, ...)”, dictionary “{ $x_1 : y_1, x_2 : y_2, x_3 : y_3, \dots$ }” etc. Note that we do not explain Python’s syntax in detail because it is not absolutely necessary to use NZMATH. However, we recommend that you learn Python for developing your potential. Python grammar are easy to study. For information on how to use Python, see <http://docs.python.org> or many other documents about Python.

1.1.5 Note on the Document

† Some beginnings of lines or blocks such as sections or sentences may be marked †. This means these lines or blocks is for advanced users. For example, the class *FiniteFieldElement* (See **FinitePrimeFieldElement**) is one of abstract classes in NZMATH, which can be inherited to new classes similar to the finite field.

[...] For example, we may sometimes write as *function(a,b[,c,d])*. It means the argument “c, d” or only “d” can be discarded. Such functions use “default argument values”, which is one of the feature of Python.

(See <http://docs.python.org/tutorial/controlflow.html#default-argument-values>)

Warning: Python also have the feature “keyword arguments”. We have tried to keep the feature in NZMATH too. However, some functions cannot be used with this feature because these functions are written expecting that arguments are given in order.

Chapter 2

Basic Utilities

2.1 config – setting features

このモジュールはユーザーの config ファイルで設定する。 [User Settings](#) を参照すると詳細あり。

2.1.1 Default Settings

2.1.1.1 Dependencies

Some third party / platform dependent modules are possibly used, and they are configurable.

HAVE_MPMATH `mpmath` is a package providing multiprecision math. See its [project page](#). This package is used in **ecpp** module.

HAVE_SQLITE3 `sqlite3` is the default database module for Python , but it need to be enabled at the build time.

HAVE_NET いくつかの関数はネットワークに接続することがあります。ネットワークに接続していない環境では、この設定を `false` にしておくとまれに処理が高速になることがあります。

2.1.1.2 Plug-ins

PLUGIN_MATH Python standard float/complex types and [math/cmath](#) modules only provide fixed precision (double precision), but sometimes multi-precision floating point is needed.

2.1.1.3 Assumptions

Some conjectures are useful for assuring the validity of a faster algorithm.

All assumptions are default to `False`, but you can set them `True` if you believe them.

GRH Generalized Riemann Hypothesis. For example, primality test is $O((\log n)^2)$ if GRH is true while $O((\log n)^6)$ or something without it.

2.1.1.4 Files

DATADIR The directory where `NZMATH` (static) data files are stored. The default will be `os.path.join(sys.prefix, 'share', 'nzmatah')` or `os.path.join(sys.prefix, 'Data', 'nzmatah')` on Windows.

2.1.2 Automatic Configuration

The items above can be set automatically by testing the environment.

2.1.2.1 Checks

Here are check functions.

The constants accompanying the check functions which enable the check if it is `True`, can be overridden in user settings.

Both check functions and constants are not exposed.

check_mpmath() Check whether `mpmath` is available or not.
constant: `CHECK_MPMATH`

check_sqlite3() Check if `sqlite3` is importable or not. `pysqlite2` may be a substitution.
constant: `CHECK_SQLITE3`

check_net() Check the net connection by HTTP call.
constant: `CHECK_NET`

check_plugin_math() Check which math plug-in is available.
constant: `CHECK_PLUGIN_MATH`

default_datadir() Return default value for `DATADIR`.

This function selects the value from various candidates. If this function is called with `DATADIR` set, the value of (previously-defined) `DATADIR` is the first candidate to be returned. Other possibilities are, `sys.prefix + 'Data/nzmatah'` on Windows, or `sys.prefix + 'share/nzmatah'` on other platforms.

Be careful that all the above paths do not exist, the function returns `None`.

constant: `CHECK_DATADIR`

2.1.3 User Settings

The module try to load the user's config file named *nzmathconf.py*. The search path is the following:

1. The directory which is specified by an environment variable `NZMATHCONFDIR`.
2. If the platform is Windows, then
 - (a) If an environment variable `APPDATA` is set, `APPDATA/nzmath`.
 - (b) If, alternatively, an environment variable `USERPROFILE` is set, `USERPROFILE/Application Data/nzmath`.
3. On other platforms, if an environment variable `HOME` is set, `HOME/.nzmath.d`.

nzmathconf.py is a `Python` script. Users can set the constants like `HAVE_MPMATH`, which will override the default settings. These constants, except assumption ones, are automatically set, unless constants accompanying the check functions are false (see the [Automatic Configuration](#) section above).

2.2 bigrandom – random numbers

Historical Note The module was written for replacement of the `Python` standard module `random`, because in the era of `Python 2.2` (prehistorical period of `NZMATH`) the random module raises `OverflowError` for long integer arguments for the `randrange` function, which is the only function having a use case in `NZMATH`.

After the creation of `Python 2.3`, it was theoretically possible to use `random.randrange`, since it started to accept long integer as its argument. Use of it was, however, not considered, since there had been the `bigrandom` module. It was lucky for us. In fall of 2006, we found a bug in `random.randrange` and reported it (see issue tracker); the `random.randrange` accepts long integers but returns unreliable result for truly big integers. The bug was fixed for `Python 2.5.1`. You can, therefore, use `random.randrange` instead of `bigrandom.randrange` for `Python 2.5.1` or higher.

2.2.1 random – random number generator

`random()` → *float*

$[0, 1)$ 内の浮動小数点数の値をランダムに返す。

This function is an alias to `random.random` in the `Python` standard library.

2.2.2 randrange – random integer generator

```
randrange(start: integer, stop: integer=None, step: integer=1 )  
→ integer
```

ある範囲の乱数の値を返す。

The argument names do not correspond to their roles, but users are familiar with the **range** built-in function of Python and understand the semantics. Calling with one argument n , then the result is an integer in the range $[0, n)$ chosen randomly. With two arguments n and m , in $[n, m)$, and with third l , in $[n, m) \cap (n + l\mathbb{Z})$.

This function is almost the same as `random.randrange` in the Python standard library. See the historical note [2.2](#).

Examples

```
>>> randrange(4, 10000, 3)  
9919L  
>>> randrange(4 * 10**60)  
31925916908162253969182327491823596145612834799876775114620151L
```

2.2.3 map_choice – choice from image of mapping

```
map_choice(mapping: function, upperbound: integer )  
→ integer
```

Return a choice from a set given as the image of the mapping from natural numbers (more precisely `range(upperbound)`). In other words, it is equivalent to: `random.choice([mapping(i) for i in range(upperbound)])`, if upperbound is small enough for the list size limit.

The mapping can be a partial function, i.e. it may return `None` for some input. However, if the resulting set is empty, it will end up with an infinite loop.

2.3 bigrange – range-like generator functions

2.3.1 count – count up

`count(n: integer=0) → iterator`

n まで数え上げる。 . [itertools.count](#) 参照。

n must be int, long or rational.Integer.

2.3.2 range – range-like iterator

`range(start: integer, stop: integer=None, step: integer=1)
→ iterator`

多倍長対応の range である。

It can generate more than [sys.maxint](#) elements, which is the limitation of the [range](#) built-in function.

The argument names do not correspond to their roles, but users are familiar with the [range](#) built-in function of Python and understand the semantics. Note that the output is not a list.

Examples

```
>>> range(1, 100, 3) # built-in
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46,
 49, 52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91,
 94, 97]
>>> bigrange.range(1, 100, 3)
<generator object at 0x18f8c8>
```

2.3.3 arithmetic_progression – arithmetic progression iterator

`arithmetic_progression(init: integer, difference: integer)
→ iterator`

Return an iterator which generates an arithmetic progression starting from init and difference step.

2.3.4 `geometric_progression` – geometric progression iterator

```
geometric_progression(init: integer, ratio: integer )  
    → iterator
```

Return an iterator which generates a geometric progression starting from `init` and multiplying `ratio`.

2.3.5 `multirange` – multiple range iterator

```
multirange(triples: list of range triples ) → iterator
```

Return an iterator over Cartesian product of elements of ranges.

Be cautious that using `multirange` usually means you are trying to do brute force looping.

The range triples may be doubles (`start`, `stop`) or single (`stop`), but they have to be always tuples.

Examples

```
>>> bigrange.multirange([(1, 10, 3), (1, 10, 4)])  
<generator object at 0x18f968>  
>>> list(_)  
[(1, 1), (1, 5), (1, 9), (4, 1), (4, 5), (4, 9), (7, 1),  
 (7, 5), (7, 9)]
```

2.3.6 `multirange_restrictions` – multiple range iterator with restrictions

```
multirange_restrictions(triples: list of range triples, **kwds: keyword arguments )  
    → iterator
```

`multirange_restrictions` is an iterator similar to the `multirange` but putting restrictions on each ranges.

Restrictions are specified by keyword arguments: `ascending`, `descending`, `strictly_ascending` and `strictly_descending`.

A restriction `ascending`, for example, is a sequence that specifies the indices where the number emitted by the range should be greater than or equal to the number at the previous index. Other restrictions `descending`, `strictly_ascending`

and `strictly_descending` are similar. Compare the examples below and of **multirange**.

Examples

```
>>> bigrange.multirange_restrictions([(1, 10, 3), (1, 10, 4)], ascending=(1,))
<generator object at 0x18f978>
>>> list(_)
[(1, 1), (1, 5), (1, 9), (4, 5), (4, 9), (7, 9)]
```

2.4 compatibility – Keep compatibility between Python versions

This module should be simply imported:

```
import nzmeth.compatibility
```

then it will do its tasks.

2.4.1 set, frozenset

The module provides `set` for Python 2.3. Python ≥ 2.4 have `set` in built-in namespace, while Python 2.3 has `sets` module and `sets.Set`. The `set` the module provides for Python 2.3 is the `sets.Set`. Similarly, `sets.ImmutableSet` would be assigned to `frozenset`. Be careful that the compatibility is not perfect. Note also that `NZMATH`'s recommendation is Python 2.5 or higher in 2.x series.

2.4.2 card(virtualset)

Return cardinality of the virtualset.

The built-in `len()` raises `OverflowError` when the result is greater than `sys.maxint`. It is not clear this restriction will go away in the future. The function `card()` ought to be used instead of `len()` for obtaining cardinality of sets or set-like objects in `nzmeth`.

Chapter 3

Functions

3.1 algorithm – basic number theoretic algorithms

3.1.1 digital_method – univariate polynomial evaluation

digital_method(coefficients: *list*, val: *object*, add: *function*, mul: *function*, act: *function*, power: *function*, zero: *object*, one: *object*)
→ *object*

Evaluate a univariate polynomial corresponding to coefficients at val.

If the polynomial corresponding to coefficients is of R -coefficients for some ring R , then val should be in an R -algebra D .

coefficients should be a descending ordered list of tuples (d, c) , where d is an integer which expresses the degree and c is an element of R which expresses the coefficient. All operations 'add', 'mul', 'act', 'power', 'zero', 'one' should be explicitly given, where:

'add' means addition ($D \times D \rightarrow D$), 'mul' multiplication ($D \times D \rightarrow D$), 'act' action of R ($R \times D \rightarrow D$), 'power' powering ($D \times \mathbf{Z} \rightarrow D$), 'zero' the additive unit (an constant) in D and 'one', the multiplicative unit (an constant) in D .

3.1.2 digital_method_func – function of univariate polynomial evaluation

digital_method(add: *function*, mul: *function*, act: *function*, power: *function*, zero: *object*, one: *object*)
→ *function*

Return a function which evaluates polynomial corresponding to 'coefficients' at 'val' from an iterator 'coefficients' and an object 'val'.

All operations 'add', 'mul', 'act', 'power', 'zero', 'one' should be inputted in

a manner similar to **digital_method**.

3.1.3 rl_binary_powering – right-left powering

```
rl_binary_powering(element: object, index: integer, mul: function,  
square: function=None, one: object=None, )  
→ object
```

Return element to the index power by using right-left binary method.

index should be a non-negative integer. If square is None, square is defined by using mul.

3.1.4 lr_binary_powering – left-right powering

```
lr_binary_powering(element: object, index: integer, mul: function,  
square: function=None, one: object=None, )  
→ object
```

Return element to the index power by using left-right binary method.

index should be a non-negative integer. If square is None, square is defined by using mul.

3.1.5 window_powering – window powering

```
window_powering(element: object, index: integer, mul: function,  
square: function=None, one: object=None, )  
→ object
```

Return element to the index power by using small-window method.

The window size is selected by average analytic optimization.

index should be a non-negative integer. If square is None, square is defined by using mul.

3.1.6 powering_func – function of powering

```
powering_func(mul: function, square: function=None, one: object=None, type: integer=0 )  
    → function
```

Return a function which computes 'element' to the 'index' power from an object 'element' and an integer 'index'.

If square is None, square is defined by using mul. type should be an integer which means one of the following:

```
0; rl_binary_powering  
1; lr_binary_powering  
2; window_powering
```

Examples

```
>>> d_func = algorithm.digital_method_func(  
... lambda a,b:a+b, lambda a,b:a*b, lambda i,a:i*a, lambda a,i:a**i,  
... matrix.zeroMatrix(3,0), matrix.identityMatrix(3,1)  
... )  
>>> coefficients = [(2,1), (1,2), (0,1)] # X^2+2*X+I  
>>> A = matrix.SquareMatrix(3, [1,2,3]+[4,5,6]+[7,8,9])  
>>> d_func(coefficients, A) # A**2+2*A+I  
[33L, 40L, 48L]+[74L, 92L, 108L]+[116L, 142L, 169L]  
>>> p_func = algorithm.powering_func(lambda a,b:a*b, type=2)  
>>> p_func(A, 10) # A**10 by window method  
[132476037840L, 162775103256L, 193074168672L]+[300005963406L, 368621393481L,  
437236823556L]+[467535888972L, 574467683706L, 681399478440L]
```

3.2 arith1 - miscellaneous arithmetic functions

3.2.1 floorsqrt – floor of square root

`floorsqrt(a: integer/Rational) → integer`

a の 2 乗根の小数点切り捨てた値を返す。

3.2.2 floorpowerroot – floor of some power root

`floorpowerroot(n: integer, k: integer) → integer`

n の k 乗根の小数点切り捨てた値を返す。

3.2.3 legendre - Legendre(Jacobi) Symbol

`legendre(a: integer, m: integer) → integer`

Legendre 記号と Jacobi 記号を返す $\left(\frac{a}{m}\right)$ 。

3.2.4 modsqrt – square root of a for modulo p

`modsqrt(a: integer, p: integer) → integer`

a の 2 乗根が存在する時は p を法とする a の 2 乗根の値を返す。さもなければエラーを返す。

p は素数。

3.2.5 expand – p-adic expansion

`expand(n: integer, m: integer) → list`

n の m 進展開を返す。 .

n は正の整数。 m は 2 以上。 . 出力は降順の係数展開のリスト。 .

3.2.6 inverse – inverse

`inverse(x: integer, p: integer) → integer`

法 p における x の逆関数を返す。

p は素数。

3.2.7 CRT – Chinese Remainder Theorem

CRT(nlist: list) → integer

Return the uniquely determined integer satisfying all modulus conditions given by nlist.

入力 nlist は 2 つの要素からなるリスト。一つ目は割った余りで二つ目は割る数。どちらも整数。

3.2.8 AGM – Arithmetic Geometric Mean

AGM(a: integer, b: integer) → float

a と b の算術幾何平均を返す。

3.2.9 vp – p -adic valuation

vp(n: integer, p: integer, k: integer=0) → tuple

p 進評価と n の他の部分群を返す。

$\dagger k$ が与えられたら、評価と np^k の他の部分群を返す。

3.2.10 issquare - Is it square?

issquare(n: integer) → integer

n が二乗になっていたら根を返し、さもなければ 0 を返す。

3.2.11 log – integer part of logarithm

log(n: integer, base: integer=2) → integer

n の対数の整数部分を返す。base.

3.2.12 product – product of some numbers

```
product(iterable: list, init: integer/Rational=None)
    → prod: integer/Rational
```

iterable のすべての要素からなるものを返す。

If init is given, the multiplication starts with init instead of the first element in iterable.

Input list iterable must be list of numbers including integers, **Rational** etc. The output prod may be determined by the type of elements of iterable and init.

Examples

```
>>> arith1.AGM(10, 15)
12.373402181181522

>>> arith1.CRT([[2, 5], [3, 7]])
17
>>> arith1.CRT([[2, 5], [3, 7], [5, 11]])
192
>>> arith1.expand(194, 5)
[4, 3, 2, 1]
>>> arith1.vp(54, 3)
(3, 2)
>>> arith1.product([1.5, 2, 2.5])
7.5
>>> arith1.product([3, 4], 2)
24
>>> arith1.product([])
1
```

3.3 arygcd – binary-like gcd algorithms

3.3.1 bit_num – the number of bits

```
bit_num(a: integer) → integer
```

a のビット数の値を返す。

3.3.2 binarygcd – gcd by the binary algorithm

binarygcd(a: *integer*, b: *integer*) → *integer*

binary gcd algorithm を使って a, b の最大公約数の値を返す。

3.3.3 arygcd_i – gcd over gauss-integer

**arygcd_i(a1: *integer*, a2: *integer*, b1: *integer*, b2: *integer*)
→ (*integer*, *integer*)**

二つの gauss 数体 $a1+a2i$, $b1+b2i$ の最大公約数の値を返す。“ i ” は虚数。

If the output of `arygcd_i(a1, a2, b1, b2)` is $(c1, c2)$, then the gcd of $a1+a2i$ and $b1+b2i$ equals $c1+c2i$.

†This function uses $(1+i)$ -ary gcd algorithm, which is an generalization of the binary algorithm, proposed by A.Weilert[20].

3.3.4 arygcd_w – gcd over Eisenstein-integer

**arygcd_w(a1: *integer*, a2: *integer*, b1: *integer*, b2: *integer*)
→ (*integer*, *integer*)**

Eisenstein 数体 $a1+a2\omega$, $b1+b2\omega$ の最大公約数の値を返す。“ ω ” は 1 の虚立方根。

If the output of `arygcd_w(a1, a2, b1, b2)` is $(c1, c2)$, then the gcd of $a1+a2\omega$ and $b1+b2\omega$ equals $c1+c2\omega$.

†This functions uses $(1-\omega)$ -ary gcd algorithm, which is an generalization of the binary algorithm, proposed by I.B. Damgård and G.S. Frandsen [16].

Examples

```
>>> arygcd.binarygcd(32, 48)
16
>>> arygcd_i(1, 13, 13, 9)
(-3, 1)
>>> arygcd_w(2, 13, 33, 15)
(4, 5)
```

3.4 combinatorial – combinatorial functions

3.4.1 binomial – binomial coefficient

binomial(*n*: integer, *m*: integer) → integer

n と *m* の二項係数の値を返す。すなはち、 $\frac{n!}{(n-m)!m!}$ 。

† 便宜上、binomial(*n*, *n*+*i*) は 0 整数 *i* に対して 0 を返し、binomial(0,0) は 1 を返す。

n は自然数。 *m* は整数。

3.4.2 combinationIndexGenerator – iterator for combinations

combinationIndexGenerator(*n*: integer, *m*: integer) → iterator

Return an iterator which generates indices of *m* element subsets of *n* element set.

combination_index_generator is an alias of combinationIndexGenerator.

3.4.3 factorial – factorial

factorial(*n*: integer) → integer

n! の値を返す。 *n* は整数。

3.4.4 permutationGenerator – iterator for permutation

permutationGenerator(*n*: integer) → iterator

Generate all permutations of *n* elements as list iterator.

The number of generated list is *n*'s **factorial**, so be careful to use big *n*.

permutation_generator is an alias of permutationGenerator.

3.4.5 fallingfactorial – the falling factorial

fallingfactorial(*n*: integer, *m*: integer) → integer

下降階乗の値を返す。; *n* から *m* へ。i.e. $n(n-1)\cdots(n-m+1)$.

3.4.6 risingfactorial – the rising factorial

risingfactorial(*n*: integer, *m*: integer) → integer

上昇階乗の値を返す。; *n* から *m* へ。i.e. $n(n+1)\cdots(n+m-1)$.

3.4.7 multinomial – the multinomial coefficient

multinomial(*n*: integer, *parts*: list) → integer

多項係数の値を返す。

parts は自然数数列。*parts* の要素をすべてあわせると *n* と等しくなる。

3.4.8 bernoulli – the Bernoulli number

bernoulli(*n*: integer) → Rational

n 次 Bernoulli 数の値を返す。

3.4.9 catalan – the Catalan number

catalan(*n*: integer) → integer

n 次 Catalan 数の値を返す。

3.4.10 euler – the Euler number

euler(*n*: integer) → integer

n 次 Euler 数の値を返す。

3.4.11 bell – the Bell number

bell(n: *integer*) → *integer*

n 次ベル数の値を返す。

ベル数 b の定義:

$$b(n) = \sum_{i=0}^n S(n, i),$$

S は第 2 種スターリング数。(**stirling2**).

3.4.12 stirling1 – Stirling number of the first kind

stirling1(n: *integer*, m: *integer*) → *integer*

第 1 種スターリング数の値を返す。

s はスターリング数。 $(x)_n$ は下降階乗。

$$(x)_n = \sum_{i=0}^n s(n, i) x^i.$$

s satisfies the recurrence relation:

$$s(n, m) = s(n-1, m-1) - (n-1)s(n-1, m).$$

3.4.13 stirling2 – Stirling number of the second kind

stirling2(n: *integer*, m: *integer*) → *integer*

Return Stirling number of the second kind.

S はスターリング数。 $(x)_i$ は下降階乗。:

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

S は以下の関係を満たす。

$$S(n, m) = S(n-1, m-1) + mS(n-1, m)$$

3.4.14 `partition_number` – the number of partitions

`partition_number(n: integer) → integer`

`n` の分割数の値を返す。

3.4.15 `partitionGenerator` – iterator for partition

`partitionGenerator(n: integer, maxi: integer=0) → iterator`

Return an iterator which generates partitions of `n`.

If `maxi` is given, then summands are limited not to exceed `maxi`.

The number of partitions (given by `partition_number`) grows exponentially, so be careful to use big `n`.

`partition_generator` is an alias of `partitionGenerator`.

3.4.16 `partition_conjugate` – the conjugate of partition

`partition_conjugate(partition: tuple) → tuple`

Return the conjugate of partition.

Examples

```
>>> combinatorial.binomial(5, 2)
10L
>>> combinatorial.factorial(3)
6L
>>> combinatorial.fallingfactorial(7, 3) == 7 * 6 * 5
True
>>> combinatorial.risingfactorial(7, 3) == 7 * 8 * 9
True
>>> combinatorial.multinomial(7, [2, 2, 3])
210L
>>> for idx in combinatorial.combinationIndexGenerator(5, 3):
...     print idx
...
[0, 1, 2]
[0, 1, 3]
[0, 1, 4]
```

```

[0, 2, 3]
[0, 2, 4]
[0, 3, 4]
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]
>>> for part in combinatorial.partitionGenerator(5):
...     print part
...
(5,)
(4, 1)
(3, 2)
(3, 1, 1)
(2, 2, 1)
(2, 1, 1, 1)
(1, 1, 1, 1, 1)
>>> combinatorial.partition_number(5)
7
>>> def limited_summands(n, maxi):
...     "partition with limited number of summands"
...     for part in combinatorial.partitionGenerator(n, maxi):
...         yield combinatorial.partition_conjugate(part)
...
>>> for part in limited_summands(5, 3):
...     print part
...
(2, 2, 1)
(3, 1, 1)
(3, 2)
(4, 1)
(5,)

```

3.5 cubic_root – cubic root, residue, and so on

3.5.1 c_root_p – cubic root mod p

`c_root_p(a: integer, p: integer) → list`

a 法 p の a の 3 乗根の値を返す。(すなわち、 $x^3 = a \pmod{p}$).

p は素数。
この関数は a の 3 乗根のすべての値をリストで返す。

3.5.2 c_residue – cubic residue mod p

`c_residue(a: integer, p: integer) → integer`

法 p で有理数 a が 3 乗になっているか調べる。

もし $p \mid a$ なら 0 を返す。また、法 p で a が 3 乗になっているならば 1 を返す。
そうでなければ (3 乗になっていないとき) -1 を返す。

p は素数。

3.5.3 c_symbol – cubic residue symbol for Eisenstein-integers

`c_symbol(a1: integer, a2: integer, b1: integer, b2: integer)
→ integer`

二つの Eisenstein 整数である (Jacobi) 立方剰余記号の値を返す。 $\left(\frac{a1+a2\omega}{b1+b2\omega}\right)_3$, ω は 1 の 3 乗根の値である。

もし $b1 + b2\omega$ が $\mathbb{Z}[\omega]$ に含まれる素数であるならば、 $a1 + a2\omega$ は立方剰余かわかる。

$b1 + b2\omega$ は $1 - \omega$ に分けられないと仮定する。 .

3.5.4 decomposite_p – decomposition to Eisenstein-integers

`decompose_p(p: integer) → (integer, integer)`

$\mathbb{Z}[\omega]$ に含まれる素数の一つ p の値を返す。

もし出力が (a, b) なら、 $\frac{p}{a+b\omega}$ は $\mathbb{Z}[\omega]$. に含まれる素数である。すなわち p が

$\mathbb{Z}[\omega]$. に含まれる $a + b\omega$ and $p/(a + b\omega)$ の二つの素因数に分解することができる。

p は有理数かつ素数。 $p \equiv 1 \pmod{3}$ と仮定する。

3.5.5 cornacchia – solve $x^2 + dy^2 = p$

cornacchia(d: integer, p: integer) → (integer, integer)

$x^2 + dy^2 = p$ の値を返す。

この関数は Cornacchia のアルゴリズムを使用。 [\[13\]](#) 参照。

p は有理数かつ素数。 d は $0 < d < p$ の関係を充たす。 . この関数は $x^2 + dy^2 = p$ の値として (x, y) を返す。

Examples

```
>>> cubic_root.c_root_p(1, 13)
[1, 3, 9]
>>> cubic_root.c_residue(2, 7)
-1
>>> cubic_root.c_symbol(3, 6, 5, 6)
1
>>> cubic_root.decompose_p(19)
(2, 5)
>>> cubic_root.cornacchia(5, 29)
(3, 2)
```


3.6 ecpp – elliptic curve primality proving

このモジュールは ECPP (Elliptic Curve Primality Proving) の様々な関数から作られている。

It is probable that the module will be refactored in the future so that each function be placed in other modules.

ecpp モジュールは mpmath のダウンロードが必要。

3.6.1 ecpp – elliptic curve primality proving

`ecpp(n: integer, era: list=None) → bool`

楕円曲線素数証明を行う。
もし `n` が素数なら `True` を返す。さもなければ `False` を返す。

また、`era` とは素数のリストである。(これは ERAtosthenes に基づいている。)

`n` は巨大な整数。

3.6.2 hilbert – Hilbert class polynomial

`hilbert(D: integer) → (integer, list)`

類数と Hilbert 類方程式 for 虚 2 次体 with fundamental 判別式 `D` の値を返す。

この関数は Hilbert 類方程式の係数のリストを返す。
†もし **HAVE_NET** を設定しているなら、まず <http://hilbert-class-polynomial.appspot.com/> を検索し、もし `ramD` に一致する情報が見つからなければ Hilbert 類方程式を直接計算してください。
`D` は `int` または `long`。 [15] 参照。

3.6.3 dedekind – Dedekind’s eta function

`dedekind(tau: mpmath.mpc, floatpre: integer) → mpmath.mpc`

Return Dedekind のイータ of a complex number `tau` in the upper half-plane.

Additional argument `floatpre` specifies the precision of calculation in decimal digits.

`floatpre` must be positive int.

3.6.4 cmm – CM method

cmm(p: integer) → list

CM 曲線のカーブパラメータの値を返す。

もし一つだけ楕円曲線でよいのならば **cmm_order** を使うとよい。

p は奇素数でなければならない。
この関数は (a, b) のリストを返す。(a, b) は Weierstrass' short form を表している。

3.6.5 cmm_order – CM method with order

cmm_order(p: integer) → list

CM 曲線のカーブパラメータの値と位数を返す。

もし一つだけ楕円曲線でよいのならば **cmm_order** を使うとよい。

p は奇素数でなければならない。
この関数は (a, b, order) のリストを返す。(a, b) は Weierstrass' short form を表し、order は楕円曲線での位数を表す。

3.6.6 cornacchiamodify – Modified cornacchia algorithm

cornacchiamodify(d: integer, p: integer) → list

(u, v) of $u^2 - dv^2 = 4p$ の解を返す。

もし解がなければ ValueError を返す。

p は素数。d は $d < 0$ and $d > -4p$ with $d \equiv 0, 1 \pmod{4}$ を満たす整数。

Examples

```
>>> ecpp.ecpp(300000000000000000053)
True
>>> ecpp.hilbert(-7)
(1, [3375, 1])
>>> ecpp.cmm(7)
[(6L, 3L), (5L, 4L)]
>>> ecpp.cornacchiamodify(-7, 29)
(2, 4)
```

3.7 equation – solving equations, congruences

In the following descriptions, some type aliases are used.

poly_list :

poly_list is a list [a0, a1, ..., an] representing a polynomial coefficients in ascending order, i.e., meaning $a_0 + a_1X + \cdots + a_nX^n$. The type of each ai depends on each function (explained in their descriptions).

integer :

integer is one of *int*, *long* or **Integer**.

complex :

complex includes all number types in the complex field: **integer**, *float*, *complex* of Python, **Rational** of NZMATH, etc.

3.7.1 e1 – solve equation with degree 1

e1(f: poly_list) → complex

$ax + b = 0$ の値を返す。

f は **complex** の linkingoneequationpoly_list [b, a] でなければならない。

3.7.2 e1_ZnZ – solve congruent equation modulo n with degree 1

e1_ZnZ(f: poly_list, n: integer) → integer

$ax + b \equiv 0 \pmod{n}$ の値を返す。

f は **integer** の **poly_list** [b, a] でなければならない。

3.7.3 e2 – solve equation with degree 2

e2(f: poly_list) → tuple

$ax^2 + bx + c = 0$ の値を返す。

f は **complex** の **poly_list** [c, b, a] でなければならない。
結果のタプルは副根も含め二つの根である。

3.7.4 e2_Fp – solve congruent equation modulo p with degree 2

`e2_Fp(f: poly_list, p: integer) → list`

$ax^2 + bx + c \equiv 0 \pmod{p}$ の値を返す。

同じ値が返ってきたならば、その値は多重根である。

f は `integers [c, b, a]` の `poly_list` でなければならない。さらに、p は素数整数。 `integer`.

3.7.5 e3 – solve equation with degree 3

`e3(f: poly_list) → list`

$ax^3 + bx^2 + cx + d = 0$ の値を返す。

f は `complex` の `poly_list [d, c, b, a]` でなければならない。
この結果のタプルには重根を含めて三つの根がある。

3.7.6 e3_Fp – solve congruent equation modulo p with degree 3

`e3_Fp(f: poly_list, p: integer) → list`

$ax^3 + bx^2 + cx + d \equiv 0 \pmod{p}$ の値を返す。

同じ値が返ってきたならば、その値は多重根である。

f は `integer` の `poly_list [d, c, b, a]` でなければならない。In addition, p は素数整数である。 `integer`.

3.7.7 Newton – solve equation using Newton's method

`Newton(f: poly_list, initial: complex=1, repeat: integer=250)`
`→ complex`

$a_n x^n + \dots + a_1 x + a_0 = 0$ の値を返す。

もしすべての根を得たいのなら `SimMethod` を使うことをお勧めする。
† もし initial が実数根を持たない実数ならばこの関数は役に立たない。

f は **complex** の **poly_list** でなければならない。
 initial は an initial approximation **complex** number. repeat は根を近似する数である。

3.7.8 SimMethod – find all roots simultaneously

SimMethod(f: **poly_list**, NewtonInitial: **complex**=1, repeat: *integer*=250)
 → *list*

$a_n x^n + \cdots + a_1 x + a_0$ の根の一つを返す。

† もしこの方程式が多重根を持っていたら、エラーが返ってくるかもしれない。

f は **complex** の **poly_list** でなければならない。
 NewtonInitial と repeat は近似値を得るため **Newton** を通過するだろう。

3.7.9 root_Fp – solve congruent equation modulo p

root_Fp(f: **poly_list**, p: *integer*) → *integer*

$a_n x^n + \cdots + a_1 x + a_0 \equiv 0 \pmod{p}$ の根の一人を返す。

すべての根を得たいのなら **allroots_Fp** を使ってください。

f は **integer** の **poly_list** でなければならない。さらに p は素数。
 根が一つもないときはこの関数は何も返さない。

3.7.10 allroots_Fp – solve congruent equation modulo p

allroots_Fp(f: **poly_list**, p: *integer*) → *integer*

$a_n x^n + \cdots + a_1 x + a_0 \equiv 0 \pmod{p}$ のすべての根を返す。

f は **integer** の **poly_list** でなければならない。さらに p は素数。
 根が一つもないときはこの関数はからのリストを返す。

Examples

```
>>> equation.e1([1, 2])
```

```

-0.5
>>> equation.e1([1j, 2])
-0.5j
>>> equation.e1_ZnZ([3, 2], 5)
1
>>> equation.e2([-3, 1, 1])
(1.3027756377319946, -2.3027756377319948)
>>> equation.e2_Fp([-3, 1, 1], 13)
[6, 6]
>>> equation.e3([1, 1, 2, 1])
[(-0.12256116687665397-0.74486176661974479j),
 (-1.7548776662466921+1.8041124150158794e-16j),
 (-0.12256116687665375+0.74486176661974468j)]
>>> equation.e3_Fp([1, 1, 2, 1], 7)
[3]
>>> equation.Newton([-3, 2, 1, 1])
0.84373427789806899
>>> equation.Newton([-3, 2, 1, 1], 2)
0.84373427789806899
>>> equation.Newton([-3, 2, 1, 1], 2, 1000)
0.84373427789806899
>>> equation.SimMethod([-3, 2, 1, 1])
[(0.84373427789806887+0j),
 (-0.92186713894903438+1.6449263775999723j),
 (-0.92186713894903438-1.6449263775999723j)]
>>> equation.root_Fp([-3, 2, 1, 1], 7)
>>> equation.root_Fp([-3, 2, 1, 1], 11)
9L
>>> equation.allroots_Fp([-3, 2, 1, 1], 7)
[]
>>> equation.allroots_Fp([-3, 2, 1, 1], 11)
[9L]
>>> equation.allroots_Fp([-3, 2, 1, 1], 13)
[3L, 7L, 2L]

```

3.8 gcd – gcd algorithm

3.8.1 gcd – the greatest common divisor

`gcd(a: integer, b: integer) → integer`

二つの整数 a と b の最大公約数の値を返す。

a, b は `int` または `long` 型か **Integer**。Even if one of the arguments is negative, 結果は整数。

3.8.2 binarygcd – binary gcd algorithm

`binarygcd(a: integer, b: integer) → integer`

バイナリー GCD アルゴリズムを使って二つの整数 a と b の最大公約数の値を返す。

† この関数は **binarygcd** のエイリアスである。

a, b は `int` または `long` 型。または、**Integer**。

3.8.3 extgcd – extended gcd algorithm

`extgcd(a: integer, b: integer) → (integer, integer, integer)`

$d = au + bv$ の関係式を満たす a と b の最大公約数 d の値を返す。

a, b は `int` または `long` 型 **Integer**.
結果は (u, v, d) の形で返ってくる。

3.8.4 lcm – the least common multiple

`lcm(a: integer, b: integer) → integer`

二つの整数 a と b の最小公倍数の値を返す。

† もし a と b どちらも 0 ならば、`enception` を返す。

a, b は `int` または `long` 型、または **Integer**.

3.8.5 gcd_of_list – gcd of many integers

`gcd_of_list(integers: list) → list`

倍数の最大公約数の値を返す。

与えれた integers $[x_1, \dots, x_n]$ に対して、リスト $[d, [c_1, \dots, c_n]]$ を返す。すなわち $d = c_1x_1 + \dots + c_nx_n$ が成り立ち d は x_1, \dots, x_n の最大公約数である。

integers は int または long 型のリストである。
この関数は $[d, [c_1, \dots, c_n]]$ を返す。 d と c_i は整数。

3.8.6 coprime – coprime check

`coprime(a: integer, b: integer) → bool`

a と b が互いに素であれば True を返し、さもなければ False を返す。

a と b は int または long 型、または **Integer**。

3.8.7 pairwise_coprime – coprime check of many integers

`pairwise_coprime(integers: list) → bool`

integers が互いに素ならば True を、さもなければ False を返す。

integers は int または long 型、または **Integer** のリスト。

Examples

```
>>> gcd.gcd(12, 18)
6
>>> gcd.gcd(12, -18)
6
>>> gcd.gcd(-12, -18)
6
>>> gcd.extgcd(12, -18)
(-1, -1, 6)
>>> gcd.extgcd(-12, -18)
(1, -1, 6)
>>> gcd.extgcd(0, -18)
(0, -1, 18)
```



```
>>> gcd.lcm(12, 18)
36
>>> gcd.lcm(12, -18)
-36
>>> gcd.gcd_of_list([60, 90, 210])
[30, [-1, 1, 0]]
```

3.9 multiplicative – 乗法的数論関数

このメソッドの全ての関数は、特に断りのない限り自然数のみ受け付ける。

3.9.1 euler – オイラーのファイ関数

`euler(n: integer) → integer`

n と互いに素かつ n よりも小さい数の総数を返す。この関数はよく φ として言及される。

3.9.2 moebius – メビウス関数

`moebius(n: integer) → integer`

この関数は以下のいずれかの値を返す:

- 1 (n が素因数に奇数をもっているとき、)
- 1 (n が素因数に偶数をもっているとき、)
- 0 (n が素因数が平方数を持っているとき、)

この関数はよく μ として言及される。

3.9.3 sigma – 約数の冪の合計

`sigma(m: integer, n: integer) → integer`

n の因数の m 乗を返す。引数 m は零でもよく、因数の数を返す。この関数はよく σ として言及される。

Examples

```
>>> multiplicative.euler(1)
1
>>> multiplicative.euler(2)
1
>>> multiplicative.euler(4)
2
>>> multiplicative.euler(5)
4
>>> multiplicative.moebius(1)
1
>>> multiplicative.moebius(2)
-1
>>> multiplicative.moebius(4)
```

```
0
>>> multiplicative.moebius(6)
1
>>> multiplicative.sigma(0, 1)
1
>>> multiplicative.sigma(1, 1)
1
>>> multiplicative.sigma(0, 2)
2
>>> multiplicative.sigma(1, 3)
4
>>> multiplicative.sigma(1, 4)
7
>>> multiplicative.sigma(1, 6)
12L
>>> multiplicative.sigma(2, 7)
50
```

3.10 prime – 素数判定, 素数生成

3.10.1 trialDivision – 試し割り算

`trialDivision(n: integer, bound: integer/float=0) → True/False`

奇数に対する試し割り算.

`bound` は素数の探索範囲. もし `bound` が与えられ, n の平方根よりも小さいという条件のもと 1 を返せば, それは `bound` 以下に素因数がないことを意味する.

3.10.2 spsp – 強擬素数テスト

`spsp(n: integer, base: integer, s: integer=None, t: integer=None) → True/False`

`base` を基にした強擬素数テスト.

`s` と `t` は $n - 1 = 2^s t$ かつ `t` は奇数, となるような数.

3.10.3 smallSpsp – 小さい数に対する強擬素数テスト

`smallSpsp(n: integer) → True/False`

10^{12} より小さい整数 n に対する強擬素数テスト.

4 回の強擬素数テストによって 10^{12} より小さい整数が素数かどうか決定するには十分なものである.

3.10.4 miller – Miller の素数判定

`miller(n: integer) → True/False`

Miller の素数判定.

このテストは GRH のもと有効です. [config](#) を見てください.

3.10.5 millerRabin – Miller-Rabin の素数判定

`millerRabin(n: integer, times: integer=20) → True/False`

Miller の素数判定.

miller との違いは, Miller-Rabin メソッドは早いが高確率的なアルゴリズムであり, 一方で, **miller** は GRH のもと決定性アルゴリズムとなる.

`times` (初期設定は 20) は繰り返しの数. エラーの確率は多くても 4^{-times} となる.

3.10.6 `lpsp` – Lucas テスト

`lpsp(n: integer, a: integer, b: integer) → True/False`

Lucas 擬素数テスト.

もし `n` がパラメータ `a` と `b` の, すなわち $x^2 - ax + b$ についての, Lucas 擬素数なら `True` を返す.

3.10.7 `fpsp` – Frobenius テスト

`fpsp(n: integer, a: integer, b: integer) → True/False`

Frobenius 擬素数テスト.

もし `n` がパラメータ `a` と `b` の, すなわち $x^2 - ax + b$ についての, Frobenius 擬素数なら `True` を返す.

3.10.8 `by_primitive_root` – Lehmer’s test

`by_primitive_root(n: integer, divisors: sequence) → True/False`

Lehmer の素数判定法 [17].

`n` が素数のときかつそのときに限り `True` を返す.
このメソッドは原始根の存在に基づいて `n` が素数であることを示す. このために $n - 1$ の素因数を知っていることが必要である.

`divisors` は $n - 1$ の素因数のシーケンス (list, tuple, etc).

3.10.9 `full_euler` – Brillhart & Selfridge’s test

`full_euler(n: integer, divisors: sequence) → True/False`

Brillhart & Selfridge の素数判定法 [12].

このメソッドは $\varphi(n) = n - 1$ の成立により `n` が素数であることを示す, ただし φ はオイラーのファイ関数 (**euler** を参照). このために $n - 1$ の素因数を知っ

ていることが必要である.

`divisors` は $n - 1$ の素因数のシーケンス (list, tuple, etc).

3.10.10 `apr` – Jacobi 和テスト

`apr(n: integer) → True/False`

APR (Adleman-Pomerance-Rumery) 素数判定または Jacobi 和テストと呼ばれる判定法.

`n` は 32 より小さい素因数がないと仮定する. また `n` がいくつかの底に対する `spsp` (強擬素数テスト) を通過したと仮定する.

3.10.11 `primeq` – 自動的な素数判定

`primeq(n: integer) → True/False`

素数判定に対する便利な関数.

`n` のサイズに依存して `trialDivision`, `smallSpsp` または `apr` を使う.

3.10.12 `prime` – n 番目の素数

`prime(n: integer) → integer`

`n` 番目の素数を返す.

3.10.13 `nextPrime` – 次の素数を生成

`nextPrime(n: integer) → integer`

与えられた整数 `n` より大きい数の中で, 最も小さい素数を返す.

3.10.14 `randPrime` – ランダムに素数を生成

`randPrime(n: integer) → integer`

10 進 `n` 桁の素数をランダムに返す.

3.10.15 generator – 素数生成

`generator((None)) → generator`

2 から ∞ までの素数を生成する (ジェネレータとして).

3.10.16 generator_eratosthenes – Eratosthenes の篩を使っている素数生成

`generator_eratosthenes(n: integer) → generator`

Eratosthenes の篩を使って n までの素数を順に生成する.

3.10.17 primonial – 素数の積

`primonial(p: integer) → integer`

以下の積を返す

$$\prod_{q \in \mathbb{P} \leq p} q = 2 \cdot 3 \cdot 5 \cdots p .$$

3.10.18 properDivisors – 真の約数

`properDivisors(n: integer) → list`

n の真の約数を返す (1 と n を除いた n の全ての約数).

小さな素数の積に対してのみ役に立つ. より一般的な場合には `proper_divisors` を使用.

出力は全ての真の約数のリスト.

3.10.19 primitive_root – 平方根

`primitive_root(p: integer) → integer`

p の平方根を返す.

p は奇素数でなければならない.

3.10.20 Lucas_chain – Lucas 数列

```
Lucas_chain(n: integer, f: function, g: function, x_0: integer, x_1: integer)
    → (integer, integer)
```

以下のように定義される数列 $\{x_i\}$ に対する値 (x_n, x_{n+1}) を返す:

$$\begin{aligned}x_{2i} &= f(x_i) \\ x_{2i+1} &= g(x_i, x_{i+1}),\end{aligned}$$

初項は x_0, x_1 .

f は 1 変数の整数関数. g は 2 変数の整数関数.

Examples

```
>>> prime.primeq(131)
True
>>> prime.primeq(133)
False
>>> g = prime.generator()
>>> g.next()
2
>>> g.next()
3
>>> prime.prime(10)
29
>>> prime.nextPrime(100)
101
>>> prime.primitive_root(23)
5
```


3.11 prime_decomp – 素イデアル分解

3.11.1 prime_decomp – 素イデアル分解

`prime_decomp(p: Integer, polynomial: list) → list`

数体 $\mathbb{Q}[x]/(\text{polynomial})$ 上のイデアル (p) の素イデアル分解を返す.

p は (有理) 素数であるべきである. polynomial はモニック既約多項式を定義する整数のリストであるべきである.

このメソッドは (P_k, e_k, f_k) のリストを返す.

P_k は (p) を割る素イデアルを表す **Ideal_with_generator** のインスタンスで, e_k は P_k の分岐指数で, f_k は P_k の剰余次数.

Examples

```
>>> for fact in prime_decomp.prime_decomp(3,[1,9,0,1]):
...     print fact
...
(Ideal_with_generator([BasicAlgNumber([[3, 0, 0], 1], [1, 9, 0, 1]), BasicAlgNumber([[7L, 20L, 4L], 3L], [1, 9, 0, 1])]), 1, 1)
(Ideal_with_generator([BasicAlgNumber([[3, 0, 0], 1], [1, 9, 0, 1]), BasicAlgNumber([[10L, 20L, 4L], 3L], [1, 9, 0, 1])]), 2, 1)
```

3.12 quad – 虚二次体

- Classes
 - **ReducedQuadraticForm**
 - **ClassGroup**
- Functions
 - **class_formula**
 - **class_number**
 - **class_group**
 - **class_number_bsgs**
 - **class_group_bsgs**

3.12.1 ReducedQuadraticForm – 簡約二次形式クラス

Initialize (Constructor)

ReducedQuadraticForm(f: list, unit: list) → *ReducedQuadraticForm*

ReducedQuadraticForm オブジェクトを作成.

f, unit は 3 整数 [a, b, c] のリストでなければならず, 二次形式を $ax^2 + bxy + cy^2$ と表記. unit は単元形式を表す.

Operations

operator	explanation
M * N	M と N の合成を返す.
M ** a	M の a 乗を返す.
M / N	二次形式の除算.
M == N	M と N が等しいかどうか返す.
M != N	M と N が等しくないかどうか返す.

Methods

3.12.1.1 inverse

`inverse(self)` → *ReducedQuadraticForm*

`self` の逆元を返す.

3.12.1.2 disc

`disc(self)` → *ReducedQuadraticForm*

`self` の判別式を返す.

3.12.2 ClassGroup – 類群クラス

Initialize (Constructor)

`ClassGroup(disc: integer, cl: integer, element: integer=None)`
→ *ClassGroup*

`ClassGroup` オブジェクトを作成.

Methods

3.12.3 class_formula

```
class_formula(d: integer, uprbd: integer) → integer
```

類数公式を使い, 判別式 d を持つ類数 h の近似値を返す.

$$\text{類数公式 } h = \frac{\sqrt{|d|}}{\pi} \prod_p \left(1 - \left(\frac{d}{p} \right) \frac{1}{p} \right)^{-1}.$$

入力する数 d は int 型, long 型 または **Integer** でなければならない.

3.12.4 class_number

```
class_number(d: integer, limit_of_d: integer=1000000000)
→ integer
```

簡約形式の数を数えることにより判別式 d を持つ類数を返す.

d は基本判別式とは限らない.

入力する数 d は int 型, long 型 または **Integer** でなければならない.

3.12.5 class_group

```
class_group(d: integer, limit_of_d: integer=1000000000)
→ integer
```

簡約形式の数を数えることにより判別式 d を持つ類数と類群を返す.

d は基本判別式とは限らない.

入力する数 d は int 型, long 型 または **Integer** でなければならない.

3.12.6 class_number_bsgs

```
class_number_bsgs(d: integer) → integer
```

Baby-step Giant-step アルゴリズムを使い, 判別式 d を持つ類数を返す.

d は基本判別式とは限らない.

入力する数 d は int 型, long 型 または **Integer** でなければならない.

3.12.7 class_group_bsgs

```
class_group_bsgs(d: integer, cl: integer, qin: list)
    → integer
```

判別式 disc を持つ位数 p^{exp} の類群の構造を返す。 $qin = [p, exp]$ である。

入力する数 d, cl は `int` 型, `long` 型 または **Integer** でなければならない。

Examples

```
>>> quad.class_formula(-1200, 100000)
12
>>> quad.class_number(-1200)
12
>>> quad.class_group(-1200)
(12, [ReducedQuadraticForm(1, 0, 300), ReducedQuadraticForm(3, 0, 100),
ReducedQuadraticForm(4, 0, 75), ReducedQuadraticForm(12, 0, 25),
ReducedQuadraticForm(7, 2, 43), ReducedQuadraticForm(7, -2, 43),
ReducedQuadraticForm(16, 4, 19), ReducedQuadraticForm(16, -4, 19),
ReducedQuadraticForm(13, 10, 25), ReducedQuadraticForm(13, -10, 25),
ReducedQuadraticForm(16, 12, 21), ReducedQuadraticForm(16, -12, 21)])
>>> quad.class_number_bsgs(-1200)
12L
>>> quad.class_group_bsgs(-1200, 12, [3, 1])
([ReducedQuadraticForm(16, -12, 21)], [[3L]])
>>> quad.class_group_bsgs(-1200, 12, [2, 2])
([ReducedQuadraticForm(12, 0, 25), ReducedQuadraticForm(4, 0, 75)],
[[2L], [2L, 0]])
```

3.13 round2 – the round 2 method

- **Classes**
 - **ModuleWithDenominator**
- **Functions**
 - **round2**
 - **Dedekind**

The round 2 method is for obtaining the maximal order of a number field from an order generated by a root of a defining polynomial of the field.

This implementation of the method is based on [13](Algorithm 6.1.8) and [21](Chapter 3).

3.13.1 ModuleWithDenominator – bases of \mathbb{Z} -module with denominator.

Initialize (Constructor)

ModuleWithDenominator(basis: *list*, denominator: *integer*, **hints: *dict*)

→ *ModuleWithDenominator*

This class represents bases of \mathbb{Z} -module with denominator. It is not a general purpose \mathbb{Z} -module, you are warned. basis is a list of integer sequences.

denominator is a common denominator of all bases.

† Optionally you can supply keyword argument dimension if you would like to postpone the initialization of basis.

Operations

operator	explanation
$A + B$	sum of two modules
$a * B$	scalar multiplication
B / d	divide by an integer

Methods

3.13.1.1 `get_rationals` – get the bases as a list of rationals

`get_rationals(self)` → *list*

Return a list of lists of rational numbers, which is bases divided by denominator.

3.13.1.2 `get_polynomials` – get the bases as a list of polynomials

`get_polynomials(self)` → *list*

Return a list of rational polynomials, which is made from bases divided by denominator.

3.13.1.3 `determinant` – determinant of the bases

`determinant(self)` → *list*

Return determinant of the bases (bases ought to be of full rank and in Hermite normal form).

3.13.2 round2(function)

round2(minpoly_coeff: list) → (list, integer)

Return integral basis of the ring of integers of a field with its discriminant. The field is given by a list of integers, which is a polynomial of generating element θ . The polynomial ought to be monic, in other word, the generating element ought to be an algebraic integer.

The integral basis will be given as a list of rational vectors with respect to θ .

3.13.3 Dedekind(function)

**Dedekind(minpoly_coeff: list, p: integer, e: integer)
→ (bool, ModuleWithDenominator)**

This is the Dedekind criterion.

minpoly_coeff is an integer list of the minimal polynomial of θ .

$p**e$ divides the discriminant of the minimal.

The first element of the returned tuple is whether the computation about p is finished or not.

3.14 squarefree – Squarefreeness tests

There are two method groups. A function in one group raises **Undetermined** when it cannot determine squarefreeness. A function in another group returns **None** in such cases. The latter group of functions have “_ternary” suffix on their names. We refer a set $\{\text{True}, \text{False}, \text{None}\}$ as *ternary*.

The parameter type *integer* means either *int*, *long* or **Integer**.

This module provides an exception class.

Undetermined : Report undetermined state of calculation. The exception will be raised by **lenstra** or **trivial_test**.

3.14.1 Definition

We define squarefreeness as:

n is squarefree \iff there is no prime p whose square divides n .

Examples:

- 0 is non-squarefree because any square of prime can divide 0.
- 1 is squarefree because there is no prime dividing 1.
- 2, 3, 5, and any other primes are squarefree.
- 4, 8, 9, 12, 16 are non-squarefree composites.
- 6, 10, 14, 15, 21 are squarefree composites.

3.14.2 lenstra – Lenstra’s condition

lenstra(n : *integer*) \rightarrow *bool*

If return value is True, n is squarefree. Otherwise, the squarefreeness is still unknown and **Undetermined** is raised. The algorithm is based on [18].

†The condition is so strong that it seems n has to be a prime or a Carmichael number to satisfy it.

Input parameter n ought to be an odd **integer**.

3.14.3 trial_division – trial division

trial_division(n : *integer*) \rightarrow *bool*

Check whether n is squarefree or not.

The method is a kind of trial division and inefficient for large numbers.

Input parameter `n` ought to be an **integer**.

3.14.4 `trivial_test` – trivial tests

`trivial_test(n: integer) → bool`

Check whether `n` is squarefree or not. If the squarefreeness is still unknown, then **Undetermined** is raised.

This method do anything but factorization including Lenstra's method.

Input parameter `n` ought to be an odd **integer**.

3.14.5 `viafactor` – via factorization

`viafactor(n: integer) → bool`

Check whether `n` is squarefree or not.

It is obvious that if one knows the prime factorization of the number, he/she can tell whether the number is squarefree or not.

Input parameter `n` ought to be an **integer**.

3.14.6 `viadecomposition` – via partial factorization

`viadecomposition(n: integer) → bool`

Test the squarefreeness of `n`. The return value is either one of **True** or **False**; **None** never be returned.

The method uses partial factorization into squarefree parts, if such partial factorization is possible. In other cases, It completely factor `n` by trial division.

Input parameter `n` ought to be an **integer**.

3.14.7 `lenstra_ternary` – Lenstra's condition, ternary version

`lenstra_ternary(n: integer) → ternary`

Test the squarefreeness of `n`. The return value is one of the ternary logical constants. If return value is **True**, `n` is squarefree. Otherwise, the squarefreeness is still unknown and **None** is returned.

†The condition is so strong that it seems n has to be a prime or a Carmichael number to satisfy it.

This is a ternary version of **lenstra**.

Input parameter n ought to be an odd **integer**.

3.14.8 **trivial_test_ternary** – trivial tests, ternary version

trivial_test_ternary(n : *integer*) \rightarrow *ternary*

Test the squarefreeness of n . The return value is one of the ternary logical constants.

The method uses a series of trivial tests including **lenstra_ternary**.

This is a ternary version of **trivial_test**.

Input parameter n ought to be an **integer**.

3.14.9 **trial_division_ternary** – trial division, ternary version

trial_division_ternary(n : *integer*) \rightarrow *ternary*

Test the squarefreeness of n . The return value is either one of **True** or **False**; **None** never be returned.

The method is a kind of trial division.

This is a ternary version of **trial_division**.

Input parameter n ought to be an **integer**.

3.14.10 **viafactor_ternary** – via factorization, ternary version

viafactor_ternary(n : *integer*) \rightarrow *ternary*

Just for symmetry, this function is defined as an alias of **viafactor**.

Input parameter n ought to be an **integer**.

Chapter 4

Classes

4.1 `algfield` – Algebraic Number Field

- **Classes**

- `NumberField`
- `BasicAlgNumber`
- `MatAlgNumber`

- **Functions**

- `changetype`
- `disc`
- `fppoly`
- `qpoly`
- `zpoly`

4.1.1 `NumberField` – number field

Initialize (Constructor)

```
NumberField( f: list, precompute: bool=False ) → NumberField
```

Create `NumberField` object.

This field defined by the polynomial `f`.
The class inherits `Field`.

`f`, which expresses coefficients of a polynomial, must be a list of integers. `f` should be written in ascending order. `f` must be monic irreducible over rational

field.

If `precompute` is `True`, all solutions of `f` (by `getConj`), the discriminant of `f` (by `disc`), the signature (by `signature`) and the field discriminant of the basis of the integer ring (by `integer_ring`) are precomputed.

Attributes

degree : The (absolute) extension degree of the number field.

polynomial : The defining polynomial of the number field.

Operations

operator	explanation
<code>K * F</code>	Return the composite field of <code>K</code> and <code>F</code> .
<code>K == F</code>	Check whether the equality of <code>K</code> and <code>F</code> .

Examples

```
>>> K = algfield.NumberField([-2, 0, 1])
>>> L = algfield.NumberField([-3, 0, 1])
>>> print K, L
NumberField([-2, 0, 1]) NumberField([-3, 0, 1])
>>> print K * L
NumberField([1L, 0L, -10L, 0L, 1L])
```

Methods

4.1.1.1 `getConj` – roots of polynomial

`getConj(self)` → *list*

Return all (approximate) roots of the `self.polynomial`.

The output is a list of (approximate) complex number.

4.1.1.2 `disc` – polynomial discriminant

`disc(self)` → *integer*

Return the (polynomial) discriminant of the `self.polynomial`.

†The output is not discriminant of the number field itself.

4.1.1.3 `integer_ring` – integer ring

`integer_ring(self)` → **FieldSquareMatrix**

Return a basis of the ring of integers of `self`.

†The function uses **round2**.

4.1.1.4 `field_discriminant` – discriminant

`field_discriminant(self)` → **Rational**

Return the field discriminant of `self`.

†The function uses **round2**.

4.1.1.5 `basis` – standard basis

`basis(self, j: integer)` → **BasicAlgNumber**

Return the j -th basis (over the rational field) of `self`.

Let θ be a solution of `self.polynomial`. Then θ^j is a part of basis of `self`, so

the method returns them. This basis is called “standard basis” or “power basis”.

4.1.1.6 `signature` – signature

`signature(self) → list`

Return the signature of `self`.

†The method uses Strum’s algorithm.

4.1.1.7 `POLRED` – polynomial reduction

`POLRED(self) → list`

Return some polynomials defining subfields of `self`.

†“POLRED” means “polynomial reduction”. That is, it finds polynomials whose coefficients are not so large.

4.1.1.8 `isIntBasis` – check integral basis

`isIntBasis(self) → bool`

Check whether power basis of `self` is also an integral basis of the field.

4.1.1.9 `isGaloisField` – check Galois field

`isGaloisField(self) → bool`

Check whether the extension `self` over the rational field is Galois.

†As it stands, it only checks the signature.

4.1.1.10 `isFieldElement` – check field element

**`isFieldElement(self, A: BasicAlgNumber/MatAlgNumber)
→ bool`**

Check whether `A` is an element of the field `self`.

4.1.1.11 `getCharacteristic` – characteristic

`getCharacteristic(self) → integer`

Return the characteristic of `self`.

It returns always zero. The method is only for ensuring consistency.

4.1.1.12 `createElement` – create an element

`createElement(self, seed: list) → BasicAlgNumber/MatAlgNumber`

Return an element of `self` with `seed`.

`seed` determines the class of returned element.

For example, if `seed` forms as $[[e_1, e_2, \dots, e_n], d]$, then it calls **BasicAlgNumber**.

Examples

```
>>> K = algfield.NumberField([3, 0, 1])
>>> K.getConj()
[-1.7320508075688774j, 1.7320508075688772j]
>>> K.disc()
-12L
>>> print K.integer_ring()
1/1 1/2
0/1 1/2
>>> K.field_discriminant()
Rational(-3, 1)
>>> K.basis(0), K.basis(1)
BasicAlgNumber([[1, 0], 1], [3, 0, 1]) BasicAlgNumber([[0, 1], 1], [3, 0, 1])
>>> K.signature()
(0, 1)
>>> K.POLRED()
[IntegerPolynomial([(0, 4L), (1, -2L), (2, 1L)], IntegerRing()),
IntegerPolynomial([(0, -1L), (1, 1L)], IntegerRing())]
>>> K.isIntBasis()
False
```

4.1.2 BasicAlgNumber – Algebraic Number Class by standard basis

Initialize (Constructor)

```
BasicAlgNumber( valuelist: list, polynomial: list, precompute:
bool=False )
    → BasicAlgNumber
```

Create an algebraic number with standard (power) basis.

$valuelist = [[e_1, e_2, \dots, e_n], d]$ means $\frac{1}{d}(e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1})$, where θ is a solution of the polynomial polynomial. Note that $\langle \theta^i \rangle$ is a (standard) basis of the field defining by polynomial over the rational field.

e_i, d must be integers. Also, polynomial should be list of integers. If precompute is True, all solutions of polynomial (by **getConj**), approximation values of all conjugates of self (by **getApprox**) and a polynomial which is a solution of self (by **getCharPoly**) are precomputed.

Attributes

value : The list of numerators (the integer part) and the denominator of self.

coeff : The coefficients of numerators (the integer part) of self.

denom : The denominator of the algebraic number for standard basis.

degree : The degree of extension of the field over the rational field.

polynomial : The defining polynomial of the field.

field : The number field in which self is.

Operations

operator	explanation
$a + b$	Return the sum of a and b.
$a - b$	Return the subtraction of a and b.
$- a$	Return the negation of a.
$a * b$	Return the product of a and b.
$a ** k$	Return the k-th power of a.
a / b	Return the quotient of a by b.

Examples

```
>>> a = algfield.BasicAlgNumber([[1, 1], 1], [-2, 0, 1])
>>> b = algfield.BasicAlgNumber([[-1, 2], 1], [-2, 0, 1])
>>> print a + b
BasicAlgNumber([[0, 3], 1], [-2, 0, 1])
>>> print a * b
BasicAlgNumber([[3L, 1L], 1], [-2, 0, 1])
>>> print a ** 3
BasicAlgNumber([[7L, 5L], 1], [-2, 0, 1])
>>> a // b
BasicAlgNumber([[5L, 3L], 7L], [-2, 0, 1])
```

Methods

4.1.2.1 `inverse` – `inverse`

`inverse(self) → BasicAlgNumber`

Return the inverse of `self`.

4.1.2.2 `getConj` – roots of polynomial

`getConj(self) → list`

Return all (approximate) roots of `self.polynomial`.

4.1.2.3 `getApprox` – approximate conjugates

`getApprox(self) → list`

Return all (approximate) conjugates of `self`.

4.1.2.4 `getCharPoly` – characteristic polynomial

`getCharPoly(self) → list`

Return the characteristic polynomial of `self`.

†`self` is a solution of the characteristic polynomial.

The output is a list of integers.

4.1.2.5 `getRing` – the field

`getRing(self) → NumberField`

Return the field which `self` belongs to.

4.1.2.6 `trace` – `trace`

`trace(self) → Rational`

Return the trace of `self` in the `self.field` over the rational field.

4.1.2.7 `norm` – `norm`

`norm(self) → Rational`

Return the norm of `self` in the `self.field` over the rational field.

4.1.2.8 `isAlgInteger` – check (algebraic) integer

`isAlgInteger(self) → bool`

Check whether `self` is an (algebraic) integer or not.

4.1.2.9 `ch_matrix` – obtain `MatAlgNumber` object

`ch_matrix(self) → MatAlgNumber`

Return `MatAlgNumber` object corresponding to `self`.

Examples

```
>>> a = algfield.BasicAlgNumber([[1, 1], 1], [-2, 0, 1])
>>> a.inverse()
BasicAlgNumber([[-1L, 1L], 1L], [-2, 0, 1])
>>> a.getConj()
[(1.4142135623730951+0j), (-1.4142135623730951+0j)]
>>> a.getApprox()
[(2.4142135623730949+0j), (-0.41421356237309515+0j)]
>>> a.getCharPoly()
[-1, -2, 1]
>>> a.getRing()
NumberField([-2, 0, 1])
>>> a.trace(), a.norm()
2 -1
>>> a.isAlgInteger()
True
>>> a.ch_matrix()
MatAlgNumber([1, 1]+[2, 1], [-2, 0, 1])
```

4.1.3 MatAlgNumber – Algebraic Number Class by matrix representation

Initialize (Constructor)

MatAlgNumber(coefficient: list, polynomial: list) → MatAlgNumber

Create an algebraic number represented by a matrix.

“matrix representation” means the matrix A over the rational field such that $(e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1})(1, \theta, \dots, \theta^{n-1})^T = A(1, \theta, \dots, \theta^{n-1})^T$, where t expresses transpose operation.

$coefficient = [e_1, e_2, \dots, e_n]$ means $e_1 + e_2\theta + e_3\theta^2 + \dots + e_n\theta^{n-1}$, where θ is a solution of the polynomial polynomial. Note that $\langle \theta^i \rangle$ is a (standard) basis of the field defining by polynomial over the rational field. coefficient must be a list of (not only integers) rational numbers. polynomial must be a list of integers.

Attributes

coeff : The coefficients of the algebraic number for standard basis.

degree : The degree of extension of the field over the rational field.

matrix : The representation matrix of the algebraic number.

polynomial : The defining polynomial of the field.

field : The number field in which self is.

Operations

operator	explanation
a + b	Return the sum of a and b .
a - b	Return the subtraction of a and b .
- a	Return the negation of a .
a * b	Return the product of a and b .
a ** k	Return the k-th power of a .
a / b	Return the quotient of a by b .

Examples

```
>>> a = algfield.MatAlgNumber([1, 2], [-2, 0, 1])
>>> b = algfield.MatAlgNumber([-2, 3], [-2, 0, 1])
>>> print a + b
MatAlgNumber([-1, 5]+[10, -1], [-2, 0, 1])
>>> print a * b
MatAlgNumber([10, -1]+[-2, 10], [-2, 0, 1])
>>> print a ** 3
MatAlgNumber([25L, 22L]+[44L, 25L], [-2, 0, 1])
>>> print a / b
MatAlgNumber([Rational(1, 1), Rational(1, 2)]+
[Rational(1, 1), Rational(1, 1)], [-2, 0, 1])
```

Methods

4.1.3.1 inverse – inverse

`inverse(self) → MatAlgNumber`

Return the inverse of `self`.

4.1.3.2 getRing – the field

`getRing(self) → NumberField`

Return the field which `self` belongs to.

4.1.3.3 trace – trace

`trace(self) → Rational`

Return the trace of `self` in the `self.field` over the rational field.

4.1.3.4 norm – norm

`norm(self) → Rational`

Return the norm of `self` in the `self.field` over the rational field.

4.1.3.5 ch_basic – obtain BasicAlgNumber object

`ch_basic(self) → BasicAlgNumber`

Return **BasicAlgNumber** object corresponding to `self`.

Examples

```
>>> a = algfield.MatAlgNumber([1, -1, 1], [-3, 1, 2, 1])
>>> a.inverse()
MatAlgNumber([Rational(2, 3), Rational(4, 9), Rational(1, 9)]+
[Rational(1, 3), Rational(5, 9), Rational(2, 9)]+
[Rational(2, 3), Rational(1, 9), Rational(1, 9)], [-3, 1, 2, 1])
>>> a.trace()
Rational(7, 1)
```



```
>>> a.norm()
Rational(27, 1)
>>> a.getRing()
NumberField([-3, 1, 2, 1])
>>> a.ch_basic()
BasicAlgNumber([[1, -1, 1], 1], [-3, 1, 2, 1])
```

4.1.4 `changetype(function)` – obtain `BasicAlgNumber` object

`changetype(a: integer, polynomial: list=[0, 1]) → BasicAlgNumber`

`changetype(a: Rational, polynomial: list=[0, 1]) → BasicAlgNumber`

`changetype(polynomial: list) → BasicAlgNumber`

Return a `BasicAlgNumber` object corresponding to `a`.

If `a` is an integer or an instance of `Rational`, the function returns `BasicAlgNumber` object whose field is defined by polynomial. If `a` is a list, the function returns `BasicAlgNumber` corresponding to a solution of `a`, considering `a` as the polynomial.

The input parameter `a` must be an integer, `Rational` or a list of integers.

4.1.5 `disc(function)` – discriminant

`disc(A: list) → Rational`

Return the discriminant of a_i , where $A = [a_1, a_2, \dots, a_n]$.

a_i must be an instance of `BasicAlgNumber` or `MatAlgNumber` defined over a same number field.

4.1.6 `fppoly(function)` – polynomial over finite prime field

`fppoly(coeffs: list, p: integer) → FinitePrimeFieldPolynomial`

Return the polynomial whose coefficients `coeffs` are defined over the prime field \mathbb{Z}_p .

`coeffs` should be a list of integers or of instances of `FinitePrimeFieldElement`.

4.1.7 `qpoly(function)` – polynomial over rational field

`qpoly(coeffs: list) → FieldPolynomial`

Return the polynomial whose coefficients `coeffs` are defined over the rational

field.

coeffs must be a list of integers or instances of **Rational**.

4.1.8 `zpoly(function)` – polynomial over integer ring

`zpoly(coeffs: list) → IntegerPolynomial`

Return the polynomial whose coefficients coeffs are defined over the (rational) integer ring.

coeffs must be a list of integers.

Examples

```
>>> a = algfield.changetype(3, [-2, 0, 1])
>>> b = algfield.BasicAlgNumber([[1, 2], 1], [-2, 0, 1])
>>> A = [a, b]
>>> algfield.disc(A)
288L
```

4.2 elliptic – elliptic class object

- **Classes**
 - **ECGeneric**
 - **ECoverQ**
 - **ECoverGF**
- **Functions**
 - **EC**

This module using following type:

weierstrassform :

weierstrassform is a list $(a_1, a_2, a_3, a_4, a_6)$ or (a_4, a_6) , it represents E : $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ or $E : y^2 = x^3 + a_4x + a_6$, respectively.

infpoint :

infpoint is the list $[0]$, which represents infinite point on the elliptic curve.

point :

point is two-dimensional coordinate list $[x, y]$ or **infpoint**.

4.2.1 †ECGeneric – generic elliptic curve class

Initialize (Constructor)

```
ECGeneric( coefficient: weierstrassform, basefield: Field=None )  
→ ECGeneric
```

楕円曲線を作る。

The class is for the definition of elliptic curves over general fields. Instead of using this class directly, we recommend that you call **EC**.

†The class precomputes the following values.

- shorter form: $y^2 = b_2x^3 + b_4x^2 + b_6x + b_8$
- shortest form: $y^2 = x^3 + c_4x + c_6$
- discriminant
- j-invariant

All elements of coefficient must be in basefield.

See **weierstrassform** for more information about coefficient. If discriminant of `self` equals 0, it raises `ValueError`.

Attributes

basefield :

It expresses the field which each coordinate of all points in `self` is on. (This means not only `self` is defined over `basefield`.)

ch :

It expresses the characteristic of `basefield`.

infpoint :

It expresses infinity point (i.e. `[0]`).

a1, a2, a3, a4, a6 :

It expresses the coefficients `a1`, `a2`, `a3`, `a4`, `a6`.

b2, b4, b6, b8 :

It expresses the coefficients `b2`, `b4`, `b6`, `b8`.

c4, c6 :

It expresses the coefficients `c4`, `c6`.

disc :

It expresses the discriminant of `self`.

j :
It expresses the j-invariant of **self**.

coefficient :
It expresses the **weierstrassform** of **self**.

Methods

4.2.1.1 simple – simplify the curve coefficient

simple(self) → ECGeneric

Return elliptic curve corresponding to the short Weierstrass form of **self** by changing the coordinates.

4.2.1.2 changeCurve – change the curve by coordinate change

changeCurve(self, V: list) → ECGeneric

Return elliptic curve corresponding to the curve obtained by some coordinate change $x = u^2x' + r$, $y = u^3y' + su^2x' + t$.

For $u \neq 0$, the coordinate change gives some curve which is **basefield**-isomorphic to **self**.

V must be a list of the form $[u, r, s, t]$, where u, r, s, t are in **basefield**.

4.2.1.3 changePoint – change coordinate of point on the curve

changePoint(self, P: point, V: list) → point

Return the point corresponding to the point obtained by the coordinate change $x' = (x - r)u^{-2}$, $y' = (y - s(x - r) + t)u^{-3}$.

Note that the inverse coordinate change is $x = u^2x' + r$, $y = u^3y' + su^2x' + t$. See **changeCurve**.

V must be a list of the form $[u, r, s, t]$, where u, r, s, t are in **basefield**. u must be non-zero.

4.2.1.4 coordinateY – Y-coordinate from X-coordinate

coordinateY(self, x: FieldElement) → FieldElement / False

Return Y-coordinate of the point on **self** whose X-coordinate is **x**.

The output would be one Y-coordinate (if a coordinate is found). If such a Y-coordinate does not exist, it returns False.

4.2.1.5 whetherOn – Check point is on curve

whetherOn(self, P: point) → bool

Check whether the point P is on self or not.

4.2.1.6 add – Point addition on the curve

add(self, P: point, Q: point) → point

Return the sum of the point P and Q on self.

4.2.1.7 sub – Point subtraction on the curve

sub(self, P: point, Q: point) → point

Return the subtraction of the point P from Q on self.

4.2.1.8 mul – Scalar point multiplication on the curve

mul(self, k: integer, P: point) → point

Return the scalar multiplication of the point P by a scalar k on self.

4.2.1.9 divPoly – division polynomial

divPoly(self, m: integer=None) → FieldPolynomial/(f: list, H: integer)

Return the division polynomial.

If m is odd, this method returns the usual division polynomial. If m is even, return the quotient of the usual division polynomial by $2y + a_1x + a_3$.

†If m is not specified (i.e. m=None), then return (f, H). H is the least prime satisfying $\prod_{2 \leq l \leq H, l: \text{prime}} l > 4\sqrt{q}$, where q is the order of **basefield**. f is the list of k-division polynomials up to $k \leq H$. These are used for Schoof's algorithm.

4.2.2 ECoverQ – elliptic curve over rational field

The class is for elliptic curves over the rational field \mathbb{Q} (**RationalField** in `nzmath.rational`).

The class is a subclass of **ECGeneric**.

Initialize (Constructor)

ECoverQ(coefficient: **weierstrassform**) \rightarrow **ECoverQ**

Create elliptic curve over the rational field.

All elements of coefficient must be integer or **Rational**.
See **weierstrassform** for more information about coefficient.

Examples

```
>>> E = elliptic.ECoverQ([rational.Rational(1, 2), 3])
>>> print E.disc
-3896/1
>>> print E.j
1728/487
```

Methods

4.2.2.1 `point` – obtain random point on curve

`point(self, limit: integer=1000) → point`

Return a random point on `self`.

`limit` expresses the time of trying to choose points. If failed, raise `ValueError`.
†Because it is difficult to search the rational point over the rational field, it might raise error with high frequency.

Examples

```
>>> print E.changeCurve([1, 2, 3, 4])
y ** 2 + 6/1 * x * y + 8/1 * y = x ** 3 - 3/1 * x ** 2 - 23/2 * x - 4/1
>>> E.divPoly(3)
FieldPolynomial([(0, Rational(-1, 4)), (1, Rational(36, 1)), (2, Rational(3, 1)
), (4, Rational(3, 1))], RationalField())
```

4.2.3 ECoverGF – elliptic curve over finite field

The class is for elliptic curves over a finite field, denoted by \mathbb{F}_q (**FiniteField** and its subclasses in `nzmath`).

The class is a subclass of **ECGeneric**.

Initialize (Constructor)

```
ECoverGF( coefficient: weierstrassform, basefield: FiniteField )  
→ ECoverGF
```

Create elliptic curve over a finite field.

All elements of coefficient must be in basefield. basefield should be an instance of **FiniteField**.

See **weierstrassform** for more information about coefficient.

Examples

```
>>> E = elliptic.ECoverGF([2, 5], finitefield.FinitePrimeField(11))  
>>> print E.j  
7 in F_11  
>>> E.whetherOn([8, 4])  
True  
>>> E.add([3, 4], [9, 9])  
[FinitePrimeFieldElement(0, 11), FinitePrimeFieldElement(4, 11)]  
>>> E.mul(5, [9, 9])  
[FinitePrimeFieldElement(0, 11)]
```

Methods

4.2.3.1 point – find random point on curve

point(self) → **point**

Return a random point on **self**.

This method uses a probabilistic algorithm.

4.2.3.2 naive – Frobenius trace by naive method

naive(self) → *integer*

Return Frobenius trace t by a naive method.

†The function counts up the Legendre symbols of all rational points on **self**. Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on **self** over **self.basefield** \mathbb{F}_q .

The characteristic of **self.basefield** must not be 2 nor 3.

4.2.3.3 Shanks_Mestre – Frobenius trace by Shanks and Mestre method

Shanks_Mestre(self) → *integer*

Return Frobenius trace t by Shanks and Mestre method.

†This uses the method proposed by Shanks and Mestre. †See Algorithm 7.5.3 of [15] for more information about the algorithm. Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on **self** over **self.basefield** \mathbb{F}_q .

self.basefield must be an instance of **FinitePrimeField**.

4.2.3.4 Schoof – Frobenius trace by Schoof's method

Schoof(self) → *integer*

Return Frobenius trace t by Schoof's method.

†This uses the method proposed by Schoof.

Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on `self` over `self.basefield` \mathbb{F}_q .

4.2.3.5 `trace` – Frobenius trace

`trace(self, r: integer=None) → integer`

Return Frobenius trace t .

Frobenius trace of the curve is t such that $\#E(\mathbb{F}_q) = q + 1 - t$, where $\#E(\mathbb{F}_q)$ stands for the number of points on `self` over `self.basefield` \mathbb{F}_q .

If positive r given, it returns $q^r + 1 - \#E(\mathbb{F}_{q^r})$.

†The method selects algorithms by investigating `self.ch` when `self.basefield` is an instance of `FinitePrimeField`. If `ch` < 1000, the method uses `naive`. If $10^4 < ch < 10^{30}$, the method uses `Shanks_Mestre`. Otherwise, it uses `Schoof`.

The parameter r must be positive integer.

4.2.3.6 `order` – order of group of rational points on the curve

`order(self, r: integer=None) → integer`

Return order $\#E(\mathbb{F}_q) = q + 1 - t$.

If positive r given, this computes $\#E(\mathbb{F}_{q^r})$ instead.

†On the computation of Frobenius trace t , the method calls `trace`.

The parameter r must be positive integer.

4.2.3.7 `pointorder` – order of point on the curve

`pointorder(self, P: point, ord_factor: list=None) → integer`

Return order of a point P .

†The method uses factorization of `order`.

If `ord_factor` is given, computation of factorizing the order of `self` is omitted and it applies `ord_factor` instead.

4.2.3.8 TatePairing – Tate Pairing

TatePairing(self, m: *integer*, P: **point**, Q: **point**) → **FiniteFieldElement**

Return Tate-Lichtenbaum pairing $\langle P, Q \rangle_m$.

†The method uses Miller’s algorithm.
The image of the Tate pairing is $\mathbb{F}_q^*/\mathbb{F}_q^{*m}$, but the method returns an element of \mathbb{F}_q , so the value is not uniquely defined. If uniqueness is needed, use **TatePairing_Extend**.

The point P has to be a m-torsion point (i.e. $mP = [0]$). Also, the number m must divide **order**.

4.2.3.9 TatePairing_Extend – Tate Pairing with final exponentiation

TatePairing_Extend(self, m: *integer*, P: **point**, Q: **point**)
→ **FiniteFieldElement**

Return Tate Pairing with final exponentiation, i.e. $\langle P, Q \rangle_m^{(q-1)/m}$.

†The method calls **TatePairing**.

The point P has to be a m-torsion point (i.e. $mP = [0]$). Also the number m must divide **order**.
The output is in the group generated by m -th root of unity in \mathbb{F}_q^* .

4.2.3.10 WeilPairing – Weil Pairing

WeilPairing(self, m: *integer*, P: **point**, Q: **point**) → **FiniteFieldElement**

Return Weil pairing $e_m(P, Q)$.

†The method uses Miller’s algorithm.

The points P and Q has to be a m-torsion point (i.e. $mP = mQ = [0]$). Also, the number m must divide **order**.

The output is in the group generated by m -th root of unity in \mathbb{F}_q^* .

4.2.3.11 BSGS – point order by Baby-Step and Giant-Step

BSGS(self, P: **point**) → *integer*

Return order of point P by Baby-Step and Giant-Step method.

†See [19] for more information about the algorithm.

4.2.3.12 DLP_BSGS – solve Discrete Logarithm Problem by Baby-Step and Giant-Step

DLP_BSGS(self, n: integer, P: point, Q: point) → m: integer

Return m such that $Q = mP$ by Baby-Step and Giant-Step method.

The points P and Q has to be a n -torsion point (i.e. $nP = nQ = [0]$). Also, the number n must divide **order**. The output m is an integer.

4.2.3.13 structure – structure of group of rational points

structure(self) → structure: tuple

Return the group structure of **self**.

The structure of $E(\mathbb{F}_q)$ is represented as $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$. The method uses **WeilPairing**.

The output **structure** is a tuple of positive two integers (d, n) . d divides n .

4.2.3.14 issupersingular – check supersingular curve

structure(self) → bool

Check whether **self** is a supersingular curve or not.

Examples

```
>>> E=nzmath.elliptic.ECoverGF([2, 5], nzmath.finitefield.FinitePrimeField(11))
>>> E.whetherOn([0, 4])
True
>>> print E.coordinateY(3)
4 in F_11
>>> E.trace()
2
>>> E.order()
10
```

```
>>> E.pointorder([3, 4])
10L
>>> E.TatePairing(10, [3, 4], [9, 9])
FinitePrimeFieldElement(3, 11)
>>> E.DLP_BSGS(10, [3, 4], [9, 9])
6
```


4.2.4 EC(function)

```
EC(coefficient: weierstrassform, basefield: Field)  
    → ECGeneric
```

Create an elliptic curve object.

All elements of coefficient must be in basefield.
basefield must be **RationalField** or **FiniteField** or their subclasses. See also **weierstrassform** for coefficient.

4.3 `finitefield` – Finite Field

- **Classes**
 - **`FiniteField`**
 - **`FiniteFieldElement`**
 - **`FinitePrimeField`**
 - **`FinitePrimeFieldElement`**
 - **`ExtendedField`**
 - **`ExtendedFieldElement`**

4.3.1 †FiniteField – finite field, abstract

有限体のクラスについて考える。直接的にクラスを扱うのではなく、**FinitePrimeField** や **ExtendedField** のサブクラスとして扱う。

クラスとは **Field** のサブクラスのことである。

4.3.2 †FiniteFieldElement – element in finite field, abstract

有限体の要素のクラスについて考える。直接的にクラスを扱うのではなく、**FinitePrimeFieldElement** や **ExtendedFieldElement** のサブクラスとして扱う。

クラスとは **Field** のサブクラスのことである。

4.3.3 FinitePrimeField – finite prime field

Finite prime field is also known as \mathbb{F}_p or $\text{GF}(p)$. It has prime number cardinality.

The class is a subclass of **FiniteField**.

Initialize (Constructor)

FinitePrimeField(characteristic: *integer*) \rightarrow *FinitePrimeField*

Create a FinitePrimeField instance with the given characteristic. characteristic must be positive prime integer.

Attributes

zero :

It expresses the additive unit 0. (read only)

one :

It expresses the multiplicative unit 1. (read only)

Operations

operator	explanation
F==G	equality test.
x in F	membership test.
card(F)	Cardinality of the field.

Methods

4.3.3.1 createElement – create element of finite prime field

`createElement(self, seed: integer) → FinitePrimeFieldElement`

seed の **FinitePrimeFieldElement** を作る。
seed は int 型か long 型。

4.3.3.2 getCharacteristic – get characteristic

`getCharacteristic(self) → integer`

体の標数の値を返す。

4.3.3.3 issubring – subring test

`issubring(self, other: Ring) → bool`

他の環が部分環として体に含まれているか教えてくれる。

4.3.3.4 issuperring – superring test

`issuperring(self, other: Ring) → bool`

Report whether the field is a superring of another ring.
Since the field is a prime field, it can be a superring of itself only.

4.3.4 FinitePrimeFieldElement – element of finite prime field

The class provides elements of finite prime fields.

It is a subclass of **FiniteFieldElement** and **IntegerResidueClass**.

Initialize (Constructor)

FinitePrimeFieldElement(representative: *integer*, modulus: *integer*)
→ *FinitePrimeFieldElement*

Create element in finite prime field of modulus with residue representative.
modulus は正の素数の整数である。

Operations

operator	explanation
a+b	addition.
a-b	subtraction.
a*b	multiplication.
a**n, pow(a,n)	power.
-a	negation.
+a	make a copy.
a==b	equality test.
a!=b	inequality test.
repr(a)	return representation string.
str(a)	return string.

Methods

4.3.4.1 `getRing` – get ring object

`getRing(self)` \rightarrow *FinitePrimeField*

Return an instance of `FinitePrimeField` to which the element belongs.

4.3.4.2 `order` – order of multiplicative group

`order(self)` \rightarrow *integer*

\mathbb{F}_p の乗法群の要素の配列を返す。

4.3.5 ExtendedField – extended field of finite field

ExtendedField is a class for finite field, whose cardinality $q = p^n$ with a prime p and $n > 1$. It is usually called \mathbb{F}_q or $\text{GF}(q)$.

The class is a subclass of **FiniteField**.

Initialize (Constructor)

ExtendedField(basefield: *FiniteField*, modulus: *FiniteFieldPolynomial*)
→ *ExtendedField*

体の拡張を行う。basefield[X]/(modulus(X)).

与えられた characteristic の有限素体のインスタンス。The modulus は basefield 上の係数をもつ既約多項式でなければならない。

Attributes

zero :

It expresses the additive unit 0. (read only)

one :

It expresses the multiplicative unit 1. (read only)

Operations

operator	explanation
F==G	equality or not.
x in F	membership test.
card(F)	Cardinality of the field.
repr(F)	representation string.
str(F)	string.

Methods

4.3.5.1 createElement – create element of extended field

`createElement(self, seed: extended element seed) → ExtendedFieldElement`

シードから体の要素を作る。その結果は **ExtendedFieldElement** のインスタンスである。

seed が成りうるのは:

- a **FinitePrimeFieldPolynomial**
- an integer, which will be expanded in `card(basefield)` and interpreted as a polynomial.
- basefield element.
- 多項式の係数としてベースフィールドの要素が並びリスト。

4.3.5.2 getCharacteristic – get characteristic

`getCharacteristic(self) → integer`

体の標数の値を返す。

4.3.5.3 issubring – subring test

`issubring(self, other: Ring) → bool`

他の環が部分環として体を含んでいるか教えてくれる。

4.3.5.4 issuperring – superring test

`issuperring(self, other: Ring) → bool`

Report whether the field is a superring of another ring.

4.3.5.5 primitive_element – generator of multiplicative group

`primitive_element(self) → ExtendedFieldElement`

体の原始元の値を返す。

4.3.6 ExtendedFieldElement – element of finite field

ExtendedFieldElement is a class for an element of F_q .

The class is a subclass of **FiniteFieldElement**.

Initialize (Constructor)

ExtendedFieldElement(representative: *FiniteFieldPolynomial*, field: *ExtendedField*)
→ *ExtendedFieldElement*

有限拡張体の要素を作る。

representative must be an **FiniteFieldPolynomial** has same basefield. field は拡張体のインスタンス。

Operations

operator	explanation
a+b	addition.
a-b	subtraction.
a*b	multiplication.
a/b	inverse multiplication.
a**n, pow(a,n)	power.
-a	negation.
+a	make a copy.
a==b	equality test.
a!=b	inequality test.
repr(a)	return representation string.
str(a)	return string.

Methods

4.3.6.1 getRing – get ring object

`getRing(self)` \rightarrow *FinitePrimeField*

ある要素が入っている有限素体のインスタンスを返す。

4.3.6.2 inverse – inverse element

`inverse(self)` \rightarrow *ExtendedFieldElement*

逆元の値を返す。

4.4 group – algorithms for finite groups

- Classes
 - **Group**
 - **GroupElement**
 - **GenerateGroup**
 - **AbelianGenerate**

4.4.1 †Group – group structure

Initialize (Constructor)

Group(value: class, operation: int=-1) → Group

Create an object which wraps value (typically a ring or a field) only to expose its group structure.

The instance has methods defined for (abstract) group. For example, **identity** returns the identity element of the group from wrapped value.

value must be an instance of a class expresses group structure. operation must be 0 or 1; If operation is 0, value is regarded as the additive group. On the other hand, if operation is 1, value is considered as the multiplicative group. The default value of operation is 0.

†You can input an instance of **Group** itself as value. In this case, the default value of operation is the attribute **operation** of the instance.

Attributes

entity :

The wrapped object.

operation :

It expresses the mode of operation; 0 means additive, while 1 means multiplicative.

Operations

operator	explanation
A==B	Return whether A and B are equal or not.
A!=B	Check whether A and B are not equal.
repr(A)	representation
str(A)	simple representation

Examples

```
>>> G1=group.Group(finitefield.FinitePrimeField(37), 1)
>>> print G1
F_37
>>> G2=group.Group(intresidue.IntegerResidueClassRing(6), 0)
```

```
>>> print G2  
Z/6Z
```

Methods

4.4.1.1 setOperation – change operation

setOperation(self, operation: int) → (None)

群のタイプを加法 (0) または乗法 (1) に変える。

operation は 0 または 1。

4.4.1.2 †createElement – generate a GroupElement instance

createElement(self, *value) → GroupElement

Return **GroupElement** object whose group is self, initialized with value.

† この方法は self と呼ぶ。linkingtwogroupGroupentity.createElement.

value must fit the form of argument for self.**entity**.createElement.

4.4.1.3 †identity – identity element

identity(self) → GroupElement

群の単位元の値を返す。

operation によって 0 (加法) または 1 (乗法) を返す。† この方法は param-self.**entity** と呼ばれている。identity または **entity** が属性をもたないときは 0 か 1 を返す。

4.4.1.4 grouporder – order of the group

grouporder(self) → long

paramself の要素の個数の値を返す。.

† この方法は self と呼ばれている。**entity**.grouporder, card or `__len__`.
ここではこの群は有限と考え、返す値は long 型の整数である。もしこの群が無
限の場合、この方法では出力は定義できない。

Examples

```
>>> G1=group.Group(finitefield.FinitePrimeField(37), 1)
>>> G1.grouporder()
36
>>> G1.setOperation(0)
>>> print G1.identity()
FinitePrimeField,0 in F_37
>>> G1.grouporder()
37
```

4.4.2 GroupElement – elements of group structure

Initialize (Constructor)

GroupElement(value: *class*, operation: *int*=-1) → **GroupElement**

Create an object which wraps value (typically a ring element or a field element) to make it behave as an element of group.

The instance has methods defined for an (abstract) element of group. For example, **inverse** returns the inverse element of value as the element of group object.

value must be an instance of a class expresses an element of group structure. operation must be 0 or 1; If operation is 0, value is regarded as the additive group. On the other hand, if operation is 1, value is considered as the multiplicative group. The default value of operation is 0.

†You can input an instance of **GroupElement** itself as value. In this case, the default value of operation is the attribute **operation** of the instance.

Attributes

entity :

The wrapped object.

set :

It is an instance of **Group**, which expresses the group to which **self** belongs.

operation :

It expresses the mode of operation; 0 means additive, while 1 means multiplicative.

Operations

operator	explanation
A==B	Return whether A and B are equal or not.
A!=B	Check whether A and B are not equal.
A.ope(B)	Basic operation (additive +, multiplicative *)
A.ope2(n)	Extended operation (additive *, multiplicative **)
A.inverse()	Return the inverse element of self
repr(A)	representation
str(A)	simple representation

Examples

```
>>> G1=group.GroupElement(finitefield.FinitePrimeFieldElement(18, 37), 1)
>>> print G1
FinitePrimeField,18 in F_37
>>> G2=group.Group(intresidue.IntegerResidueClass(3, 6), 0)
IntegerResidueClass(3, 6)
```

Methods

4.4.2.1 setOperation – change operation

`setOperation(self, operation: int) → (None)`

群のタイプを加法 (0) または乗法 (1) に変える。

operation は 0 か 1。

4.4.2.2 †getGroup – generate a Group instance

`getGroup(self) → Group`

Return **Group** object to which self belongs.

†This method calls self.**entity**.getRing or getGroup.

†In an initialization of **GroupElement**, the attribute **set** is set as the value returned from the method.

4.4.2.3 order – order by factorization method

`order(self) → long`

self の位数の値を返す。

† この方法は群の位数の因数分解を使う。

† ここではこの群は有限と考え、返す値は long 型の整数である。† もしこの群が無限ならば、この方法はエラーを返すか有効でない値を返す。

4.4.2.4 t_order – order by baby-step giant-step

`t_order(self, v: int=2) → long`

self の位数の値を返す。

† この方法は Terry's baby-step giant-step algorithm を使う。

この方法は群の位数を使わない。v に baby-step の数を入れる。† ここではこの群は有限と考え、返す値は long 型の整数である。† もしこの群が無限ならば、この方法はエラーを返すか有効でない値を返す。

v は int 型の整数。

Examples

```
>>> G1=group.GroupElement(finitefield.FinitePrimeFieldElement(18, 37), 1)
>>> G1.order()
36
>>> G1.t_order()
36
```

4.4.3 †GenerateGroup – group structure with generator

Initialize (Constructor)

GenerateGroup(value: *class*, operation: *int*=-1) → GroupElement

Create an object which is generated by value as the element of group structure.

This initializes a group ‘including’ the group elements, not a group with generators, now. We do not recommend using this module now. The instance has methods defined for an (abstract) element of group. For example, **inverse** returns the inverse element of value as the element of group object. The class inherits the class **Group**.

value must be a list of generators. Each generator should be an instance of a class expresses an element of group structure. operation must be 0 or 1; If operation is 0, value is regarded as the additive group. On the other hand, if operation is 1, value is considered as the multiplicative group. The default value of operation is 0.

Examples

```
>>> G1=group.GenerateGroup([intresidue.IntegerResidueClass(2, 20),
... intresidue.IntegerResidueClass(6, 20)])
>>> G1.identity()
intresidue.IntegerResidueClass(0, 20)
```

4.4.4 AbelianGenerate – abelian group structure with generator

Initialize (Constructor)

GenerateGroup のクラスを継承する。

4.4.4.1 relationLattice – relation between generators

relationLattice(self) → **Matrix**

格子原理関係にある数のリストを返す。as a square matrix each of whose column vector is a relation basis.

関係の原理 V は $\prod_i \text{generator}_i V_i = 1$ を充たす。

4.4.4.2 computeStructure – abelian group structure

computeStructure(self) → **tuple**

有限アーベル群構造を計算する。

もし $\text{self } G \simeq \oplus_i \langle h_i \rangle$ で、 $[(h_1, \text{ord}(h_1)), \dots, (h_n, \text{ord}(h_n))]$ と $\#G$ を返す。
 $\langle h_i \rangle$ は変数 h_i の巡回群。

出力は二つずつ要素を持つ三つの組である。; 最初の要素は h_i とその位数のリストである。; また、二番目の要素は群の位数である。

Examples

```
>>> G=AbelianGenerate([intresidue.IntegerResidueClass(2, 20),
... intresidue.IntegerResidueClass(6, 20)])
>>> G.relationLattice()
10 7
0 1
>>> G.computeStructure()
([IntegerResidueClassRing,IntegerResidueClass(2, 20), 10]), 10L)
```

4.5 imaginary – complex numbers and its functions

このモジュール `imaginary` では複素数に扱う。この関数は主に `cmath` 標準モジュールと対応している。

- **Classes**

- **ComplexField**
- **Complex**
- †**ExponentialPowerSeries**
- †**AbsoluteError**
- †**RelativeError**

- **Functions**

- **exp**
- **expi**
- **log**
- **sin**
- **cos**
- **tan**
- **sinh**
- **cosh**
- **tanh**
- **atanh**
- **sqrt**

このモジュールは以下の内容も扱う。:

e :
This constant is obsolete (Ver 1.1.0).

pi :
This constant is obsolete (Ver 1.1.0).

j :
j is the imaginary unit.

theComplexField :
theComplexField is the instance of **ComplexField**.

4.5.1 ComplexField – field of complex numbers

クラスは複素数上の体である。このクラスは一つのインスタンス **theComplexField** を持つ。

このクラスは **Field** のサブクラスである。

Initialize (Constructor)

ComplexField() → *ComplexField*

複素数体のインスタンスを作る。もしインスタンスを作りたくない場合は、**theComplexField**.

Attributes

zero :

It expresses The additive unit 0. (read only)

one :

It expresses The multiplicative unit 1. (read only)

Operations

operator	explanation
in	membership test; return whether an element is in or not.
repr	return representation string.
str	return string.

Methods

4.5.1.1 createElement – create Imaginary object

`createElement(self, seed: integer) → Integer`

`seed` の複素数オブジェクトを返す。 .

`seed` は複素数か複素数を埋め込んだ数でなければならない。

4.5.1.2 getCharacteristic – get characteristic

`getCharacteristic(self) → integer`

標数が 0 を返す。 .

4.5.1.3 issubring – subring test

`issubring(self, aRing: Ring) → bool`

他の環が複素数体上に部分環として含まれているか教えてくれる。

4.5.1.4 issuperring – superring test

`issuperring(self, aRing: Ring) → bool`

複素数体が他の環を部分環として含んでいるか教えてくれる。

4.5.2 Complex – a complex number

Complex とは複素数のクラスである。どのインスタスも二つの数を持つ。すまわちある数の実部と虚部である。

このクラスは **FieldElement** のサブクラスである。

All implemented operators in this class are delegated to complex type.

Initialize (Constructor)

`Complex(re: number im: number=0) → Imaginary`

複素数を作る。

re は実数でも虚数でも構わない。もし re が実数で im が与えられていないと、虚数は 0 ということである。

Attributes

real :

複素数の実数部分を表す。

imag :

複素数の虚数部分を表す。

Methods

4.5.2.1 getRing – get ring object

`getRing(self)` → *ComplexField*

複素数体のインスタンスを返す。

4.5.2.2 arg – argument of complex

`arg(self)` → *radian*

Return the angle between the x-axis and the number in the Gaussian plane.
radian は Float 型。

4.5.2.3 conjugate – complex conjugate

`conjugate(self)` → *Complex*

ある数の複素共役の値を返す。

4.5.2.4 copy – copied number

`copy(self)` → *Complex*

ある数自身の値を返す。

4.5.2.5 inverse – complex inverse

`inverse(self)` → *Complex*

ある数の逆数の値を返す。

入力されて数が 0 のとき、ZeroDivisionError を返す。

4.5.3 ExponentialPowerSeries – exponential power series

This class is obsolete (Ver 1.1.0).

4.5.4 AbsoluteError – absolute error

This class is obsolete (Ver 1.1.0).

4.5.5 RelativeError – relative error

This class is obsolete (Ver 1.1.0).

4.5.6 exp(function) – exponential value

This function is obsolete (Ver 1.1.0).

4.5.7 expi(function) – imaginary exponential value

This function is obsolete (Ver 1.1.0).

4.5.8 log(function) – logarithm

This function is obsolete (Ver 1.1.0).

4.5.9 sin(function) – sine function

This function is obsolete (Ver 1.1.0).

4.5.10 cos(function) – cosine function

This function is obsolete (Ver 1.1.0).

4.5.11 tan(function) – tangent function

This function is obsolete (Ver 1.1.0).

4.5.12 sinh(function) – hyperbolic sine function

This function is obsolete (Ver 1.1.0).

4.5.13 cosh(function) – hyperbolic cosine function

This function is obsolete (Ver 1.1.0).

4.5.14 tanh(function) – hyperbolic tangent function

This function is obsolete (Ver 1.1.0).

4.5.15 atanh(function) – hyperbolic arc tangent function

This function is obsolete (Ver 1.1.0).

4.5.16 sqrt(function) – square root

This function is obsolete (Ver 1.1.0).

4.6 intresidue – integer residue

intresidue module provides integer residue classes or $\mathbf{Z}/m\mathbf{Z}$.

- **Classes**
 - **IntegerResidueClass**
 - **IntegerResidueClassRing**

4.6.1 IntegerResidueClass – integer residue class

このクラスは **CommutativeRingElement** のサブクラスである。

Initialize (Constructor)

IntegerResidueClass(representative: *integer*, modulus: *integer*)
→ *Integer*

Create a residue class of modulus with residue representative.
modulus は正の整数。

Operations

operator	explanation
a+b	addition.
a-b	subtraction.
a*b	multiplication.
a/b	division.
a**i, pow(a, i)	power.
-a	negation.
+a	make a copy.
a==b	equality or not.
a!=b	inequality or not.
repr(a)	return representation string.
str(a)	return string.

Methods

4.6.1.1 getRing – get ring object

`getRing(self)` → *IntegerResidueClassRing*

環を返す。

4.6.1.2 getResidue – get residue

`getResidue(self)` → *integer*

余りの値を返す。

4.6.1.3 getModulus – get modulus

`getModulus(self)` → *integer*

係数の値を返す。

4.6.1.4 inverse – inverse element

`inverse(self)` → *IntegerResidueClass*

逆元を持つときは逆元の値を返し、さもなければ `ValueError` を返す。

4.6.1.5 minimumAbsolute – minimum absolute representative

`minimumAbsolute(self)` → **Integer**

クラスの代表的な最小な絶対値を返す。

4.6.1.6 minimumNonNegative – smallest non-negative representative

`minimumNonNegative(self)` → **Integer**

`residue` クラスの代表的な最小の整数の要素を返す。 † この方法はエイリアスを持ち、整数より名前がつけられた。

4.6.2 IntegerResidueClassRing – ring of integer residue

このクラスは integer residue classes の環である。

このクラスは **CommutativeRing** のサブクラスである。

Initialize (Constructor)

IntegerResidueClassRing(modulus: integer) → IntegerResidueClassRing

IntegerResidueClassRing のインスタンスを作る。The argument modulus = m specifies an ideal $m\mathbb{Z}$.

Attributes

zero :
加法における 0 を表す。(読み込むときのみ)

one :
乗法における 1 を表す。(読み込むときのみ)

Operations

operator	explanation
R==A	ring equality.
card(R)	return cardinality. See also compatibility module.
e in R	return whether an element is in or not.
repr(R)	return representation string.
str(R)	return string.

Methods

4.6.2.1 createElement – create IntegerResidueClass object

createElement(self, seed: *integer*) → *Integer*

IntegerResidueClass の seed におけるインスタンスを返す。 .

4.6.2.2 isfield – field test

isfield(self) → *bool*

もし係数が素数ならば True をさもなければ False を返す。 Since a finite domain is a field, other ring property tests are merely aliases of isfield; they are isdomain, iseuclidean, isnoetherian, ispid, isufd.

4.6.2.3 getInstance – get instance of IntegerResidueClassRing

getInstance(cls, modulus: *integer*) → *IntegerResidueClass*

ある特定の係数のクラスのインスタンスを返す。これはクラスの方法である。 :

IntegerResidueClassRing.getInstance(3)

to create a $\mathbb{Z}/3\mathbb{Z}$ object, for example.

4.7 lattice – Lattice

- Classes
 - **Lattice**
 - **LatticeElement**
- Functions
 - **LLL**

4.7.1 Lattice – lattice

Initialize (Constructor)

```
Lattice( basis: RingSquareMatrix, quadraticForm: RingSquareMa-  
trix)  
    → Lattice
```

Create Lattice object.

Attributes

basis : The basis of self lattice.

quadraticForm : The quadratic form corresponding the inner product.

Methods

4.7.1.1 createElement – create element

createElement(self, compo: *list*) → **LatticeElement**

Create the element which has coefficients with given compo.

4.7.1.2 bilinearForm – bilinear form

bilinearForm(self, v_1: **Vector**, v_2: **Vector**) → *integer*

Return the inner product of v_1 and v_2 with **quadraticForm**.

4.7.1.3 isCyclic – Check whether cyclic lattice or not

isCyclic(self) → *bool*

Check whether self lattice is a cyclic lattice or not.

4.7.1.4 isIdeal – Check whether ideal lattice or not

isIdeal(self) → *bool*

Check whether self lattice is an ideal lattice or not.

4.7.2 LatticeElement – element of lattice

Initialize (Constructor)

```
LatticeElement( lattice: Lattice, compo: list, )  $\rightarrow$  LatticeElement
```

Create LatticeElement object.

Elements of lattices are represented as linear combinations of basis. The class inherits **Matrix**. Then, instances are regarded as $n \times 1$ matrix whose coefficients consist of compo, where n is the dimension of lattice.

lattice is an instance of Lattice object. compo is coefficients list of basis.

Attributes

lattice : the lattice which includes **self**

Methods

4.7.2.1 `getLattice` – Find lattice belongs to

`getLattice(self)` → **Lattice**

Obtain the Lattice object corresponding to `self`.

4.7.3 LLL(function) – LLL reduction

LLL(M: RingSquareMatrix) → L: RingSquareMatrix, T: RingSquareMatrix

Return LLL-reduced basis for the given basis M.

The output L is the LLL-reduced basis. T is the transportation matrix from the original basis to the LLL-reduced basis.

Examples

```
>>> M=mat.Matrix(3,3,[1,0,12,0,1,26,0,0,13]);  
>>> lat.LLL(M);  
([1, 0, 0]+[0, 1, 0]+[0, 0, 13], [1L, 0L, -12L]+[0L, 1L, -26L]+[0L, 0L, 1L])
```


4.8 matrix – 行列

- **Classes**
 - **Matrix**
 - **SquareMatrix**
 - **RingMatrix**
 - **RingSquareMatrix**
 - **FieldMatrix**
 - **FieldSquareMatrix**
 - **MatrixRing**
 - **Subspace**
- **Functions**
 - **createMatrix**
 - **identityMatrix**
 - **unitMatrix**
 - **zeroMatrix**

matrix モジュールにもいくつかの例外クラスがある.

MatrixSizeError : 入力された行列のサイズが矛盾していると報告.

VectorsNotIndependent : 列ベクトルが一次独立でないと報告.

NoInverseImage : 逆像が存在しないことを報告.

NoInverse : その行列が可逆でないことを報告.

このモジュールは以下のタイプを使うことができる:

compo : compo は以下のどれかの形式でなければならない.

- $[1,2]+[3,4]+[5,6]$ のような連結された行のリスト.
- $[[1,2], [3,4], [5,6]]$ のような行のリストのリスト.
- $[(1, 3, 5), (2, 4, 6)]$ のような列のタプルのリスト.
- $[vector.Vector([1, 3, 5]), vector.Vector([2, 4, 6])]$ のような長さの等しい列ベクトルのリスト.

これらの例はすべて以下の行列を表している:

$$\begin{array}{cc} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{array}$$

4.8.1 Matrix – 行列

Initialize (Constructor)

```
Matrix(row: integer, column: integer, compo: compo=0, coeff_ring:
CommutativeRing=0)
→ Matrix
```

新しい行列オブジェクトを作成.

† この作成されたオブジェクトは自動的に自身のクラスを次に述べるクラスのうちの一つに変える: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

入力すると行列のサイズと係数環を調べ自動的に自身のクラスを決定. row と column は整数, coeff_ring は **Ring** のインスタンスでなければならない. compo についての情報は **compo** を参照. compo を省略すると, 全て 0 のリストであるとみなされる.

予想される入力と出力のリストは以下の通り:

- Matrix(row, column, compo, coeff_ring)
→ 成分は compo, 係数環は coeff_ring である row×column の行列.
- Matrix(row, column, compo)
→ 成分が compo である row×column の行列 (係数環は自動的に決定).
- Matrix(row, column, coeff_ring)
→ 係数環が coeff_ring である row×column の行列 (すべての成分は coeff_ring 上で 0).
- Matrix(row, column)
→ 係数環は **Integer**, すべての成分は 0 である row×column の行列.

Attributes

row : 行列の行数.

column : 行列の列数.

coeff_ring : 行列の係数環.

compo : 行列の成分.

operator	explanation
M==N	M と N が等しいかそうでないか返す.
M[i, j]	行列 M の i 行目 j 列目の成分を返す.
M[i]	行列 M の i 列目の列ベクトルを返す.
M[i, j]=c	行列 M の i 行目 j 列目の成分を c に置き換える.
M[j]=c	行列 M の i 列目の列ベクトルを c に置き換える.
c in M	成分 c が行列 M に入っているかどうか返す.
repr(M)	行列 M の repr 文字列を返す. 文字列は行ベクトルのリストの連結リストを表している.
str(M)	行列 M の string 文字列を返す.

Operations

Examples

```

>>> A = matrix.Matrix(2, 3, [1,0,0]+[0,0,0])
>>> A.__class__.__name__
'RingMatrix'
>>> B = matrix.Matrix(2, 3, [1,0,0,0,0,0])
>>> A == B
True
>>> B[1, 1] = 0
>>> A != B
True
>>> B == 0
True
>>> A[1, 1]
1
>>> print repr(A)
[1, 0, 0]+[0, 0, 0]
>>> print str(A)
1 0 0
0 0 0

```

Methods

4.8.1.1 map – 各成分に関数を適用

`map(self, function: function) → Matrix`

各成分に `function` を適用した行列を返す.

†map 関数は組み込み関数である `map` の類似である.

4.8.1.2 reduce – 繰り返し関数を適用

`reduce(self, function: function, initializer: RingElement=None)
→ RingElement`

左上から右下に `function` を繰り返し適用する. それにより得られる単一の値を返す

†reduce 関数は組み込み関数である `reduce` の類似である.

4.8.1.3 copy – コピー作成

`copy(self) → Matrix`

`self` のコピーを作成する.

† この関数によって作成された行列は `self` と等しい行列だが, インスタンスとしては等しいわけではない.

4.8.1.4 set – 成分を設定

`set(self, compo: compo) → (None)`

`compo` として `compo` のリストを設定.

`compo` は `compo` の形式でなければならない.

4.8.1.5 setRow – m 行目に行ベクトルを設定

```
setRow(self, m: integer, arg: list/Vector) → (None)
```

リストまたは Vector である arg を m 行目として設定.

arg の長さは self.column と等しくなければならない.

4.8.1.6 setColumn – n 列目に列ベクトルを設定

```
setColumn(self, n: integer, arg: list/Vector) → (None)
```

リストまたは Vector である arg を n 列目として設定.

arg の長さは self.row と等しくなければならない.

4.8.1.7 getRow – i 行目の行ベクトルを返す

```
getRow(self, i: integer) → Vector
```

self の形式で i 行目を返す.

この関数は (Vector のインスタンスである) 行ベクトルを返す.

4.8.1.8 getColumn – j 列目の列ベクトルを返す

```
getColumn(self, j: integer) → Vector
```

self の形式で j 列目を返す.

この関数は (Vector のインスタンスである) 列ベクトルを返す.

4.8.1.9 swapRow – 二つの行ベクトルを交換

```
swapRow(self, m1: integer, m2: integer) → (None)
```

self の m1 行目の行ベクトルと m2 行目の行ベクトルを交換.

4.8.1.10 swapColumn – 二つの列ベクトルを交換

```
swapColumn(self, n1: integer, n2: integer) → (None)
```

self の n1 列目の列ベクトルと n2 列目の列ベクトルを交換.

4.8.1.11 insertRow – 行ベクトルを挿入

```
insertRow(self, i: integer, arg: list/Vector/Matrix)  
→ (None)
```

行ベクトル arg を i 行目 row に挿入.

arg はリスト, **Vector** または **Matrix** でなければならない. その長さ (または **column**) は self の列の長さと等しくするべきである.

4.8.1.12 insertColumn – 列ベクトル挿入

```
insertColumn(self, j: integer, arg: list/Vector/Matrix)  
→ (None)
```

列ベクトル arg を j 列目 column に挿入.

arg は, **Vector** または **Matrix** のリストでなければならない. その長さ (または **row**) は self の行の長さと等しくするべきである.

4.8.1.13 extendRow – 行ベクトルを伸張

```
extendRow(self, arg: list/Vector/Matrix) → (None)
```

self に行ベクトル arg を結合 (垂直方向に).

この関数は self の最後の行ベクトルの次に arg を結合. つまり extendRow(arg) は insertRow(self.row+1, arg) と同じ.

arg は, **Vector** または **Matrix** のリストでなければならない. その長さ (または **column**) self の列と等しくするべきである.

4.8.1.14 extendColumn – 列ベクトルを伸張

```
extendColumn(self, arg: list/Vector/Matrix) → (None)
```

self に列ベクトル arg を結合 (水平方向に).

この関数は `self` の最後の列ベクトルの次に `arg` を結合. つまり `extendColumn(arg)` は `(self.column+1, arg)` と同じ.

`arg` は **Vector** または **Matrix** のリストでなければならない. その長さ (または **row**) は `self` の行と等しくすべきである.

4.8.1.15 deleteRow – 行ベクトルを削除

```
deleteRow(self, i: integer) → (None)
```

`i` 行目の行ベクトルを削除.

4.8.1.16 deleteColumn – 列ベクトルを削除

```
deleteColumn(self, j: integer) → (None)
```

`j` 列目の列ベクトルを削除.

4.8.1.17 transpose – 転置行列

```
transpose(self) → Matrix
```

`self` の転置行列を返す.

4.8.1.18 getBlock – ブロック行列

```
getBlock(self, i: integer, j: integer, row: integer, column: integer=None)
→ Matrix
```

`(i, j)` 成分からの `row`×`column` 行列を返す.

もし `column` が省略されたら, `column` は `row` と同じ値とみなす.

4.8.1.19 subMatrix – 部分行列

```
subMatrix(self, I: integer, J: integer=None) → Matrix
subMatrix(self, I: list, J: list=None) → Matrix
```

この関数は二つの意味がある.

- I と J は整数:
I 列目と J 行目を削除した部分行列を返す.
- I と J はリスト:
列 I と行 J で指定された self の成分から構成された部分行列を返す.

もし J を省略すると, J は I と同じ値とみなす.

Examples

```
>>> A = matrix.Matrix(2, 3, [1,2,3]+[4,5,6])
>>> A
[1, 2, 3]+[4, 5, 6]
>>> A.map(complex)
[(1+0j), (2+0j), (3+0j)]+[(4+0j), (5+0j), (6+0j)]
>>> A.reduce(max)
6
>>> A.swapRow(1, 2)
>>> A
[4, 5, 6]+[1, 2, 3]
>>> A.extendColumn([-2, -1])
>>> A
[4, 5, 6, -2]+[1, 2, 3, -1]
>>> B = matrix.Matrix(3, 3, [1,2,3]+[4,5,6]+[7,8,9])
>>> B.subMatrix(2, 3)
[1, 2]+[7, 8]
>>> B.subMatrix([2, 3], [1, 2])
[4, 5]+[7, 8]
```


4.8.2 SquareMatrix – 正方行列

Initialize (Constructor)

```
SquareMatrix(row: integer, column: integer=0, compo: compo=0,  
coeff_ring: CommutativeRing=0)  
→ SquareMatrix
```

新しい正方行列オブジェクトを作成.

SquareMatrix は **Matrix** のサブクラス. † この作成されたオブジェクトは自動的に自身のクラスを次に述べるクラスの内のひとつに変える: **RingMatrix**, **RingSquareMatrix**, **FieldMatrix**, **FieldSquareMatrix**.

入力すると行列のサイズと係数環を調べるにより自動的にそのクラスを決定. row と column は整数, coeff_ring は **Ring** のインスタンスでなければならない. compo に関する情報は **compo** を参照. compo を省略すると, 全て 0 のリストであるとみなされる.

予想される入力と出力のリストは以下の通り:

- Matrix(row, compo, coeff_ring)
→ 成分は compo, 係数環は coeff_ring の row 次正方行列
- Matrix(row, compo)
→ 成分は compo の (係数環は自動的に決定) row 次正方行列
- Matrix(row, coeff_ring)
→ 係数環は coeff_ring の (すべての成分は coeff_ring 上の 0.) row 次正方行列
- Matrix(row)
→ (係数環は整数. すべての成分は 0.) row 次正方行列

† **Matrix** として初期化できるが, その場合 column は row と同じでなければならない.

Methods

4.8.2.1 isUpperTriangularMatrix – 上三角行列かどうか

`isUpperTriangularMatrix(self) → True/False`

`self` が上三角行列かどうか返す.

4.8.2.2 isLowerTriangularMatrix – 下三角行列かどうか

`isLowerTriangularMatrix(self) → True/False`

`self` が下三角行列かどうか返す.

4.8.2.3 isDiagonalMatrix – 対角行列かどうか

`isDiagonalMatrix(self) → True/False`

`self` が対角行列かどうか返す.

4.8.2.4 isScalarMatrix – スカラー行列かどうか

`isScalarMatrix(self) → True/False`

`self` がスカラー行列かどうか返す.

4.8.2.5 isSymmetricMatrix – 対称行列かどうか

`isSymmetricMatrix(self) → True/False`

`self` が対称行列かどうか返す.

Examples

```
>>> A = matrix.SquareMatrix(3, [1,2,3]+[0,5,6]+[0,0,9])
>>> A.isUpperTriangularMatrix()
```

```
True
>>> B = matrix.SquareMatrix(3, [1,0,0]+[0,-2,0]+[0,0,7])
>>> B.isDiagonalMatrix()
True
```

4.8.3 RingMatrix – 成分が環に属する行列

```
RingMatrix(row: integer, column: integer, compo: compo=0,  
            coeff_ring: CommutativeRing=0)  
            → RingMatrix
```

新しい係数が環に属する行列を作成.

RingMatrix は **Matrix** のサブクラス. 初期化に関する情報は Matrix を参照.

Operations

operator	explanation
M+N	M と N の行列の和を返す.
M-N	M と N の行列の差を返す.
M*N	M と N の行列の積を返す. N は行列, ベクトルまたはスカラーでなければならない
M % d	M を d で割った余りを返す. d は 0 でない整数でなければならない.
-M	各成分が M の符号を変えた成分である行列を返す.
+M	M のコピーを返す.

Examples

```
>>> A = matrix.Matrix(2, 3, [1,2,3]+[4,5,6])  
>>> B = matrix.Matrix(2, 3, [7,8,9]+[0,-1,-2])  
>>> A + B  
[8, 10, 12]+[4, 4, 4]  
>>> A - B  
[-6, -6, -6]+[4, 6, 8]  
>>> A * B.transpose()  
[50, -8]+[122, -17]  
>>> -B * vector.Vector([1, -1, 0])  
Vector([1, -1])  
>>> 2 * A  
[2, 4, 6]+[8, 10, 12]  
>>> B % 3  
[1, 2, 0]+[0, 2, 1]
```

Methods

4.8.3.1 getCoefficientRing – 係数環を返す

`getCoefficientRing(self)` → *CommutativeRing*

`self` の係数環を返す.

このメソッドは `self` の全ての成分を調べ, `coeff_ring` を正しい係数環に設定.

4.8.3.2 toFieldMatrix – 係数環として体を設定

`toFieldMatrix(self)` → (*None*)

行列のクラスを係数環が現在の整域の商体になるように **FieldMatrix** または **FieldSquareMatrix** に変える.

4.8.3.3 toSubspace – ベクトル空間としてみなす

`toSubspace(self, isbasis: True/False=None)` → (*None*)

行列のクラスを係数環が現在の整域の商体になるように `Subspace` に変える.

4.8.3.4 hermiteNormalForm (HNF) – Hermite 正規形

`hermiteNormalForm(self)` → *RingMatrix*

`HNF(self)` → *RingMatrix*

上三角行列である Hermite 正規形 (HNF) を返す.

4.8.3.5 exthermiteNormalForm (extHNF) – 拡張 Hermite 正規形アルゴリズム

`exthermiteNormalForm(self)` → (*RingSquareMatrix, RingMatrix*)

`extHNF(self)` → (*RingSquareMatrix, RingMatrix*)

Hermite 正規形 `M` と `self · U = M` を満たす `U` を返す.

この関数は **RingSquareMatrix** のインスタンスである U と **RingMatrix** のインスタンスである M のタプルである (U, M) を返す.

4.8.3.6 kernelAsModule – \mathbb{Z} 加群としての核

`kernelAsModule(self)` \rightarrow *RingMatrix*

\mathbb{Z} 加群としての核を返す.

この関数と **kernel** の違いは値として返されたそれぞれの値は整数であるということ.

Examples

```
>>> A = matrix.Matrix(3, 4, [1,2,3,4,5,6,7,8,9,-1,-2,-3])
>>> print A.hermiteNormalForm()
0 36 29 28
0 0 1 0
0 0 0 1
>>> U, M = A.hermiteNormalForm()
>>> A * U == M
True
>>> B = matrix.Matrix(1, 2, [2, 1])
>>> print B.kernelAsModule()
1
-2
```

4.8.4 RingSquareMatrix – 各成分が環に属する正方行列

```
RingSquareMatrix(row: integer, column: integer=0, compo:
compo=0, coeff_ring: CommutativeRing=0)
→ RingMatrix
```

係数環が環に属する新しい正方行列を作成.

RingSquareMatrix **RingMatrix** と **SquareMatrix** のサブクラス. 初期化に関する情報は SquareMatrix を参照.

Operations

operator	explanation
$M ** c$	行列 M の c 乗を返す.

Examples

```
>>> A = matrix.RingSquareMatrix(3, [1,2,3]+[4,5,6]+[7,8,9])
>>> A ** 2
[30L, 36L, 42L]+[66L, 81L, 96L]+[102L, 126L, 150L]
```

Methods

4.8.4.1 getRing – 行列の環を返す

`getRing(self)` → *MatrixRing*

`self` の所属する **MatrixRing** を返す.

4.8.4.2 isOrthogonalMatrix – 直交行列かどうか

`isOrthogonalMatrix(self)` → *True/False*

`self` が直交行列かどうか返す.

4.8.4.3 isAlternatingMatrix (isAntiSymmetricMatrix, isSkewSymmetricMatrix) – 交代行列かどうか

`isAlternatingMatrix(self)` → *True/False*

`self` が交代行列かどうか返す.

4.8.4.4 isSingular – 特異行列かどうか

`isSingular(self)` → *True/False*

`self` が特異行列かどうか返す.

この関数は `self` が 0 かどうか明らかにする. 正則行列が自動的に逆行列を持つわけではないということに注意; 逆行列が存在するかどうかの性質は係数環に依存する.

4.8.4.5 trace – トレース

`trace(self)` → *RingElement*

`self` のトレースを返す.

4.8.4.6 determinant – 行列式

determinant(self) → *RingElement*

self の行列式を返す.

4.8.4.7 cofactor – 余因子

cofactor(self, i: *integer*, j: *integer*) → *RingElement*

(i, j) の余因子を返す.

4.8.4.8 commutator – 交換子

commutator(self, N: *RingSquareMatrix element*) → *RingSquareMatrix*

self と N の交換子を返す.

$[M, N]$ と表記される M と N の交換子は $[M, N] = MN - NM$ と定義される.

4.8.4.9 characteristicMatrix – 特性行列

characteristicMatrix(self) → *RingSquareMatrix*

self の特性行列を返す.

4.8.4.10 adjugateMatrix – 随伴行列

adjugateMatrix(self) → *RingSquareMatrix*

self の随伴行列を返す.

M に対する随伴行列は単位行列 E に対し $MN = NM = (\det M)E$ となる行列 N.

4.8.4.11 cofactorMatrix (cofactors) – 余因子行列

cofactorMatrix(self) → *RingSquareMatrix*

cofactors(self) → *RingSquareMatrix*

`self` の余因子行列を返す.

M に対する余因子行列は M の (i, j) 成分が (i, j) 余因子である行列. 余因子行列は随伴行列の転置と同じ.

4.8.4.12 `smithNormalForm (SNF, elementary_divisor)` – Smith 正規形 (SNF)

`smithNormalForm(self)` → *RingSquareMatrix*

`SNF(self)` → *RingSquareMatrix*

`elementary_divisor(self)` → *RingSquareMatrix*

`self` に対する Smith 正規形 (SNF) の対角成分のリストを返す.

この関数は `self` が非特異行列であることを仮定している.

4.8.4.13 `extsmithNormalForm (extSNF)` – Smith 正規形 (SNF)

`extsmithNormalForm(self)` → (*RingSquareMatrix, RingSquareMatrix, RingSquareMatrix*)

`extSNF(self)` → *RingSquareMatrix, RingSquareMatrix, RingSquareMatrix*

`self` に対する Smith 正規形である M と $U \cdot self \cdot V = M$ を満たす U, V を返す,.

Examples

```
>>> A = matrix.RingSquareMatrix(3, [3,-5,8]+[-9,2,7]+[6,1,-4])
>>> A.trace()
1L
>>> A.determinant()
-243L
>>> B = matrix.RingSquareMatrix(3, [87,38,80]+[13,6,12]+[65,28,60])
>>> U, V, M = B.extsmithNormalForm()
>>> U * B * V == M
True
>>> print M
4 0 0
0 2 0
0 0 1
>>> B.smithNormalForm()
[4L, 2L, 1L]
```

4.8.5 FieldMatrix – 各成分が体に属する行列

```
FieldMatrix(row: integer, column: integer, compo: compo=0,  
             coeff_ring: CommutativeRing=0)  
             → RingMatrix
```

係数環が体に属する新しい行列を作成.

FieldMatrix は **RingMatrix** のサブクラス. 初期化に関する情報は **Matrix** を参照.

Operations

operator	explanation
M/d	M を d で割った商を返す.d はスカラー.
M//d	M を d で割った商を返す.d はスカラー.

Examples

```
>>> A = matrix.FieldMatrix(3, 3, [1,2,3,4,5,6,7,8,9])  
>>> A / 210  
1/210 1/105 1/70  
2/105 1/42 1/35  
1/30 4/105 3/70
```

Methods

4.8.5.1 kernel – 核

kernel(self) → *FieldMatrix*

self の核を返す.

出力は列ベクトルが核の基底となっている行列.
この関数は核が存在しなければ None を返す.

4.8.5.2 image – 像

image(self) → *FieldMatrix*

self の像を返す.

出力は列ベクトルが像の基底となっている行列.
この関数は核が存在しなければ None を返す.

4.8.5.3 rank – 階数

rank(self) → *integer*

self の階数を返す.

4.8.5.4 inverseImage – 逆像:一次方程式の基底解

inverseImage(self, v: *Vector/RingMatrix*) → *RingMatrix*

self による v の逆像を返す.

この関数は $self \cdot X = v$ と等しい一次式の一つの解を返す.

4.8.5.5 solve – 一次方程式の解

solve(self, B: *Vector/RingMatrix*) → (*RingMatrix*, *RingMatrix*)

$self \cdot X = B$ を解く.

この関数は特殊解 sol と self の核を行列として返す. もし特殊解のみ得たいと

きは **inverseImage** を使用.

4.8.5.6 columnEchelonForm – 列階段行列

columnEchelonForm(self) \rightarrow *RingMatrix*

列被約階段行列を返す.

Examples

```
>>> A = matrix.FieldMatrix(2, 3, [1,2,3]+[4,5,6])
>>> print A.kernel
1/1
-2/1
1
>>> print A.image()
1 2
4 5
>>> C = matrix.FieldMatrix(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[-1,-2,-3])
>>> D = matrix.FieldMatrix(4, 2, [1,0]+[7,6]+[13,12]+[-1,0])
>>> print C.inverseImage(D)
3/1 4/1
-1/1 -2/1
0/1 0/1
>>> sol, ker = C.solve(D)
>>> C * (sol + ker[0]) == D
True
>>> AA = matrix.FieldMatrix(3, 3, [1,2,3]+[4,5,6]+[7,8,9])
>>> print AA.columnEchelonForm()
0/1 2/1 -1/1
0/1 1/1 0/1
0/1 0/1 1/1
```

4.8.6 FieldSquareMatrix – 各成分が体に属する正方行列

```
FieldSquareMatrix(row: integer, column: integer=0, compo:
compo=0, coeff_ring: CommutativeRing=0)
→ FieldSquareMatrix
```

係数環が体に属する新しい正方行列を返す.

FieldSquareMatrix は **FieldMatrix** と **SquareMatrix** のサブクラスです.
†**determinant** 関数はオーバーライドされていて,**determinant** とは異なるアル
ゴリズムを用いている; この関数は **triangulate** から呼ばれる. 初期化に関する
情報は **SquareMatrix** を参照.

Methods

4.8.6.1 triangulate - 行基本変形による三角化

triangulate(self) → *FieldSquareMatrix*

行基本変形によって得られる上三角行列を返す.

4.8.6.2 inverse - 逆行列

inverse(self v: *Vector*/*RingMatrix*=None) → *FieldSquareMatrix*

self の逆行列を返す. v が与えられたら $self^{-1}v$ を返す.

† もし逆行列が存在しなければ **NoInverse** を返す.

4.8.6.3 hessenbergForm - Hessenberg 行列

hessenbergForm(self) → *FieldSquareMatrix*

self の Hessenberg 行列を返す.

4.8.6.4 LUdecomposition - LU 分解

LUdecomposition(self) → (*FieldSquareMatrix*, *FieldSquareMatrix*)

self == LU を満たす下三角行列 L と上三角行列 U を返す.

4.8.7 †MatrixRing – 行列の環

MatrixRing(size: *integer*, scalars: *CommutativeRing*)
→ *MatrixRing*

size と係数環 scalars を与えられた新しい行列の環を作成.

MatrixRing は **Ring** のサブクラス.

Methods

4.8.7.1 unitMatrix - 単位行列

`unitMatrix(self)` \rightarrow *RingSquareMatrix*

単位行列を返す.

4.8.7.2 zeroMatrix - 零行列

`zeroMatrix(self)` \rightarrow *RingSquareMatrix*

零行列を返す.

Examples

```
>>> M = matrix.MatrixRing(3, rational.theIntegerRing)
>>> print M
M_3(Z)
>>> M.unitMatrix()
[1L, 0L, 0L]+[0L, 1L, 0L]+[0L, 0L, 1L]
>>> M.zero
[0L, 0L, 0L]+[0L, 0L, 0L]+[0L, 0L, 0L]
```

4.8.7.3 getInstance(class function) - キャッシュされたインスタンスを返す

```
getInstance(cls, size: integer, scalars: CommutativeRing)  
→ RingSquareMatrix
```

与えられた size とスカラーの環に対する MatrixRing のインスタンスを返す.

初期化の代わりにこのメソッドを使うメリットは, 効率のためメソッドによって作成されたインスタンスがキャッシュされ再利用されることにある.

Examples

```
>>> print MatrixRing.getInstance(3, rational.theIntegerRing)  
M_3(Z)
```

4.8.8 Subspace – 有限次元ベクトル空間の部分空間

```
Subspace(row: integer, column: integer=0, compo: compo=0,  
coeff_ring: CommutativeRing=0, isbasis: True/False=None)  
    → Subspace
```

いくつかの有限次元ベクトル空間の新しい部分空間を作成.

Subspace は **FieldMatrix** のサブクラス.
初期化に関する情報は **Matrix** を参照. 部分空間は self の列ベクトルに張られる空間を示す.

isbasis が True なら, 列ベクトルは一次独立と仮定.

Attributes

isbasis 列ベクトルが一次独立の性質を表す. もし各ベクトルが空間の基底を成せば isbasis は True, そうでなければ False.

Methods

4.8.8.1 issubspace - 部分空間かどうか

Subspace(self, other: *Subspace*) → *True/False*

もし other の部分空間であれば True, そうでなければ False を返す.

4.8.8.2 toBasis - 基底を選択

toBasis(self) → (*None*)

列ベクトルが基底を成すように self を書き直し, その isbasis を True にする.

この関数は isbasis がすでに True ならばなにもしない.

4.8.8.3 supplementBasis - 最大階数にする

supplementBasis(self) → *Subspace*

self に基底を補完したことによる最大階数行列を返す.

4.8.8.4 sumOfSubspaces - 部分空間の和

sumOfSubspaces(self, other: *Subspace*) → *Subspace*

二つの部分空間の和集合の基底を成す列の行列を返す.

4.8.8.5 intersectionOfSubspaces - 部分空間の共通部分

intersectionOfSubspaces(self, other: *Subspace*) → *Subspace*

二つの部分空間の共通部分の基底を成す列の行列を返す.

Examples

```
>>> A = matrix.Subspace(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[10,11,12])
>>> A.toBasis()
>>> print A
1 2
4 5
7 8
10 11
>>> B = matrix.Subspace(3, 2, [1,2]+[3,4]+[5,7])
>>> print B.supplementBasis()
1 2 0
3 4 0
5 7 1
>>> C = matrix.Subspace(4, 1, [1,2,3,4])
>>> D = matrix.Subspace(4, 2, [2,-4]+[4,-3]+[6,-2]+[8,-1])
>>> print C.intersectionOfSubspaces(D)
-2/1
-4/1
-6/1
-8/1
```

4.8.8.6 fromMatrix(class function) - 部分空間を作成

```
fromMatrix(cls, mat: FieldMatrix, isbasis: True/False=None)  
→ Subspace
```

クラスが *Matrix* のサブクラスとなり得る行列のインスタンス *mat* から *Subspace* のインスタンスを作成.

Subspace のインスタンスがほしい場合はこのメソッドを使用.

4.8.9 createMatrix[function] – インスタンスを作成

```
createMatrix(row: integer, column: integer=0, compo: compo=0,  
             coeff_ring: CommutativeRing=None)  
             → RingMatrix
```

RingMatrix, **RingSquareMatrix**, **FieldMatrix** または **FieldSquareMatrix** のインスタンスを作成.

入力すると行列のサイズと係数環を調べることで自動的に自身のクラスを決定. 初期化に関する情報は **Matrix** または **SquareMatrix** を参照.

4.8.10 identityMatrix(unitMatrix)[function] – 単位行列

```
identityMatrix(size: integer, coeff: CommutativeR-  
              ing/CommutativeRingElement=None)  
              → RingMatrix
```

```
unitMatrix(size: integer, coeff: CommutativeR-  
           ing/CommutativeRingElement=None)  
           → RingMatrix
```

size 次元の単位行列を返す.

coeff により, 整数上だけでなく coeff により決定された係数環上でも行列を作成することができる.

coeff は **Ring** のインスタンス, または積に関する単位元でなければならない.

4.8.11 zeroMatrix[function] – 零行列

```
zeroMatrix(row: integer, column: 0=, coeff: CommutativeR-  
          ing/CommutativeRingElement=None)  
          → RingMatrix
```

row × column 零行列を返す.

coeff により, 整数上だけでなく coeff により決定された係数環上でも行列を作成することができる.

coeff は **Ring** のインスタンス, または和に関する単位元でなければならない.
column を省略したら, column は row と同じで設定される.

Examples

```

>>> M = matrix.createMatrix(3, [1,2,3]+[4,5,6]+[7,8,9])
>>> print M
1 2 3
4 5 6
7 8 9
>>> 0 = matrix.zeroMatrix(2, 3, imaginary.ComplexField())
>>> print 0
0 + 0j 0 + 0j 0 + 0j
0 + 0j 0 + 0j 0 + 0j

```


4.9 module – HNF による加群/イデアル

- Classes
 - Submodule
 - Module
 - Ideal
 - Ideal_with_generator

4.9.1 Submodule – 行列表現としての部分加群

Initialize (Constructor)

```
Submodule(row: integer, column: integer, compo: compo=0,  
coeff_ring: CommutativeRing=0, ishnf: True/False=None)  
→ Submodule
```

行列表現で新しい部分加群を作成.

Submodule は **RingMatrix** のサブクラス.
coeff_ring はPID(単項イデアル整域) と仮定. その後行列に対応する HNF(Hermite 正規形) を得る.

ishnf が True なら入力する行列は HNF と仮定.

Attributes

ishnf もし行列が HNF なら ishnf は True, そうでなければ False.

Methods

4.9.1.1 getGenerators – 加群の生成元

`getGenerators(self) → list`

加群 `self` の (現在の) 生成元を返す.

生成元から成るベクトルのリストを返す.

4.9.1.2 isSubmodule – 部分加群かどうか

`isSubmodule(self, other: Submodule) → True/False`

部分加群インスタンスが `other` の部分加群なら `True`, そうでなければ `False` を返す.

4.9.1.3 isEqual – self と other が同じ加群かどうか

`isEqual(self, other: Submodule) → True/False`

部分加群インスタンスが加群として `other` と等しいなら `True`, そうでなければ `False` を返す.

このメソッドは行列でない加群の等式テストにも使用したほうがよい. 行列の等式テストには単純に `self==other` を使用.

4.9.1.4 isContain – other が self に含まれているかどうか

`isContains(self, other: vector.Vector) → True/False`

`other` が `self` に含まれているかどうか返す.

もし `other` を `self` の HNF 生成元の一次結合として表したい場合, `represent_element` を使用.

4.9.1.5 toHNF - HNF に変換

toHNF(self) → (None)

self を HNF (Hermite 正規形) に変換し, ishnf に True を設定.

HNF は常に self の基底を与えるわけではないことに注意. (HNF は冗長なことがある.)

4.9.1.6 sumOfSubmodules - 部分加群の和

sumOfSubmodules(self, other: Submodule) → Submodule

二つの部分空間の和である加群を返す.

4.9.1.7 intersectionOfSubmodules - 部分加群の共通部分

intersectionOfSubmodules(self, other: Submodule) → Submodule

二つの部分空間の共通部分である加群を返す.

4.9.1.8 represent_element - 一次結合として成分を表す

represent_element(self, other: vector.Vector) → vector.Vector/False

other を HNF 生成元の一次結合として表す.

other が self に含まれていなければ, False を返す. このメソッドは **toHNF** を呼ぶことに注意.

このメソッドは **Vector** のインスタンスとしての係数を返す.

4.9.1.9 linear_combination - 一次結合を計算

linear_combination(self, coeff: list) → vector.Vector

Z 係数 coeff が与えられ, (現在) の基底の一次結合に対応するベクトルを返す.

coeff はサイズが self の列と等しい **RingElement** 上のインスタンスのリスト.

Examples

```
>>> A = module.Submodule(4, 3, [1,2,3]+[4,5,6]+[7,8,9]+[10,11,12])
>>> A.toHNF()
>>> print A
9 1
6 1
3 1
0 1
>>> A.getGenerator
[Vector([9L, 6L, 3L, 0L]), Vector([1L, 1L, 1L, 1L])]
>>> V = vector.Vector([10,7,4,1])
>>> A.represent_element(V)
Vector([1L, 1L])
>>> V == A.linear_combination([1,1])
True
>>> B = module.Submodule(4, 1, [1,2,3,4])
>>> C = module.Submodule(4, 2, [2,-4]+[4,-3]+[6,-2]+[8,-1])
>>> print B.intersectionOfSubmodules(C)
2
4
6
8
```

4.9.2 fromMatrix(class function) - 部分加群を作成

```
fromMatrix(cls, mat: RingMatrix, ishnf: True/False=None)  
→ Submodule
```

クラスが `Matrix` のサブクラスになり得る行列のインスタンス `mat` から `Submodule` のインスタンスを作成.

`Submodule` のインスタンスがほしい場合このメソッドを使用.

4.9.3 Module - 数体上の加群

Initialize (Constructor)

```
Module(pair_mat_repr: list/matrix, number_field: algfield.NumberField, base: list/matrix.SquareMatrix=None, ishnf: bool=False)
    → Module
```

数体上の新しい加群オブジェクトを作成.

加群は有限生成された部分 \mathbb{Z} 加群. 加群の階数を $\deg(\text{number_field})$ と仮定しないことを注意.

加群を, θ が number_field . **polynomial** の解となる $\mathbb{Z}[\theta]$ 上の基本的な加群についての生成元として表す.

`pair_mat_repr` は次に示す形式のどれかであるべきである:

- $[M, d]$, M のサイズは number_field の次数である整数のタプルまたはベクトルのリストであり, d は分母.
- $[M, d]$, M のサイズは number_field の次数である整数行列, であり d は分母.
- 行の数は number_field の次数である有理数行列.

また, `base` は次に示す形式のうちのどれかであるべきである:

- サイズは number_field の次数である有理数のタプルまたはベクトルのリスト
- サイズは number_field である非特異かつ有理数係数の正方行列

加群は **base** について内部で $\frac{1}{d}M$ と表され, d は **denominator** で M は **mat_repr**. `ishnf` が `True` なら, `mat_repr` は HNF であると仮定.

Attributes

`mat_repr` : サイズが number_field の次数である **Submodule** M のインスタンス

`denominator` : 整数 d

`base` : サイズは number_field である正方かつ非特異有理数行列

`number_field` : 加群が定義された数体

operator	explanation
$M=N$	M と N が加群として等しいかどうか返す.
$c \in M$	M の要素のどれかが c と等しいかどうか返す.
$M+N$	M と N の加群としての部分集合を返す.
$M*N$	M と N のイデアル計算としての積を返す. N は加群またはスカラーでなければならない (<code>number_field</code> の要素). M と N の共通部分の計算したいときは <code>intersect</code> を参照.
$M**c$	イデアルの乗算を基にした M の c 乗を返す.
<code>repr(M)</code>	加群 M の <code>repr</code> 文字列を返す.
<code>str(M)</code>	加群 M の <code>string</code> 文字列を返す.

Operations

Examples

```
>>> F = algfield.NumberField([2,0,1])
>>> M_1 = module.Module([matrix.RingMatrix(2,2,[1,0]+[0,2]), 2], F)
>>> M_2 = module.Module([matrix.RingMatrix(2,2,[2,0]+[0,5]), 3], F)
>>> print M_1
([1, 0]+[0, 2], 2)
over
([1L, 0L]+[0L, 1L], NumberField([2, 0, 1]))
>>> print M_1 + M_2
([1L, 0L]+[0L, 2L], 6)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 * 2
([1L, 0L]+[0L, 2L], 1L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 * M_2
([2L, 0L]+[0L, 1L], 6L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
>>> print M_1 ** 2
([1L, 0L]+[0L, 2L], 4L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
```


Methods

4.9.3.1 toHNF - Hermite 正規形 (HNF) に変換

`toHNF(self) → (None)`

`self.mat_repr` を Hermite 正規形 (HNF) に変換.

4.9.3.2 copy - コピーを作成

`copy(self) → Module`

`self` のコピーを作成.

4.9.3.3 intersect - 共通部分を返す

`intersect(self, other: Module) → Module`

`self` と `other` の共通部分を返す.

4.9.3.4 issubmodule - 部分加群かどうか

`submodule(self, other: Module) → True/False`

`self` が `other` の部分加群かどうか返す.

4.9.3.5 issupermodule - 部分加群かどうか

`supermodule(self, other: Module) → True/False`

`other` が `self` の部分加群かどうか返す.

4.9.3.6 represent_element - 一次結合として表す

`represent_element(self, other: algfield.BasicAlgNumber)`
`→ list/False`

`other` を `self` で生成される一次結合として表す. もし `other` が `self` に含まれていなかったら, `False` を返す.

`self.mat_repr` は HNF であると仮定しているわけではないことに注意.

`other` が `self` に含まれていたら出力は整数のリスト.

4.9.3.7 `change_base_module` - 基底変換

```
change_base_module(self, other_base: list/matrix.RingSquareMatrix)
    → Module
```

`other_base` に関連した `self` と等しい加群を返す.

`other_base` は `base` の形式に従う.

4.9.3.8 `index` - 加群のサイズ

```
index(self) → rational.Rational
```

`self.base` に関する剰余類の位数を返す. $N \subset M$ なら $[M : N]$ を返し, $M \subset N$ のとき $[N : M]^{-1}$ を返す. M は加群 `self` で N は `self.base` に対応する加群.

4.9.3.9 `smallest_rational` - 有理数体上の \mathbb{Z} 生成元

```
smallest_rational(self) → rational.Rational
```

加群 `self` と有理数体の共通部分の \mathbb{Z} 生成元を返す.

Examples

```
>>> F = algfield.NumberField([1,0,2])
>>> M_1=module.Module([matrix.RingMatrix(2,2,[1,0]+[0,2]), 2], F)
>>> M_2=module.Module([matrix.RingMatrix(2,2,[2,0]+[0,5]), 3], F)
>>> print M_1.intersect(M_2)
([2L, 0L]+[0L, 5L], 1L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
NumberField([2, 0, 1]))
```

```

>>> M_1.represent_element( F.createElement( [[2,4], 1] ) )
[4L, 4L]
>>> print M_1.change_base_module( matrix.FieldSquareMatrix(2, 2, [1,0]+[0,1]) / 2 )
([1L, 0L]+[0L, 2L], 1L)
over
([Rational(1, 2), Rational(0, 1)]+[Rational(0, 1), Rational(1, 2)],
 NumberField([2, 0, 1]))
>>> M_2.index()
Rational(10, 9)
>>> M_2.smallest_rational()
Rational(2, 3)

```

4.9.4 Ideal - 数体上のイデアル

Initialize (Constructor)

```
Ideal(pair_mat_repr: list/matrix, number_field: algfield.NumberField, base: list/matrix.SquareMatrix=None, ishnf: bool=False)
    → Ideal
```

数体上の新しいイデアルオブジェクトを作成.

イデアルは **Module** のサブクラスです.

Module も初期化を引用.

Methods

4.9.4.1 inverse – 逆元

`inverse(self) → Ideal`

`self` の逆イデアルを返す.

このメソッドは `self.number_field.integer_ring` を呼び出す.

4.9.4.2 issubideal – 部分イデアルかどうか

`issubideal(self, other: Ideal) → Ideal`

`self` が `other` の部分イデアルかどうか返す.

4.9.4.3 issuperideal – 部分加群かどうか

`issuperideal(self, other: Ideal) → Ideal`

`other` が `self` の部分加群かどうか返す.

4.9.4.4 gcd – 最大公約数

`gcd(self, other: Ideal) → Ideal`

`self` と `other` のイデアルとしての最大公約数 (gcd) を返す.

このメソッドは単純に `self+other` を実行する.

4.9.4.5 lcm – 最小公倍数

`lcm(self, other: Ideal) → Ideal`

イデアルとしての `self` と `other` の最小公倍数 (lcm) を返す.

このメソッドは単純に `intersect` を呼び出す.

4.9.4.6 norm – ノルム

`norm(self)` → *rational.Rational*

`self` のノルムを返す.

このメソッドは `self.number_field.integer_ring` を呼び出す.

4.9.4.7 isIntegral – 整イデアルかどうか

`isIntegral(self)` → *True/False*

`self` が整イデアルかどうか判定.

Examples

```
>>> M = module.Ideal([matrix.RingMatrix(2, 2, [1,0]+[0,2]), 2], F)
>>> print M.inverse()
([-2L, 0L]+[0L, 2L], 1L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
 NumberField([2, 0, 1]))
>>> print M * M.inverse()
([1L, 0L]+[0L, 1L], 1L)
over
([Rational(1, 1), Rational(0, 1)]+[Rational(0, 1), Rational(1, 1)],
 NumberField([2, 0, 1]))
>>> M.norm()
Rational(1, 2)
>>> M.isIntegral()
False
```

4.9.5 Ideal_with_generator - 生成元によるイデアル

Initialize (Constructor)

`Ideal_with_generator(generator: list) → Ideal_with_generator`

生成元により与えられた新しいイデアルを作成.

generator は同じ数体上の生成元を表す **BasicAlgNumber** のインスタンスのリスト.

Attributes

generator : イデアルの生成元

number_field : 生成元が定義された数体

Operations

operator	explanation
M==N	M と N が加群として等しいかどうか返す.
c in M	M のどれかの要素が c と等しいかどうか返す.
M+N	M と N のイデアル生成元としての和を返す.
M*N	M と N のイデアル生成元としての積を返す.
M**c	イデアルの積を基にした M の c 乗を返す.
repr(M)	イデアル M の repr 文字列を返す.
str(M)	イデアル M の str 文字列を返す.

Examples

```
>>> F = algfield.NumberField([2,0,1])
>>> M_1 = module.Ideal_with_generator([
    F.createElement([[1,0], 2]), F.createElement([[0,1], 1])
])
>>> M_2 = module.Ideal_with_generator([
    F.createElement([[2,0], 3]), F.createElement([[0,5], 3])
])
>>> print M_1
[BasicAlgNumber([[1, 0], 2], [2, 0, 1]), BasicAlgNumber([[0, 1], 1], [2, 0, 1])]
>>> print M_1 + M_2
[BasicAlgNumber([[1, 0], 2], [2, 0, 1]), BasicAlgNumber([[0, 1], 1], [2, 0, 1]),
 BasicAlgNumber([[2, 0], 3], [2, 0, 1]), BasicAlgNumber([[0, 5], 3], [2, 0, 1])]
```

```

>>> print M_1 * M_2
[BasicAlgNumber([[1L, 0L], 3L], [2, 0, 1]), BasicAlgNumber([[0L, 5L], 6], [2, 0, 1]),
BasicAlgNumber([[0L, 2L], 3], [2, 0, 1]), BasicAlgNumber([[-10L, 0L], 3], [2, 0, 1])]
>>> print M_1 ** 2
[BasicAlgNumber([[1L, 0L], 4], [2, 0, 1]), BasicAlgNumber([[0L, 1L], 2], [2, 0, 1]),
BasicAlgNumber([[0L, 1L], 2], [2, 0, 1]), BasicAlgNumber([[-2L, 0L], 1], [2, 0, 1])]

```


Methods

4.9.5.1 copy - コピーを作成

`copy(self) → Ideal_with_generator`

`self` のコピーを作成.

4.9.5.2 to_HNFRepresentation - HNF イdealに変換

`to_HNFRepresentation(self) → Ideal`

`self` をイdealに対応した HNF(Hermite 正規形) 表現に変換.

4.9.5.3 twoElementRepresentation - 二つの要素で表す

`twoElementRepresentation(self) → Ideal_with_generator`

`self` をイdealに対応した HNF(Hermite 正規形) 表現に変換.

`self` が素イdealでなければ, このメソッドは効果がない.

4.9.5.4 smallest_rational - 有理数体上の Z 生成元

`smallest_rational(self) → rational.Rational`

加群 `self` と有理数体の共通部分の Z 生成元を返す.

このメソッドは `to_HNFRepresentation` を呼び出す.

4.9.5.5 inverse - 逆元

`inverse(self) → Ideal`

`self` の逆イdealを返す.

このメソッドは `to_HNFRepresentation` を呼び出す.

4.9.5.6 norm – ノルム

`norm(self) → rational.Rational`

self のノルムを返す.

このメソッドは `to_HNFRepresentation` を呼び出す.

4.9.5.7 intersect - 共通部分

`intersect(self, other: Ideal_with_generator) → Ideal`

self と other の共通部分を返す.

このメソッドは `to_HNFRepresentation` を呼び出す.

4.9.5.8 issubideal – 部分イデアルかどうか

`issubideal(self, other: Ideal_with_generator) → Ideal`

self が other の部分イデアルかどうか返す.

このメソッドは `to_HNFRepresentation` を呼び出す.

4.9.5.9 issuperideal – 部分イデアルかどうか

`issuperideal(self, other: Ideal_with_generator) → Ideal`

このメソッドは `to_HNFRepresentation` を呼び出す.

Examples

```
>>> M = module.Ideal_with_generator([
F.createElement([[2,0], 3]), F.createElement([[0,2], 3]), F.createElement([[1,0], 3])
])
```

```

>>> print M.to_HNFRepresentation()
([2L, 0L, 0L, -4L, 1L, 0L]+[0L, 2L, 2L, 0L, 0L, 1L], 3L)
over
([1L, 0L]+[0L, 1L], NumberField([2, 0, 1]))
>>> print M.twoElementRepresentation()
[BasicAlgNumber([[1L, 0], 3], [2, 0, 1]), BasicAlgNumber([[3, 2], 3], [2, 0, 1])]
>>> M.norm()
Rational(1, 9)

```

4.10 permute – 置換 (対称) 群

- Classes
 - **Permute**
 - **ExPermute**
 - **PermGroup**

4.10.1 Permute – 置換群の元

Initialize (Constructor)

`Permute(value: list/tuple, key: list/tuple) → Permute`

`Permute(val_key: dict) → Permute`

`Permute(value: list/tuple, key: int=None) → Permute`

置換群の元を新しく作成.

インスタンスは“普通の”方法で作成される. すなわち, ある集合の (インデックス付けられた) 全ての元のリストである `key` と, 全ての置換された元のリストである `value` を入力.

普通は, 同じ長さのリスト (またはタプル) である `value` と `key` を入力. または上記の意味での “value” のリストである `values()`, “key” のリストである `keys()` を持つ辞書 `val_key` として入力することができる. また, `key` の入力には簡単な方法がある:

- もし `key` が $[1, 2, \dots, N]$ なら, `key` を入力する必要がない.
- もし `key` が $[0, 1, \dots, N - 1]$ なら, `key` として 0 を入力.
- もし `key` が `value` を昇順として整列したリストと等しければ, 1 を入力.
- もし `key` が `value` を降順として整列したリストと等しければ, -1 を入力.

Attributes

key :
key を表す.

data :
†value のインデックス付きの形式を表す.

Operations

operator	explanation
A==B	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
A*B	右乗算 (すなわち, 通常の写像の演算 $A \circ B$)
A/B	除算 (すなわち, $A \circ B^{-1}$)
A**B	べき乗
A.inverse()	逆元
A[c]	key の c に対応した value の元
A(lst)	A で lst を置換

Examples

```
>>> p1 = permute.Permute(['b','c','d','a','e'], ['a','b','c','d','e'])
>>> print p1
['a', 'b', 'c', 'd', 'e'] -> ['b', 'c', 'd', 'a', 'e']
>>> p2 = permute.Permute([2, 3, 0, 1, 4], 0)
>>> print p2
[0, 1, 2, 3, 4] -> [2, 3, 0, 1, 4]
>>> p3 = permute.Permute(['c','a','b','e','d'], 1)
>>> print p3
['a', 'b', 'c', 'd', 'e'] -> ['c', 'a', 'b', 'e', 'd']
>>> print p1 * p3
['a', 'b', 'c', 'd', 'e'] -> ['d', 'b', 'c', 'e', 'a']
>>> print p3 * p1
['a', 'b', 'c', 'd', 'e'] -> ['a', 'b', 'e', 'c', 'd']
>>> print p1 ** 4
['a', 'b', 'c', 'd', 'e'] -> ['a', 'b', 'c', 'd', 'e']
>>> p1['d']
'a'
>>> p2([0, 1, 2, 3, 4])
[2, 3, 0, 1, 4]
```

Methods

4.10.1.1 setKey – key を変換

`setKey(self, key: list/tuple) → Permute`

他の key を設定.

key は **key** と同じ長さのリストまたはタプルでなければならない.

4.10.1.2 getValue – “value” を得る

`getValue(self) → list`

self の (data でなく) value を返す.

4.10.1.3 getGroup – PermGroup を得る

`getGroup(self) → PermGroup`

self の所属する **PermGroup** を返す.

4.10.1.4 numbering – インデックスを与える

`numbering(self) → int`

置換群の self に数を定める. (遅いメソッド)

次に示す置換群の次元による帰納的な定義に従って定められる.

$(n-1)$ 次元上の $[\sigma_1, \sigma_2, \dots, \sigma_{n-2}, \sigma_{n-1}]$ の番号付けを k とすると, n 次元上の $[\sigma_1, \sigma_2, \dots, \sigma_{n-2}, \sigma_{n-1}, n]$ の番号付けは k , また n 次元上の $[\sigma_1, \sigma_2, \dots, \sigma_{n-2}, n, \sigma_{n-1}]$ の番号付けは $k + (n-1)!$, などとなる. ([Room of Points And Lines, part 2, section 15, paragraph 2 \(Japanese\)](#))

4.10.1.5 order – 元の位数

`order(self) → int/long`

群の元としての位数を返す.

このメソッドは一般の群のそれよりも早い.

4.10.1.6 ToTranspose – 互換の積として表す

`ToTranspose(self) → ExPermute`

`self` を互換の積で表す.

互換 (すなわち二次元巡回) の積とした **ExPermute** の元を返す. これは再帰プログラムであり, **ToCyclic** よりも多くの時間がかかるだろう.

4.10.1.7 ToCyclic – ExPermute の元に対応する

`ToCyclic(self) → ExPermute`

巡回表現の積として `self` を表す.

ExPermute の元を返す. † このメソッドは `self` を互いに素な巡回置換に分解する. よってそれぞれの巡回は可換.

4.10.1.8 sgn – 置換記号

`sgn(self) → int`

置換群の元の置換符号を返す.

もし `self` が偶置換, すなわち, `self` を偶数個の互換の積として書くことができる場合, 1 を返す. さもないと, すなわち奇置換の場合, -1 を返す.

4.10.1.9 types – 巡回置換の形式

`types(self) → list`

それぞれの巡回置換の元の長さによって定義された巡回置換の形式を返す.

4.10.1.10 ToMatrix – 置換行列

`ToMatrix(self) → Matrix`

置換行列を返す.

行と列はkeyに対応する. もし self G が $G[a] = b$ を満たせば, 行列の (a, b) 成分は 1. さもなくば, その元は 0.

Examples

```
>>> p = Permute([2,3,1,5,4])
>>> p.numbering()
28
>>> p.order()
6
>>> p.ToTranspose()
[(4,5)(1,3)(1,2)](5)
>>> p.sgn()
-1
>>> p.ToCyclic()
[(1,2,3)(4,5)](5)
>>> p.types()
'(2,3)type'
>>> print p.ToMatrix()
0 1 0 0 0
0 0 1 0 0
1 0 0 0 0
0 0 0 0 1
0 0 0 1 0
```

4.10.2 ExPermute – 巡回表現としての置換群の元

Initialize (Constructor)

ExPermute(dim: int, value: list, key: list=None) → ExPermute

新しい置換群の元を作成.

インスタンスは“巡回の”方法で作成される. すなわち, 各タプルが巡回表現を表すタプルのリストである value を入力. 例えば, $(\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_k)$ は 1 対 1 写像, $\sigma_1 \mapsto \sigma_2, \sigma_2 \mapsto \sigma_3, \dots, \sigma_k \mapsto \sigma_1$.

dim は自然数でなければならない, すなわち, int, long または **Integer** のインスタンス. key は dim と同じ長さのリストであるべきである. 元が value としての key に入っているタプルのリストを入力. key が $[1, 2, \dots, N]$ という形式なら key を省略することができることに注意. また, key が $[0, 1, \dots, N - 1]$ という形式なら key として 0 を入力することができる.

Attributes

dim :
dim を表す.

key :
key を表す.

data :
† インデックスの付いた value の形式を表す.

Operations

operator	explanation
A==B	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
A*B	右乗算 (すなわち, 普通の写像 $A \circ B$)
A/B	除算 (すなわち, $A \circ B^{-1}$)
A**B	べき乗
A.inverse()	逆元
A[c]	key の c に対応する value の元
A(lst)	lst を A に置換する
str(A)	単純な表記 simplify を用いる.
repr(A)	表記

Examples

```
>>> p1 = permute.ExPermute(5, [('a', 'b')], ['a','b','c','d','e'])
>>> print p1
[('a', 'b')] <['a', 'b', 'c', 'd', 'e']>
>>> p2 = permute.ExPermute(5, [(0, 2), (3, 4, 1)], 0)
>>> print p2
[(0, 2), (1, 3, 4)] <[0, 1, 2, 3, 4]>
>>> p3 = permute.ExPermute(5, [('b','c')], ['a','b','c','d','e'])
>>> print p1 * p3
[('a', 'b'), ('b', 'c')] <['a', 'b', 'c', 'd', 'e']>
>>> print p3 * p1
[('b', 'c'), ('a', 'b')] <['a', 'b', 'c', 'd', 'e']>
>>> p1['c']
'c'
>>> p2([0, 1, 2, 3, 4])
[2, 4, 0, 1, 3]
```

Methods

4.10.2.1 setKey – key を変換

`setKey(self, key: list) → ExPermute`

他の key を設定.

key は **dim** と同じ長さのリストでなければならない.

4.10.2.2 getValue – “value” を得る

`getValue(self) → list`

self の (data でなく) value を返す.

4.10.2.3 getGroup – PermGroup を得る

`getGroup(self) → PermGroup`

self が所属する **PermGroup** を返す.

4.10.2.4 order – 元の位数

`order(self) → int/long`

群の元としての位数を返す.

このメソッドは一般の群のそれよりも早い.

4.10.2.5 ToNormal – 普通の表現

`ToNormal(self) → Permute`

self を **Permute** のインスタンスとして表す.

4.10.2.6 simplify – 単純な値を使用

`simplify(self) → ExPermute`

より単純な巡回表現を返す.

† このメソッドは **ToNormal** と **ToCyclic** を使用.

4.10.2.7 sgn – 置換符号

`sgn(self) → int`

置換群の元の置換符号を返す.

もし `self` が偶置換なら, すなわち, `self` が偶数個の互換の積として書くことができる場合, 1 を返す. さもなくば, すなわち奇置換なら, -1 を返す.

Examples

```
>>> p = permute.ExPermute(5, [(1, 2, 3), (4, 5)])
>>> p.order()
6
>>> print p.ToNormal()
[1, 2, 3, 4, 5] -> [2, 3, 1, 5, 4]
>>> p * p
[(1, 2, 3), (4, 5), (1, 2, 3), (4, 5)] <[1, 2, 3, 4, 5]>
>>> (p * p).simplify()
[(1, 3, 2)] <[1, 2, 3, 4, 5]>
```

4.10.3 PermGroup – 置換群

Initialize (Constructor)

`PermGroup(key: int/long) → PermGroup`

`PermGroup(key: list/tuple) → PermGroup`

新しい置換群を作成.

普通は, `key` としてリストを入力. もしある整数 N を入力したら, `key` は $[1, 2, \dots, N]$ として設定される.

Attributes

`key` :
 `key` を表す.

Operations

operator	explanation
<code>A==B</code>	A の value と B の value, そして A の key と B の key が等しいかどうか返す.
<code>card(A)</code>	grouporder と同じ
<code>str(A)</code>	単純な表記
<code>repr(A)</code>	表記

Examples

```
>>> p1 = permute.PermGroup(['a','b','c','d','e'])
>>> print p1
['a','b','c','d','e']
>>> card(p1)
120L
```

Methods

4.10.3.1 createElement – シードから元を作成

`createElement(self, seed: list/tuple/dict) → Permute`

`createElement(self, seed: list) → ExPermute`

self の新しい元を作成.

seed は **Permute** または **ExPermute** の “value” の形式でなければならない

4.10.3.2 identity – 単位元

`identity(self) → Permute`

普通の表現で self の単位元を返す.

巡回表現の場合, **identity_c** を使用.

4.10.3.3 identity_c – 巡回表現の単位元

`identity_c(self) → ExPermute`

巡回表現として置換群の単位元を返す.

普通の表現の場合, **identity** を使用.

4.10.3.4 grouporder – 群の位数

`grouporder(self) → int/long`

群としての self の位数を計算.

4.10.3.5 randElement – 無作為に元を選ぶ

`randElement(self) → Permute`

普通の表現として無作為に新しい self の元を作成.

Examples

```
>>> p = permute.PermGroup(5)
>>> print p.createElement([3, 4, 5, 1, 2])
[1, 2, 3, 4, 5] -> [3, 4, 5, 1, 2]
>>> print p.createElement([(1, 2), (3, 4)])
[(1, 2), (3, 4)] <[1, 2, 3, 4, 5]>
>>> print p.identity()
[1, 2, 3, 4, 5] -> [1, 2, 3, 4, 5]
>>> print p.identity_c()
[] <[1, 2, 3, 4, 5]>
>>> p.grouporder()
120L
>>> print p.randElement()
[1, 2, 3, 4, 5] -> [3, 4, 5, 2, 1]
```


4.11 rational – 整数と有理数

rational モジュールはクラス Rational, クラス Integer, クラス RationalField, そして クラス IntegerRing として整数と有理数を提供.

- Classes
 - **Integer**
 - **IntegerRing**
 - **Rational**
 - **RationalField**

このモジュールはまた以下のコンテンツを提供する:

theIntegerRing :

theIntegerRing は有理整数環を表す. **IntegerRing** のインスタンス.

theRationalField :

theRationalField は有理数体を表す. **RationalField** のインスタンス.

4.11.1 Integer – 整数

Integer は整数のクラス. 'int' と 'long' は除算において有理数を返さないで, 新しいクラスを作成する必要があった.

このクラスは **CommutativeRingElement** と long のサブクラス.

Initialize (Constructor)

Integer(integer: integer) → Integer

Integer オブジェクトを構成. もし引数が省略されたら, 値は 0 となる.

Methods

4.11.1.1 `getRing` – ring オブジェクトを得る

`getRing(self)` → *IntegerRing*

IntegerRing オブジェクトを返す.

4.11.1.2 `actAdditive` – 2 進の加法鎖の加法

`actAdditive(self, other: integer)` → *Integer*

`other` に加法的に作用, すなわち, `n` は `other` の `n` 回の加算に拡大される. 結果としては以下と同じ:

```
return sum([+other for _ in range(self)])
```

しかし, ここでは 2 進の加法鎖を使う.

4.11.1.3 `actMultiplicative` – 2 進の加法鎖の乗法

`actMultiplicative(self, other: integer)` → *Integer*

`other` に乗法的に作用する, すなわち, `n` は `other` の `n` 回の乗算に拡大される. 結果としては以下と同じ:

```
return reduce(lambda x,y: x*y, [+other for _ in range(self)])
```

しかし, ここでは 2 進の加法鎖を使う.

4.11.2 IntegerRing – 整数環

有理整数環に対するクラス.

このクラスは **CommutativeRing** のサブクラス.

Initialize (Constructor)

`IntegerRing()` \rightarrow *IntegerRing*

`IntegerRing` のインスタンスを作成. すでに `theIntegerRing` があるので, インスタンスを作成する必要があるかもしれない.

Attributes

zero :
加法の単位元 0 を表す. (読み込み専用)

one :
乗法の単位元 1 を表す. (読みこみ専用)

Operations

operator	explanation
<code>x in Z</code>	元が含まれているどうか返す.
<code>repr(Z)</code>	<code>repr</code> 文字列を返す.
<code>str(Z)</code>	<code>str</code> 文字列を返す.

Methods

4.11.2.1 createElement – Integer オブジェクトを作成

`createElement(self, seed: integer) → Integer`

seed に対する Integer オブジェクトを作成.

seed は int 型, long 型 または rational.Integer でなければならない.

4.11.2.2 gcd – 最大公約数

`gcd(self, n: integer, m: integer) → Integer`

与えられた二つの整数の最大公約数を返す.

4.11.2.3 extgcd – 拡張 GCD

`extgcd(self, n: integer, m: integer) → Integer`

タプル (u, v, d) を返す; これらは与えられた二つの整数 n と m の最大公約数 d と, $d = nu + mv$ となる u, v .

4.11.2.4 lcm – 最小公倍数

`lcm(self, n: integer, m: integer) → Integer`

与えられた二つの整数の最小公倍数を返す. もし両方とも 0 なら, エラーが起こる.

4.11.2.5 getQuotientField – 有理数体オブジェクトを得る

`getQuotientField(self) → RationalField`

有理数体 (**RationalField**) を返す.

4.11.2.6 issubring – 部分環かどうか判定

`issubring(self, other: Ring) → bool`

もう一方の環が部分環として整数環を含んでいるか報告.

もし `other` も整数環なら, 出力は `True`. その他の場合もう一方の整数環の `issuperring` メソッドにおける実装に依存.

4.11.2.7 `issuperring` – 含んでいるかどうか判定

`issuperring(self, other: Ring) → bool`

整数環がもう一方の環を部分環として含んでいるか報告.

もし `other` も整数環なら, 出力は `True`. その他の場合もう一方の整数環の `issubring` メソッドにおける実装に依存.

4.11.3 Rational – 有理数

有理数のクラス.

Initialize (Constructor)

```
Rational(numerator: numbers, denominator: numbers=1)  
→ Integer
```

有理数は以下から構成:

- 整数,
- float
- Rational.

もし toRational メソッドがあれば, 他のオブジェクトを変換することができる. さもなくば TypeError が起こる.

Methods

4.11.3.1 getRing – ring オブジェクトを得る

`getRing(self) → RationalField`

RationalField オブジェクトを返す.

4.11.3.2 decimalString – 小数を表す

`decimalString(self, N: integer) → string`

小数第 N 桁とした文字列を返す.

4.11.3.3 expand – 連分数による表現

`expand(self, base: integer, limit: integer) → string`

もし `base` が自然数なら, 分母が `base` の高々 `limit` 乗である最も近い有理数を返す.

さもなければ (すなわち, `base=0`), 分母が高々 `limit` である最も近い有理数を返す.

`base` は負の整数であってはならない.

4.11.4 RationalField – 有理数体

RationalField は有理数体のクラス. このクラスは **theRationalField** という唯一のインスタンスを持つ.

このクラスは **QuotientField** のサブクラス.

Initialize (Constructor)

RationalField() \rightarrow *RationalField*

RationalField のインスタンスを作成. すでに theRationalField があるので, インスタンスを作成する必要はないかもしれない.

Attributes

zero :

加法の単位元 0 を表す, すなわち Rational(0, 1). (読み込み専用)

one :

乗法の単位元 1 を表す, すなわち Rational(1, 1). (読み込み専用)

Operations

operator	explanation
<code>x in Q</code>	元が含まれているかどうか返す.
<code>str(Q)</code>	str 文字列を返す.

Methods

4.11.4.1 createElement – Rational オブジェクトを返す

```
createElement(self, numerator: integer or Rational, denominator: integer=1 )  
    → Rational
```

Rational オブジェクトを作成.

4.11.4.2 classNumber – 類数を得る

```
classNumber(self) → integer
```

有理数体の類数は 1 なので, 1 を返す.

4.11.4.3 getQuotientField – 有理数体オブジェクトを返す

```
getQuotientField(self) → RationalField
```

有理数体インスタンスを返す.

4.11.4.4 issubring – 部分環かどうか判定

```
issubring(self, other: Ring) → bool
```

もう一方の環が部分環として有理数体を含んでいるか報告.

もし other もまた有理数体なら, 出力は True. 他の場合もう一方の issuperring メソッドにおける実装に依存.

4.11.4.5 issuperring – 含んでいるかどうか判定

```
issuperring(self, other: Ring) → bool
```

有理数体がもう一方の環を部分環としてを含んでいるか報告.

もし other もまた有理数体なら, 出力は True. 他の場合もう一方の issubring メソッドにおける実装に依存.

4.12 real – real numbers and its functions

The module `real` provides arbitrary precision real numbers and their utilities. The functions provided are corresponding to the `math` standard module.

- **Classes**

- `RealField`
- `Real`
- `†Constant`
- `†ExponentialPowerSeries`
- `†AbsoluteError`
- `†RelativeError`

- **Functions**

- `exp`
- `sqrt`
- `log`
- `log1piter`
- `piGaussLegendre`
- `eContinuedFraction`
- `floor`
- `ceil`
- `trunc`
- `sin`
- `cos`
- `tan`
- `sinh`
- `cosh`
- `tanh`
- `asin`
- `acos`
- `atan`
- `atan2`
- `hypot`
- `pow`
- `degrees`
- `radians`

- **fabs**
- **fmod**
- **frexp**
- **ldexp**
- **EulerTransform**

This module also provides following constants:

- e** :
This constant is obsolete (Ver 1.1.0).
- pi** :
This constant is obsolete (Ver 1.1.0).
- Log2** :
This constant is obsolete (Ver 1.1.0).
- theRealField** :
theRealField is the instance of **RealField**.

4.12.1 RealField – field of real numbers

The class is for the field of real numbers. The class has the single instance **theRealField**.

This class is a subclass of **Field**.

Initialize (Constructor)

RealField() \rightarrow *RealField*

Create an instance of RealField. You may not want to create an instance, since there is already **theRealField**.

Attributes

zero :

It expresses the additive unit 0. (read only)

one :

It expresses the multiplicative unit 1. (read only)

Operations

operator	explanation
x in R	membership test; return whether an element is in or not.
repr(R)	return representation string.
str(R)	return string.

Methods

4.12.1.1 `getCharacteristic` – get characteristic

`getCharacteristic(self)` → *integer*

Return the characteristic, zero.

4.12.1.2 `issubring` – subring test

`issubring(self, aRing: Ring)` → *bool*

Report whether another ring contains the real field as subring.

4.12.1.3 `issuperring` – superring test

`issuperring(self, aRing: Ring)` → *bool*

Report whether the real field contains another ring as subring.

4.12.2 Real – a Real number

Real is a class of real number. This class is only for consistency for other **Ring** object.

This class is a subclass of **CommutativeRingElement**.

All implemented operators in this class are delegated to Float type.

Initialize (Constructor)

Real(value: *number*) \rightarrow *Real*

Construct a Real object.

value must be int, long, Float or **Rational**.

Methods

4.12.2.1 `getRing` – get ring object

`getRing(self)` \rightarrow *RealField*

Return the real field instance.

4.12.3 Constant – real number with error correction

This class is obsolete (Ver 1.1.0).

4.12.4 ExponentialPowerSeries – exponential power series

This class is obsolete (Ver 1.1.0).

4.12.5 AbsoluteError – absolute error

This class is obsolete (Ver 1.1.0).

4.12.6 RelativeError – relative error

This class is obsolete (Ver 1.1.0).

4.12.7 exp(function) – exponential value

This function is obsolete (Ver 1.1.0).

4.12.8 sqrt(function) – square root

This function is obsolete (Ver 1.1.0).

4.12.9 log(function) – logarithm

This function is obsolete (Ver 1.1.0).

4.12.10 log1piter(function) – iterator of $\log(1+x)$

log1piter(xx: *number*) → *iterator*

Return iterator for $\log(1+x)$.

4.12.11 piGaussLegendre(function) – pi by Gauss-Legendre

This function is obsolete (Ver 1.1.0).

4.12.12 eContinuedFraction(function) – Napier's Constant by continued fraction expansion

This function is obsolete (Ver 1.1.0).

4.12.13 **floor(function)** – floor the number

floor(*x*: *number*) → *integer*

Return the biggest integer not more than *x*.

4.12.14 **ceil(function)** – ceil the number

ceil(*x*: *number*) → *integer*

Return the smallest integer not less than *x*.

4.12.15 **tranc(function)** – round-off the number

tranc(*x*: *number*) → *integer*

Return the number of rounded off *x*.

4.12.16 **sin(function)** – sine function

This function is obsolete (Ver 1.1.0).

4.12.17 **cos(function)** – cosine function

This function is obsolete (Ver 1.1.0).

4.12.18 **tan(function)** – tangent function

This function is obsolete (Ver 1.1.0).

4.12.19 **sinh(function)** – hyperbolic sine function

This function is obsolete (Ver 1.1.0).

4.12.20 **cosh(function)** – hyperbolic cosine function

This function is obsolete (Ver 1.1.0).

4.12.21 **tanh(function)** – hyperbolic tangent function

This function is obsolete (Ver 1.1.0).

4.12.22 `asin(function)` – arc sine function

This function is obsolete (Ver 1.1.0).

4.12.23 `acos(function)` – arc cosine function

This function is obsolete (Ver 1.1.0).

4.12.24 `atan(function)` – arc tangent function

This function is obsolete (Ver 1.1.0).

4.12.25 `atan2(function)` – arc tangent function

This function is obsolete (Ver 1.1.0).

4.12.26 `hypot(function)` – Euclidean distance function

This function is obsolete (Ver 1.1.0).

4.12.27 `pow(function)` – power function

This function is obsolete (Ver 1.1.0).

4.12.28 `degrees(function)` – convert angle to degree

This function is obsolete (Ver 1.1.0).

4.12.29 `radians(function)` – convert angle to radian

This function is obsolete (Ver 1.1.0).

4.12.30 `fabs(function)` – absolute value

`fabs(x: number) → number`

Return absolute value of x

4.12.31 `fmod(function)` – modulo function over real

`fmod(x: number, y: number) → number`

Return $x - ny$, where n is the quotient of x / y , rounded towards zero to an integer.

4.12.32 **frexp(function) – expression with base and binary exponent**

frexp(x: *number*) → (m,e)

Return a tuple (m,e), where $x = m \times 2^e$, $1/2 \leq \text{abs}(m) < 1$ and e is an integer.

†This function is provided as the counter-part of math.frexp, but it might not be useful.

4.12.33 **ldexp(function) – construct number from base and binary exponent**

ldexp(x: *number*, i: *number*) → *number*

Return $x \times 2^i$.

4.12.34 **EulerTransform(function) – iterator yields terms of Euler transform**

EulerTransform(iterator: *iterator*) → *iterator*

Return an iterator which yields terms of Euler transform of the given iterator.

†

4.13 ring – for ring object

- **Classes**
 - **Ring**
 - **CommutativeRing**
 - **Field**
 - **QuotientField**
 - **RingElement**
 - **CommutativeRingElement**
 - **FieldElement**
 - **QuotientFieldElement**
 - **Ideal**
 - **ResidueClassRing**
 - **ResidueClass**
 - **CommutativeRingProperties**
- **Functions**
 - **getRingInstance**
 - **getRing**
 - **inverse**
 - **exact_division**

4.13.1 †Ring – abstract ring

Ring is an abstract class which expresses that the derived classes are (in mathematical meaning) rings.

Definition of ring (in mathematical meaning) is as follows: Ring is a structure with addition and multiplication. It is an abelian group with addition, and a monoid with multiplication. The multiplication obeys the distributive law.

This class is abstract and cannot be instantiated.

Attributes

zero additive unit

one multiplicative unit

Operations

operator	explanation
A==B	Return whether M and N are equal or not.

Methods

4.13.1.1 createElement – create an element

createElement(self, seed: (*undefined*)) → *RingElement*

Return an element of the ring with seed.

This is an abstract method.

4.13.1.2 getCharacteristic – characteristic as ring

getCharacteristic(self) → *integer*

Return the characteristic of the ring.

The Characteristic of a ring is the smallest positive integer n s.t. $na = 0$ for any element a of the ring, or 0 if there is no such natural number.
This is an abstract method.

4.13.1.3 issubring – check subring

issubring(self, other: *RingElement*) → *True/False*

Report whether another ring contains the ring as a subring.

This is an abstract method.

4.13.1.4 issuperring – check superring

issuperring(self, other: *RingElement*) → *True/False*

Report whether the ring is a superring of another ring.

This is an abstract method.

4.13.1.5 getCommonSuperring – get common ring

`getCommonSuperring(self, other: RingElement) → RingElement`

Return common super ring of self and another ring.

This method uses **issubring**, **issuperring**.

4.13.2 †CommutativeRing – abstract commutative ring

CommutativeRing is an abstract subclass of **Ring** whose multiplication is commutative.

CommutativeRing is subclass of **Ring**.

There are some properties of commutative rings, algorithms should be chosen accordingly. To express such properties, there is a class **CommutativeRingProperties**.

This class is abstract and cannot be instantiated.

Attributes

properties an instance of **CommutativeRingProperties**

Methods

4.13.2.1 getQuotientField – create quotient field

getQuotientField(self) → *QuotientField*

Return the quotient field of the ring.

This is an abstract method.
If quotient field of **self** is not available, it should raise exception.

4.13.2.2 isdomain – check domain

isdomain(self) → *True/False/None*

Return True if the ring is actually a domain, False if not, or None if uncertain.

4.13.2.3 isnoetherian – check Noetherian domain

isnoetherian(self) → *True/False/None*

Return True if the ring is actually a Noetherian domain, False if not, or None if uncertain.

4.13.2.4 isufd – check UFD

isufd(self) → *True/False/None*

Return True if the ring is actually a unique factorization domain (UFD), False if not, or None if uncertain.

4.13.2.5 ispid – check PID

ispid(self) → *True/False/None*

Return True if the ring is actually a principal ideal domain (PID), False if not, or None if uncertain.

4.13.2.6 iseuclidean – check Euclidean domain

iseuclidean(self) → *True/False/None*

Return True if the ring is actually a Euclidean domain, False if not, or None if uncertain.

4.13.2.7 isfield – check field

isfield(self) → *True/False/None*

Return True if the ring is actually a field, False if not, or None if uncertain.

4.13.2.8 registerModuleAction – register action as ring

registerModuleAction(self, action_ring: *RingElement*, action: *function*)
→ (*None*)

Register a ring action_ring, which act on the ring through action so the ring be an action_ring module.

See **hasaction**, **getaction**.

4.13.2.9 hasaction – check if the action has

hasaction(self, action_ring: *RingElement*) → *True/False*

Return True if action_ring is registered to provide action.

See **registerModuleAction**, **getaction**.

4.13.2.10 getaction – get the registered action

hasaction(self, action_ring: *RingElement*) → *function*

Return the registered action for action_ring.

See **registerModuleAction**, **hasaction**.

4.13.3 †Field – abstract field

Field is an abstract class which expresses that the derived classes are (in mathematical meaning) fields, i.e., a commutative ring whose multiplicative monoid is a group.

Field is subclass of **CommutativeRing**. **getQuotientField** and **isfield** are not abstract (trivial methods).

This class is abstract and cannot be instantiated.

Methods

4.13.3.1 gcd – gcd

`gcd(self, a: FieldElement, b: FieldElement) → FieldElement`

Return the greatest common divisor of `a` and `b`.

A field is trivially a UFD and should provide `gcd`. If we can implement an algorithm for computing `gcd` in an Euclidean domain, we should provide the method corresponding to the algorithm.

4.13.4 †QuotientField – abstract quotient field

QuotientField is an abstract class which expresses that the derived classes are (in mathematical meaning) quotient fields.

`self` is the quotient field of domain.

QuotientField is subclass of **Field**.

In the initialize step, it registers trivial action named as `baseaction`; i.e. it expresses that an element of a domain acts an element of the quotient field by using the multiplication in the domain.

This class is abstract and cannot be instantiated.

Attributes

basedomain domain which generates the quotient field `self`

4.13.5 †**RingElement** – abstract element of ring

RingElement is an abstract class for elements of rings.

This class is abstract and cannot be instantiated.

Operations

operator	explanation
A==B	equality (abstract)

Methods

4.13.5.1 `getRing` – `getRing`

`getRing(self) → Ring`

Return an object of a subclass of `Ring`, to which the element belongs.

This is an abstract method.

4.13.6 `‡CommutativeRingElement` – abstract element of commutative ring

`CommutativeRingElement` is an abstract class for elements of commutative rings.

This class is abstract and cannot be instantiated.

Methods

4.13.6.1 `mul_module_action` – apply a module action

`mul_module_action(self, other: RingElement) → (undefined)`

Return the result of a module action. `other` must be in one of the action rings of `self`'s ring.

This method uses `getRing`, `CommutativeRing` and `getaction`. We should consider that the method is abstract.

4.13.6.2 `exact_division` – division exactly

`exact_division(self, other: CommutativeRingElement)
→ CommutativeRingElement`

In UFD, if `other` divides `self`, return the quotient as a UFD element.

The main difference with `/` is that `/` may return the quotient as an element of quotient field.

Simple cases:

- in a Euclidean domain, if remainder of euclidean division is zero, the division `//` is exact.
- in a field, there's no difference with `/`.

If `other` doesn't divide `self`, raise `ValueError`. Though `__divmod__` can be used automatically, we should consider that the method is abstract.

4.13.7 †FieldElement – abstract element of field

FieldElement is an abstract class for elements of fields.

FieldElement is subclass of **CommutativeRingElement**. **exact_division** are not abstract (trivial methods).

This class is abstract and cannot be instantiated.

4.13.8 †QuotientFieldElement – abstract element of quotient field

QuotientFieldElement class is an abstract class to be used as a super class of concrete quotient field element classes.

QuotientFieldElement is subclass of **FieldElement**.
self expresses $\frac{\text{numerator}}{\text{denominator}}$ in the quotient field.

This class is abstract and should not be instantiated.
denominator should not be 0.

Attributes

numerator numerator of self

denominator denominator of self

Operations

operator	explanation
A+B	addition
A-B	subtraction
A*B	multiplication
A**B	powering
A/B	division
-A	sign reversion (additive inversion)
inverse(A)	multiplicative inversion
A==B	equality

4.13.9 †Ideal – abstract ideal

Ideal class is an abstract class to represent the finitely generated ideals.

†Because the finitely-generatedness is not a restriction for Noetherian rings and in the most cases only Noetherian rings are used, it is general enough.

This class is abstract and should not be instantiated.
generators must be an element of the `aring` or a list of elements of the `aring`.
If generators is an element of the `aring`, we consider `self` is the principal ideal generated by generators.

Attributes

ring the ring belonged to by `self`

generators generators of the ideal `self`

Operations

operator	explanation
<code>I+J</code>	addition $\{i + j \mid i \in I, j \in J\}$
<code>I*J</code>	multiplication $IJ = \{\sum_{i,j} ij \mid i \in I, j \in J\}$
<code>I==J</code>	equality
<code>e in I</code>	For <code>e</code> in the ring, to which the ideal <code>I</code> belongs.

Methods

4.13.9.1 `issubset` – check subset

`issubset(self, other: Ideal) → True/False`

Report whether another ideal contains this ideal.

We should consider that the method is abstract.

4.13.9.2 `issuperset` – check superset

`issuperset(self, other: Ideal) → True/False`

Report whether this ideal contains another ideal.

We should consider that the method is abstract.

4.13.9.3 `reduce` – reduction with the ideal

`issuperset(self, other: Ideal) → True/False`

Reduce an element with the ideal to simpler representative.

This method is abstract.

4.13.10 †ResidueClassRing – abstract residue class ring

Initialize (Constructor)

```
ResidueClassRing(ring: CommutativeRing, ideal: Ideal)  
→ ResidueClassRing
```

A residue class ring R/I , where R is a commutative ring and I is its ideal.

ResidueClassRing is subclass of **CommutativeRing**.
one, **zero** are not abstract (trivial methods).

Because we assume that `ring` is Noetherian, so is `ring`.

This class is abstract and should not be instantiated.
`ring` should be an instance of **CommutativeRing**, and `ideal` must be an instance of **Ideal**, whose `ring` attribute points the same ring with the given `ring`.

Attributes

ring the base ring R

ideal the ideal I

Operations

operator	explanation
<code>A==B</code>	equality
<code>e in A</code>	report whether <code>e</code> is in the residue ring <code>self</code> .

4.13.11 †ResidueClass – abstract an element of residue class ring

Initialize (Constructor)

```
ResidueClass(x: CommutativeRingElement, ideal: Ideal)  
    → ResidueClass
```

Element of residue class ring $x + I$, where I is the modulus ideal and x is a representative element.

ResidueClass is subclass of **CommutativeRingElement**.

This class is abstract and should not be instantiated.
ideal corresponds to the ideal I .

Operations

These operations uses **reduce**.

operator	explanation
x+y	addition
x-y	subtraction
x*y	multiplication
A==B	equality

4.13.12 †CommutativeRingProperties – properties for CommutativeRingProperties

Initialize (Constructor)

CommutativeRingProperties((None)) → CommutativeRingProperties

Boolean properties of ring.

Each property can have one of three values; *True*, *False*, or *None*. Of course *True* is true and *False* is false, and *None* means that the property is not set neither directly nor indirectly.

CommutativeRingProperties class treats

- Euclidean (Euclidean domain),
- PID (Principal Ideal Domain),
- UFD (Unique Factorization Domain),
- Noetherian (Noetherian ring (domain)),
- field (Field)

Methods

4.13.12.1 isfield – check field

isfield(self) → *True/False/None*

Return True/False according to the field flag value being set, otherwise return None.

4.13.12.2 setIsfield – set field

isfield(self, value: *True/False*) → (*None*)

Set True/False value to the field flag.
Propagation:

- True → euclidean

4.13.12.3 iseclidean – check euclidean

iseclidean(self) → *True/False/None*

Return True/False according to the euclidean flag value being set, otherwise return None.

4.13.12.4 setIseclidean – set euclidean

isfield(self, value: *True/False*) → (*None*)

Set True/False value to the euclidean flag.
Propagation:

- True → PID
- False → field

4.13.12.5 ispid – check PID

ispid(self) → *True/False/None*

Return True/False according to the PID flag value being set, otherwise return None.

4.13.12.6 setIspid – set PID

ispid(self, value: *True/False*) → (*None*)

Set True/False value to the euclidean flag.
Propagation:

- True → UFD, Noetherian
- False → euclidean

4.13.12.7 isufd – check UFD

isufd(self) → *True/False/None*

Return True/False according to the UFD flag value being set, otherwise return None.

4.13.12.8 setIsufd – set UFD

isufd(self, value: *True/False*) → (*None*)

Set True/False value to the UFD flag.
Propagation:

- True → domain

- False \rightarrow PID

4.13.12.9 isnoetherian – check Noetherian

isnoetherian(self) \rightarrow *True/False/None*

Return True/False according to the Noetherian flag value being set, otherwise return None.

4.13.12.10 setIsnoetherian – set Noetherian

isnoetherian(self, value: *True/False*) \rightarrow (*None*)

Set True/False value to the Noetherian flag.
Propagation:

- True \rightarrow domain
- False \rightarrow PID

4.13.12.11 isdomain – check domain

isdomain(self) \rightarrow *True/False/None*

Return True/False according to the domain flag value being set, otherwise return None.

4.13.12.12 setIsdomain – set domain

isdomain(self, value: *True/False*) \rightarrow (*None*)

Set True/False value to the domain flag.
Propagation:

- False \rightarrow UFD, Noetherian

4.13.13 `getRingInstance(function)`

`getRingInstance(obj: RingElement) → RingElement`

Return a `RingElement` instance which equals `obj`.

Mainly for python built-in objects such as `int` or `float`.

4.13.14 `getRing(function)`

`getRing(obj: RingElement) → Ring`

Return a ring to which `obj` belongs.

Mainly for python built-in objects such as `int` or `float`.

4.13.15 `inverse(function)`

`inverse(obj: CommutativeRingElement) → QuotientFieldElement`

Return the inverse of `obj`. The inverse can be in the quotient field, if the `obj` is an element of non-field domain.

Mainly for python built-in objects such as `int` or `float`.

4.13.16 `exact_division(function)`

`exact_division(self: RingElement, other: RingElement)
→ RingElement`

Return the division of `self` by `other` if the division is exact.

Mainly for python built-in objects such as `int` or `float`.

Examples

```
>>> print ring.getRing(3)
Z
```

```
>>> print ring.exact_division(6, 3)
2L
```


4.14 vector – ベクトルオブジェクトとその計算

- Classes
 - **Vector**
- Functions
 - **innerProduct**

このモジュールはある例外クラスを提供する。

VectorSizeError : ベクトルのサイズが正しくないことを報告. (主に二つのベクトルの演算において.)

4.14.1 Vector – ベクトルクラス

Vector はベクトルに対するクラス.

Initialize (Constructor)

Vector(*compo: list*) → *Vector*

compo から新しいベクトルオブジェクトを作成. *compo* は整数または **RingElement** のインスタンスである要素のリストでなければならない.

Attributes

compo :
ベクトルの成分を表す.

Operations

数学の世界での標準の通り, インデックスは 1 が最初だということに注意.

operator	explanation
$u+v$	ベクトルの和.
$u-v$	ベクトルの差.
$A*v$	行列とベクトルの積.
$a*v$	ベクトルのスカラー倍.
$v//a$	スカラー除算.
$v\%n$	compo の各要素の <i>n</i> での剰余.
$-v$	各要素の符号を変える.
$u==v$	等しいかどうか.
$u!=v$	等しくないかどうか.
$v[i]$	ベクトルの <i>i</i> 番目の成分を返す.
$v[i] = c$	ベクトルの <i>i</i> 番目の成分を <i>c</i> に置き換える.
$len(v)$	compo の長さを返す.
$repr(v)$	compo の repr 文字列を返す.
$str(v)$	compo の string 文字列を返す.

Examples

```
>>> A = vector.Vector([1, 2])
>>> A
Vector([1, 2])
>>> A.compo
[1, 2]
```

```
>>> B = vector.Vector([2, 1])
>>> A + B
Vector([3, 3])
>>> A % 2
Vector([1, 0])
>>> A[1]
1
>>> len(B)
2
```

Methods

4.14.1.1 copy – 自身のコピー

`copy(self) → Vector`

`self` のコピーを返す.

4.14.1.2 set – 他の `compo` を設定

`set(self, compo: list) → (None)`

`self` の **compo** を新しい `compo` で置き換える.

4.14.1.3 indexOfNoneZero – 0 でない最初の位置

`indexOfNoneZero(self) → integer`

`self.compo` の 0 でない成分の最初のインデックスを返す.

† **compo** の全ての成分が 0 の場合, `ValueError` が起こる.

4.14.1.4 toMatrix – Matrix オブジェクトに変換

`toMatrix(self, as_column: bool=False) → Matrix`

createMatrix 関数を使い **Matrix** オブジェクトを返す.

もし `as_column` が `True` なら, `self` を縦ベクトルとみなした行列を返す. さもなくば, `self` を横ベクトルとみなした行列を返す.

Examples

```
>>> A = vector.Vector([0, 4, 5])
>>> A.indexOfNoneZero()
2
>>> print A.toMatrix()
0 4 5
>>> print A.toMatrix()
```

0
4
5

4.14.2 innerProduct(function) – 内積

`innerProduct(bra: Vector, ket: Vector) → RingElement`

`bra` と `ket` の内積を返す.

この関数は複素数体上の元に対するエルミート内積もサポートする.

† 返される値は成分の型に依存することに注意.

Examples

```
>>> A = vector.Vector([1, 2, 3])
>>> B = vector.Vector([2, 1, 0])
>>> vector.innerProduct(A, B)
4
>>> C = vector.Vector([1+1j, 2+2j, 3+3j])
>>> vector.innerProduct(C, C)
(28+0j)
```

4.15 factor.ecm – ECM factorization

This module has curve type constants:

S : aka SUYAMA. Suyama’s parameter selection strategy.

B : aka BERNSTEIN. Bernstein’s parameter selection strategy.

A1 : aka ASUNCION1. Asuncion’s parameter selection strategy variant 1.

A2 : aka ASUNCION2. ditto 2.

A3 : aka ASUNCION3. ditto 3.

A4 : aka ASUNCION4. ditto 4.

A5 : aka ASUNCION5. ditto 5.

See J.S.Asuncion’s master thesis [11] for details of each family.

4.15.1 ecm – elliptic curve method

```
ecm(n: integer, curve_type: curvetype=A1, incs: integer=3, trials:
integer=20, verbose: bool=False)
    → integer
```

楕円曲線法を使って n の要素を探す。

n の非自明な要素が見つからなければ 1 を返す。

curve_type は **curvetype** の中から選ぶ。

incs specifies a number of changes of bounds. The function repeats factorization trials several times changing curves with a fixed bounds.

Optional argument trials can control how quickly move on to the next higher bounds.

verbose toggles verbosity.

4.16 factor.find – find a factor

このモジュールの方法は与えられた整数に対して一つの要素を返す。非自明な要素 w さがすることができない場合は 1 を返す。しかし 1 も要素であることお忘れなく。

verbose boolean flag can be specified for verbose reports. このメッセージを受け取るため、logger を準備してください。 ([logging](#) 参照。)

4.16.1 trialDivision – trial division

trialDivision(n: integer, **options) → integer

試割り算によって得る n の要素を返す。

options は以下のどちらか:

1. start と stop は範囲パラメータ。さらには step も利用可。
2. iterator は素数のイテレータ。

options が与えられない場合、この関数は非自明な要素が見つかるまで n を素数 2 から n の二乗までの数で割っていく。 verbose boolean flag can be specified for verbose reports.

4.16.2 pmom – $p - 1$ method

pmom(n: integer, **options) → integer

$p - 1$ 法を使い n の要素を返す。

この関数は [\[13\]](#) のアルゴリズム 8.8.2 ($p - 1$ first stage) を使って n の非自明な要素を探すよう試みる。

$n = 2^i$ の場合、この関数はループにおちいる。自然法によってこの方法は自明な要素しか返さないかもしれない。

verbose Boolean flag can be specified for verbose reports, though it is not so verbose indeed.

4.16.3 rhomethod – ρ method

rhomethod(n: integer, **options) → integer

Pollard の ρ 法より n の要素を返す。

この実装は [15] の説明に言及する。自然法によって因数分解は自明な要素しか返さないかもしれない。

verbose Boolean flag can be specified for verbose reports.

Examples

```
>>> factor.find.trialDivision(1001)
7
>>> factor.find.trialDivision(1001, start=10, stop=32)
11
>>> factor.find.pmom(1001)
91
>>> import logging
>>> logging.basicConfig()
>>> factor.find.rhomethod(1001, verbose=True)
INFO:nzmath.factor.find:887 748
13
```

4.17 factor.methods – factoring methods

It uses methods of **factor.find** module or some heavier methods of related modules to find a factor. Also, classes of **factor.util** module is used to track the factorization process. options are normally passed to the underlying function without modification.

This module uses the following type:

factorlist :

factorlist is a list which consists of pairs (**base**, **index**). Each pair means $base^{index}$. The product of these terms expresses prime factorization.

4.17.1 factor – easiest way to factor

```
factor(n: integer, method: string='default', **options )  
→ factorlist
```

Factor the given positive integer **n**.

By default, use several methods internally.

The optional argument method can be:

- 'ecm': use elliptic curve method.
- 'mpqs': use MPQS method.
- 'pmom': use $p - 1$ method.
- 'rhomethod': use Pollard's ρ method.
- 'trialDivision': use trial division.

(†In fact, the initial letter of method name suffices to specify.)

4.17.2 ecm – elliptic curve method

```
ecm(n: integer, **options ) → factorlist
```

Factor the given integer **n** by elliptic curve method.

(See **ecm** of **factor.ecm** module.)

4.17.3 mpqs – multi-polynomial quadratic sieve method

`mpqs(n: integer, **options) → factorlist`

Factor the given integer `n` by multi-polynomial quadratic sieve method.

(See `mpqsfind` of `factor.mpqs` module.)

4.17.4 pmom – $p - 1$ method

`pmom(n: integer, **options) → factorlist`

Factor the given integer `n` by $p - 1$ method.

The method may fail unless `n` has an appropriate factor for the method.
(See `pmom` of `factor.find` module.)

4.17.5 rhomethod – ρ method

`rhomethod(n: integer, **options) → factorlist`

Factor the given integer `n` by Pollard's ρ method.

The method is a probabilistic method, possibly fails in factorizations.
(See `rhomethod` of `factor.find` module.)

4.17.6 trialDivision – trial division

`trialDivision(n: integer, **options) → factorlist`

Factor the given integer `n` by trial division.

options for the trial sequence can be either:

1. `start` and `stop` as range parameters.
2. `iterator` as an iterator of primes.
3. `eratosthenes` as an upper bound to make prime sequence by sieve.

If none of the options above are given, the function divides `n` by primes from 2 to the floor of the square root of `n` until a non-trivial factor is found.
(See `trialDivision` of `factor.find` module.)

Examples

```
>>> factor.methods.factor(10001)
[(73, 1), (137, 1)]
>>> factor.methods.ecm(1000001)
[(101L, 1), (9901L, 1)]
```

4.18 factor.misc – miscellaneous functions related factoring

- Functions
 - `allDivisors`
 - `primeDivisors`
 - `primePowerTest`
 - `squarePart`
- Classes
 - `FactoredInteger`

4.18.1 allDivisors – all divisors

`allDivisors(n: integer) → list`

`n` で割ったすべての要素の値をリストとして返す。

4.18.2 primeDivisors – prime divisors

`primeDivisors(n: integer) → list`

`n` で割ったすべての素数である要素の値をリストとして返す。

4.18.3 primePowerTest – prime power test

`primePowerTest(n: integer) → (integer, integer)`

Judge whether `n` is of the form p^k with a prime p もし正しいのなら (p, k) を返し、さもなければ $(n, 0)$ を返す。

この関数は Algo. 1.7.5 in [13] に基づいている。

4.18.4 squarePart – square part

`squarePart(n: integer) → integer`

`n` を割り切る最大の整数の二乗の値を返す。

Examples

```
>>> factor.misc.allDivisors(1001)
[1, 7, 11, 13L, 77, 91L, 143L, 1001L]
>>> factor.misc.primeDivisors(100)
[2, 5]
>>> factor.misc.primePowerTest(128)
(2, 7)
>>> factor.misc.squarePart(128)
8L
```

4.18.5 FactoredInteger – integer with its factorization

Initialize (Constructor)

```
FactoredInteger(integer: integer, factors: dict={})  
    → FactoredInteger
```

Integer with its factorization information.

If `factors` is given, it is a dict of type `prime:exponent` and the product of $prime^{exponent}$ is equal to the integer. Otherwise, factorization is carried out in initialization.

```
from _partial_factorization(cls, integer: integer, partial: dict)  
    → FactoredInteger
```

A class method to create a new **FactoredInteger** object from partial factorization information `partial`.

Operations

operator	explanation
<code>F * G</code>	multiplication (other operand can be an int)
<code>F ** n</code>	powering
<code>F == G</code>	equal
<code>F != G</code>	not equal
<code>F % G</code>	remainder (the result is an int)
<code>F // G</code>	same as exact_division method
<code>str(F)</code>	string
<code>int(F)</code>	convert to Python integer (forgetting factorization)

Methods

4.18.5.1 `is_divisible_by`

```
is_divisible_by(self, other: integer/FactoredInteger)  
    → bool
```

`other` が `self` 割り切ったのなら `True` と返す。

4.18.5.2 `exact_division`

```
exact_division(self, other: integer/FactoredInteger)  
    → FactoredInteger
```

`other` で割るとき、`other` は `self` で必ず割り切る。

4.18.5.3 `divisors`

```
divisors(self) → list
```

すべての除数をリストとして返す。

4.18.5.4 `proper_divisors`

```
proper_divisors(self) → list
```

1 と `self` を含まないすべての除数をリストとして返す。

4.18.5.5 `prime_divisors`

```
prime_divisors(self) → list
```

すべての素数の除数をリストとして返す。

4.18.5.6 `square_part`

```
square_part(self, asfactored: bool=False) → integer/FactoredInteger object
```

`self` を割る最大の整数の値を返す。

If an optional argument `asfactored` is true, then the result is also a **Factored-Integer object**. (default is False)

4.18.5.7 `squarefree_part`

`squarefree_part(self, asfactored: bool=False) → integer/FactoredInteger object`

`self` を割り、二乗にならない最大の整数の値を返す。

If an optional argument `asfactored` is true, then the result is also a **Factored-Integer object**. (default is False)

4.18.5.8 `copy`

`copy(self) → FactoredInteger object`

自分自身をコピーした値を返す。

4.19 `factor.mpqqs` – MPQS

4.19.1 `mpqsfind`

`mpqsfind(n: integer, s: integer=0, f: integer=0, m: integer=0, verbose: bool=False) → integer`

`n` の要素を複数次多項式二次ふるい法によって探す。

複数次多項式二次ふるい法は巨大な数を因数分解する際に有効である。

`s` はふるいの範囲である。`f` は因子の数で、`m` は乗数。これらが明らかでない時、この関数は `n` から推測する。

4.19.2 `mpqs`

`mpqs(n: integer, s: integer=0, f: integer=0, m: integer=0) → factorlist`

複数次多項式二次ふるい法により `n` を素因数分解する。

mpqsfind と同様である。

4.19.3 eratosthenes

`eratosthenes(n: integer) → list`

`n` までの素数を列挙する。

4.20 factor.util – utilities for factorization

- Classes
 - **FactoringInteger**
 - **FactoringMethod**

This module uses following type:

factorlist :

factorlist is a list which consists of pairs (**base**, **index**). Each pair means $base^{index}$. The product of those terms expresses whole prime factorization.

4.20.1 FactoringInteger – keeping track of factorization

Initialize (Constructor)

FactoringInteger(number: *integer*) → *FactoringInteger*

This is the base class for factoring integers.

number is stored in the attribute **number**. The factors will be stored in the attribute **factors**, and primality of factors will be tracked in the attribute **primality**.

The given number must be a composite number.

Attributes

number :

The composite number.

factors :

Factors known at the time being referred.

primality :

A dictionary of primality information of known factors. **True** if the factor is prime, **False** composite, or **None** undetermined.

Methods

4.20.1.1 getNextTarget – next target

`getNextTarget(self, cond: function=None) → integer`

Return the next target which meets cond.

If cond is not specified, then the next target is a composite (or undetermined) factor of **number**.

cond should be a binary predicate whose arguments are base and index.
If there is no target factor, **LookupError** will be raised.

4.20.1.2 getResult – result of factorization

`getResult(self) → factors`

number の因数分解をする。

4.20.1.3 register – register a new factor

`register(self, divisor: integer, isprime: bool=None) →`

divisor が本当にある数を割るとき、**number** の divisor を記憶する。

その数は divisor により可能な限り割られる。

isprime tells the primality of the divisor (default to undetermined).

4.20.1.4 sortFactors – sort factors

`sortFactors(self) →`

要素のリストを並べる。

この関数は **getResult** に関係している。

Examples

```
>>> A = factor.util.FactoringInteger(100)
>>> A.getNextTarget()
100
```

```
>>> A.getResult()
[(100, 1)]
>>> A.register(5, True)
>>> A.getResult()
[(5, 2), (4, 1)]
>>> A.sortFactors()
>>> A.getResult()
[(4, 1), (5, 2)]
>>> A.primalities
{4: None, 5: True}
>>> A.getNextTarget()
4
```

4.20.2 FactoringMethod – method of factorization

Initialize (Constructor)

FactoringMethod() → *FactoringMethod*

Base class of factoring methods.

すべての方法は **factor.methods** で定義されている。implemented as derived classes of this class. この方法は **factor** と呼ぶこともある。 他の方法は

Methods

4.20.2.1 factor – do factorization

```
factor(self, number: integer, return_type: str='list', need_sort:
    bool=False )
    → factorlist
```

与えられた正の整数 `number` の因数分解を行う。

不履行の場合は **factorlist** を返す。

A keyword option `return_type` can be as the following:

1. 'list' for default type (**factorlist**).
2. 'tracker' for **FactoringInteger**.

Another keyword option `need_sort` is Boolean: `True` to sort the result. This should be specified with `return_type='list'`.

4.20.2.2 †continue_factor – continue factorization

```
continue_factor(self, tracker: FactoringInteger, return_type:
    str='tracker', primeq: func=primeq )
    → FactoringInteger
```

Continue factoring of the given tracker and return the result of factorization.

The default returned type is **FactoringInteger**, but if `return_type` is specified as 'list' then it returns **factorlist**. The primality is judged by a function specified in `primeq` optional keyword argument, which default is **primeq**.

4.20.2.3 †find – find a factor

```
find(self, target: integer, **options ) → integer
```

`target` から要素を探す。

この方法は優先されるべきである。または **factor** 法も

4.20.2.4 †generate – generate prime factors

```
generate(self, target: integer, **options ) → integer
```

Generate prime factors of the `target` number with their valuations.

この関数が $(1, 1)$ を返したら因数分解は不完全であることを示す。to indicate the factorization is incomplete.
This method has to be overridden, or **factor** method should be overridden not to call this method.

4.21 poly.factor – 多項式の因数分解

factor モジュールは整数係数一変数多項式の因数分解のためのもの。
このモジュールは以下に示す型を使用:

polynomial :

polynomial は `poly.uniutil.polynomial` によって生成された多項式。

4.21.1 brute_force_search – 総当たりで因数分解を探す

```
brute_force_search(f: poly.uniutil.IntegerPolynomial, fp_factors:
list, q: integer)
→ [factors]
```

`fp_factors` 上でいくつかの積の組み合わせである因数を探すことにより `f` の因数分解を見つける。この組み合わせは総当たりで探される。

引数 `fp_factors` は `poly.uniutil.FinitePrimeFieldPolynomial` のリストです。

4.21.2 divisibility_test – 可除性テスト

```
divisibility_test(f: polynomial, g: polynomial) → bool
```

多項式において、`f` が `g` で割り切れるかどうか、Boolean 値を返す。

4.21.3 minimum_absolute_injection – 係数を絶対値最小表現に渡す

```
minimum_absolute_injection(f: polynomial) → F
```

各係数を絶対値最小表現に渡す $\mathbb{Z}/p\mathbb{Z}$ 係数多項式 `f` の単射により整数係数多項式 `F` を返す。

与えられた多項式 `f` の係数環は **IntegerResidueClassRing** または **FinitePrimeField** でなければならない。

4.21.4 padic_factorization – p 進分解

```
padic_factorization(f: polynomial) → p, factors
```

素数 `p` と、与えられた平方因子を含まない整数係数多項式 `f` の `p` 進分解を返す。結果である `factors` は整数係数を持ち、 \mathbb{F}_p からその絶対値最小表現に写されている。

† 素数は以下のように選ばれる:

1. $f \bmod p$ でも平方因子を持たない,

2. 因数の数は次の素数を超えない.

与えられた多項式 f は `poly.uniutil.IntegerPolynomial` でなければならない.

4.21.5 `upper_bound_of_coefficient` – Landau-Mignotte の係数の上界

`upper_bound_of_coefficient(f: polynomial) → long`

次数は与えられた f の次数の半分を超えない大きさである Landau-Mignotte の因数の係数の上界を計算.

与えられた多項式 f は整数係数多項式でなければならない.

4.21.6 `zassenhaus` – Zassenhaus 法による平方因子のない整数係数多項式の因数分解

`zassenhaus(f: polynomial) → list of factors f`

Berlekamp-Zassenhaus 法による平方数のない整数係数の多項式 f の因数.

4.21.7 `integerpolynomialfactorization` – 整数多項式の因数分解

`integerpolynomialfactorization(f: polynomial) → factor`

Berlekamp-Zassenhaus 法により整数係数多項式 f を因数分解.

因数は `(factor, index)` という形式のタプルのリストの形式で出力される.

4.22 poly.formalsum – 形式和

- Classes

- †**FormalSumContainerInterface**
- **DictFormalSum**
- †**ListFormalSum**

形式和とは数学的な項の有限和で、項は二つの部分から成る:係数と基数. 形式和での全ての係数は共通の環に属し、一方で基数は任意.

二つの形式和は次に示す方法で足される. もし基数が共通である項があれば、それらは同じ基数と加えられた係数を持つ新しい項にまとめられる.

係数は基数より参照することができる. もし特定の基数が形式和に現れない場合、それは `null` を返す.

便宜上、`terminit` として次を参照:

`terminit` :

`terminit` は `dict` の初期化の型の一つを意味する. それにより構成された辞書は基数から係数への写像として考えられる.

Note for beginner **DictFormalSum** のみ使うことが必要となるかもしれないが、インターフェース (全てのメソッドの名前と意味付け) はその内で定義されているので **FormalSumContainerInterface** の説明を読まなければならないかもしれない.

4.22.1 FormalSumContainerInterface – インターフェースクラス

Initialize (Constructor)

インターフェースは抽象的なクラスなので、インスタンスは作らない。

インターフェースは“形式和”は何かということを定義している。派生クラスには以下に示す演算とメソッドを定義しなければならない。

Operations

operator	explanation
$f + g$	和
$f - g$	差
$-f$	符号の変更
$+f$	新しいコピー
$f * a, a * f$	スカラー a 倍
$f == g$	等しいかどうか返す
$f != g$	等しくないかどうか返す
$f[b]$	基底 b に対応した係数を返す
$b \text{ in } f$	基底 b が f に含まれているかどうか返す
$\text{len}(f)$	項の数
$\text{hash}(f)$	ハッシュ

Methods

4.22.1.1 `construct_with_default` – コピーを構成

`construct_with_default(self, maindata: terminit) → FormalSumContainerInterface`

`maindata` のみ与えられた (必要なら `self` が持つ情報を使用), `self` と同じクラスの新しい形式和を作成.

4.22.1.2 `iterterms` – 項のイテレータ

`iterterms(self) → iterator`

項のイテレータを返す.
イテレータより生成されたそれぞれの項は `(base, coefficient)` という組.

4.22.1.3 `itercoefficients` – 係数のイテレータ

`itercoefficients(self) → iterator`

係数のイテレータを返す.

4.22.1.4 `iterbases` – 基数のイテレータ

`iterbases(self) → iterator`

基数のイテレータを返す.

4.22.1.5 `terms` – 項のリスト

`terms(self) → list`

項のリストを返す.
返されるリストのそれぞれの項は `(base, coefficient)` という組.

4.22.1.6 `coefficients` – 係数のリスト

`coefficients(self) → list`

係数のリストを返す.

4.22.1.7 bases – 基数のリスト

`bases(self) → list`

基数のリストを返す.

4.22.1.8 terms_map – 項に写像を施す

`terms_map(self, func: function) → FormalSumContainerInterface`

項に写像を施す, すなわち, それぞれの項に `func` を適用することにより新しい形式和を作成.

`func` は `base` と `coefficient` という二つのパラメータをとらなければならない, その後新しい項の組を返す.

4.22.1.9 coefficients_map – 係数に写像を施す

`coefficients_map(self, func: function) → FormalSumContainerInterface`

係数に写像を施す, すなわち, 各係数に `func` を適用することにより新しい形式和を作成.

`func` は `coefficient` という一つのパラメータをとり, その後新しい係数を返す.

4.22.1.10 bases_map – 基数に写像を施す

`bases_map(self, func: function) → FormalSumContainerInterface`

基数に写像を施す, すなわち, 各基数に `func` を適用することにより新しい形式和を作成.

`func` は `base` という一つのパラメータをとり, その後新しい基数を返す.

4.22.2 DictFormalSum – 辞書で実装された形式和

`dict` を基に実装された形式和.

このクラスは **FormalSumContainerInterface** を継承. インターフェースの全てのメソッドは実装される.

Initialize (Constructor)

```
DictFormalSum(args: terminit, defaultvalue: RingElement=None)  
→ DictFormalSum
```

`args` の型については **terminit** を参照. 基数から係数への写像を作る.
任意引数 `defaultvalue` は `__getitem__` への初期設定値, すなわち, もし指定の基数に関する項がなかったら検索を試み `defaultvalue` を返す. 従ってそれは他の係数が所属している環の元である.

4.22.3 ListFormalSum – リストで実装された形式和

リストを基に実装された形式和.

FormalSumContainerInterface を継承. インターフェースの全てのメソッドは実装される.

Initialize (Constructor)

```
ListFormalSum(args: terminit, defaultvalue: RingElement=None)  
→ ListFormalSum
```

`args` の型については **terminit** を参照. 基数から係数への写像を作る.
任意引数 `defaultvalue` は `__getitem__` への初期設定値, すなわち, もし指定の基数に関する項がなかったら, 検索を試み `defaultvalue` を返す. 従ってそれは他の係数が所属している環の元である.

4.23 poly.groebner – グレブナー基底

groebner モジュールは多変数多項式イデアルに対するグレブナー基底を計算するためのもの.

このモジュールは以下に示す型を使用:

polynomial :

polynomial は関数 **polynomial** によって生み出された多項式.

order :

order は多項式の項順序.

4.23.1 buchberger – グレブナー基底を得るための素朴なアルゴリズム

buchberger(generating: *list*, order: *order*) → [*polynomials*]

order についての与えられた多項式の生成集合により生成されるイデアルのグレブナー基底を返す.

この実装は非常に素朴なものだということに注意.

引数 generating は **Polynomial** のリスト; 引数 order は項順序.

4.23.2 normal_strategy – グレブナー基底を得る普通のアルゴリズム

normal_strategy(generating: *list*, order: *order*) → [*polynomials*]

order についての与えられた多項式の生成集合により生成されるイデアルのグレブナー基底を返す.

この関数は ‘普通の戦略’ を使用.

引数 generating は **Polynomial** のリスト; 引数 order は項順序.

4.23.3 reduce_groebner – 簡約グレブナー基底

reduce_groebner(gbasis: *list*, order: *order*) → [*polynomials*]

グレブナー基底から構成された簡約グレブナー基底を返す.

出力は以下を満たす:

- $\text{lb}(f)$ は $\text{lb}(g)$ を割り切る $\Rightarrow g$ は簡約グレブナー基底ではない.

- モニック多項式.

引数 `gbasis` は多項式のリストで,(単に生成集合であるだけでなく) グレブナー基底.

4.23.4 `s_polynomial` – S-polynomial

`s_polynomial(f: polynomial, g: polynomial, order: order)`
 $\rightarrow [polynomials]$

`order` についての `f` と `g` の S-多項式を返す.

$$S(f, g) = (\text{lc}(g) * T / \text{lb}(f)) * f - (\text{lc}(f) * T / \text{lb}(g)) * g,$$

$$T = \text{lcm}(\text{lb}(f), \text{lb}(g)).$$

Examples

```
>>> f = multiutil.polynomial({(1,0):2, (1,1):1},rational.theRationalField, 2)
>>> g = multiutil.polynomial({(0,1):-2, (1,1):1},rational.theRationalField, 2)
>>> lex = termorder.lexicographic_order
>>> groebner.s_polynomial(f, g, lex)
UniqueFactorizationDomainPolynomial({(1, 0): 2, (0, 1): 2})
>>> gb = groebner.normal_strategy([f, g], lex)
>>> for gb_poly in gb:
...     print gb_poly
...
UniqueFactorizationDomainPolynomial({(1, 1): 1, (1, 0): 2})
UniqueFactorizationDomainPolynomial({(1, 1): 1, (0, 1): -2})
UniqueFactorizationDomainPolynomial({(1, 0): 2, (0, 1): 2})
UniqueFactorizationDomainPolynomial({(0, 2): -2, (0, 1): -4.0})
>>> gb_red = groebner.reduce_groebner(gb, lex)
>>> for gb_poly in gb_red:
...     print gb_poly
...
UniqueFactorizationDomainPolynomial({(1, 0): Rational(1, 1), (0, 1): Rational(1, 1)})
UniqueFactorizationDomainPolynomial({(0, 2): Rational(1, 1), (0, 1): 2.0})
```

4.24 `poly.hensel` – ヘンゼルリフト

- Classes
 - †**HenselLiftPair**
 - †**HenselLiftMulti**

- † **HenselLiftSimultaneously**
- **Functions**
 - **lift_upto**

このモジュールドキュメント内では, *polynomial* は整数係数多項式を意味.

4.24.1 HenselLiftPair – ヘンゼルリフトの組

Initialize (Constructor)

```
HenselLiftPair(f: polynomial, a1: polynomial, a2: polynomial, u1: polynomial, u2: polynomial, p: integer, q: integer=p)  
→ HenselLiftPair
```

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式を保存.

引数は以下の前提条件を満たさなければならない:

- $f, a1$ そして $a2$ はモニック多項式
- $f == a1*a2 \pmod{q}$
- $a1*u1 + a2*u2 == 1 \pmod{p}$
- p は q を割り切り, どちらも自然数

```
from_factors(f: polynomial, a1: polynomial, a2: polynomial, p: integer)  
→ HenselLiftPair
```

これは `HenselLiftPair` のインスタンスを作成し返すクラスメソッド. 初期構成のために $u1$ と $u2$ を計算し直す必要はない; これらは他の引数から用意される.

引数は以下の前提条件を満たすべきである:

- $f, a1$ と $a2$ はモニック多項式
- $f == a1*a2 \pmod{p}$
- p は素数

Attributes

`point` :
リストとしての因数 $a1, a2$.

Methods

4.24.1.1 lift – 一段階引き上げる

`lift(self) →`

いわゆる二次方程式法により多項式を引き上げる.

4.24.1.2 lift_factors – a_1 と a_2 を引き上げる

`lift_factors(self) →`

整数係数多項式 A_i たちを引き上げることににより因数を更新:

- $f == A_1 * A_2 \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, 2)$

さらに, q は $p * q$ に更新される.

† 次の前提条件は自動的に満たされる:

- $f == a_1 * a_2 \pmod{q}$
- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$
- p は q を割り切る

4.24.1.3 lift_ladder – u_1 と u_2 を引き上げる

`lift_ladder(self) →`

u_1 と u_2 を U_1 と U_2 に更新:

- $a_1 * U_1 + a_2 * U_2 == 1 \pmod{p^{**2}}$
- $U_i == u_i \pmod{p} \ (i = 1, 2)$

そして, p を p^{**2} に更新.

† 次の前提条件は自動的に満たされる:

- $a_1 * u_1 + a_2 * u_2 == 1 \pmod{p}$

4.24.2 HenselLiftMulti – 複数多項式に対するヘンゼルリフト

Initialize (Constructor)

`HenselLiftMulti(f: polynomial, factors: list, ladder: tuple, p: integer, q: integer=p)`
`→ HenselLiftMulti`

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式の因数を保存. もし因数の数が二つなら, **HenselLiftPair** を使うべきである.

`factors` は多項式のリスト; これらの多項式は二つのリスト `sis` と `tis` のタプルである `a1, a2, ... ladder` として表し, 両リストは多項式から成る. `s1, s2, ...` として `sis` の多項式を表し, `t1, t2, ...` として `tis` の多項式を表す. さらに, b_i を $i < j$ である a_j たちの積として定義. 引数は以下の前提条件を満たす:

- f と全ての `factors` はモニック多項式
- $f == a_1 * \dots * a_r \pmod{q}$
- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, 2, \dots, r)$
- p は q を割り切り, どちらも自然数

```
from _factors(f: polynomial, factors: list, p: integer)
    → HenselLiftMulti
```

これは `HenselLiftMulti` のインスタンスを作成し返すためのクラスメソッド. 初期構成のために `ladder` を計算し直す必要はない; これらは他の引数によって用意される.

引数は前提条件を満たすべきである:

- f と全ての `factors` はモニック多項式
- $f == a_1 * \dots * a_r \pmod{q}$
- p は素数

Attributes

`point` :

リストとしての因数 a_i たち.

Methods

4.24.2.1 lift – 一段階引き上げる

`lift(self) →`

いわゆる二次方程式法により多項式を引き上げる.

4.24.2.2 lift_factors – 因数を引き上げる

`lift_factors(self) →`

整数係数多項式 A_i たちを引き上げることににより因数を更新:

- $f == A_1 * \dots * A_r \pmod{p * q}$
- $A_i == a_i \pmod{q} \ (i = 1, \dots, r)$

さらに, q は $p * q$ に更新.

† 次の前提条件は自動的に満たされる:

- $f == a_1 * \dots * a_r \pmod{q}$
- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, \dots, r)$
- p は q を割り切る

4.24.2.3 lift_ladder – u_1 と u_2 を引き上げる

`lift_ladder(self) →`

s_i たちと t_i たちを S_i たちと T_i たちに更新:

- $a_1 * S_i + b_i * T_i == 1 \pmod{p^{**2}}$
- $S_i == s_i \pmod{p} \ (i = 1, \dots, r)$
- $T_i == t_i \pmod{p} \ (i = 1, \dots, r)$

そして, p を p^{**2} に更新.

† 次の前提条件は自動的に満たされる:

- $a_i * s_i + b_i * t_i == 1 \pmod{p} \ (i = 1, \dots, r)$

4.24.3 HenselLiftSimultaneously

このメソッドは [14] で説明されている.

† 以下の不変式を保存:

- a_i たち, p_i と g_i たちはすべてモニック多項式
- $f == g_1 * \dots * g_r \pmod{p}$
- $f == d_0 + d_1 * p + d_2 * p^2 + \dots + d_k * p^k$
- $h_i == g_{(i+1)} * \dots * g_r$
- $1 == g_i * s_i + h_i * t_i \pmod{p} \ (i = 1, \dots, r)$
- $\deg(s_i) < \deg(h_i), \deg(t_i) < \deg(g_i) \ (i = 1, \dots, r)$
- p は q を割り切る
- $f == l_1 * \dots * l_r \pmod{q/p}$
- $f == a_1 * \dots * a_r \pmod{q}$
- $u_i == a_i * y_i + b_i * z_i \pmod{p} \ (i = 1, \dots, r)$

Initialize (Constructor)

```
HenselLiftSimultaneously(target: polynomial, factors: list, cofactors:
list, bases: list, p: integer)
    → HenselLiftSimultaneously
```

このオブジェクトはヘンゼルの補題によって引き上げられていく整数係数多項式の因数を保存.

```
f = target, gi in factors, his in cofactors and sis and tis are in bases.
from _factors(target: polynomial, factors: list, p: integer, ubound: in-
teger=sys.maxint)
    → HenselLiftSimultaneously
```

これは、因数が **HenselLiftMulti** によって引き上げられた、**HenselLiftSimultaneously** のインスタンスを作成し返すためのクラスメソッドで、**HenselLiftMulti** はもし `sys.maxint` より小さければ `ubound` と一致し、さもなければ `sys.maxint` と一致する. 初期構成を補助する多項式を計算し直す必要はない; これらは他の引数によって用意される.

```
f = target, gis in factors.
```


Methods

4.24.3.1 lift – 一段階引き上げる

`lift(self) →`

引き上げメソッド. このメソッドのみ呼び出すべき.

4.24.3.2 first_lift – 最初のステップ

`first_lift(self) →`

引き上げを開始.

`f == l1*l2*...*lr (mod p**2)`

`di` たち,`ui` たち,`yi` たちそして `zi` たちの初期化. `ai` たちと,`bi` たちを更新. そして,`q` を `p**2` に更新.

4.24.3.3 general_lift – 次のステップ

`general_lift(self) →`

引き上げを続ける.

`f == a1*a2*...*ar (mod p*q)`

`ai` たち,`ubi` たち,`yi` たちそして `zi` たちを初期化. そして,`q` を `p*q.` に更新

4.24.4 lift_upto – main 関数

`lift_upto(self, target: polynomial, factors: list, p: integer, bound: integer)`
`→ tuple`

`bound` まで `target` の `factors mod p` をヘンゼルリフト氏,`factors mod q` と `the q` それ自身を返す.

以下の前提条件は満たされるべきである:

- `target` はモニック多項式.
- `target == product(factors) mod p`

結果 (`factors`, `q`) は以下の前提条件を満たす:

- `k` s.t. `q == p**k >= bound` なる `k` が存在
- `target == product(factors) mod q`

4.25 poly.multiutil – 多変数多項式に対するユーティリティ

- **Classes**

- **RingPolynomial**
- **DomainPolynomial**
- **UniqueFactorizationDomainPolynomial**
- OrderProvider
- NestProvider
- PseudoDivisionProvider
- GcdProvider
- RingElementProvider

- **Functions**

- **polynomial**

4.25.1 RingPolynomial

可換環係数を持つ一般の多項式.

Initialize (Constructor)

```
RingPolynomial(coefficients: termint, **keywords: dict)  
→ RingPolynomial
```

keywords は以下を含まなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項順序 (*TermOrder*)

このクラスは **BasicPolynomial**, **OrderProvider**, **NestProvider** and **RingElementProvider** を継承する.

Attributes

order :
項順序.

Methods

4.25.1.1 getRing

`getRing(self) → Ring`

多項式が所属する *Ring* のサブクラスのオブジェクトを返す.
(このメソッドは *RingElementProvider* 内の定義をオーバーライドする)

4.25.1.2 getCoefficientRing

`getCoefficientRing(self) → Ring`

すべての係数が所属する *Ring* のサブクラスのオブジェクトを返す.
(このメソッドは *RingElementProvider* 内の定義をオーバーライドする)

4.25.1.3 leading_variable

`leading_variable(self) → integer`

主変数 (全ての全次数が 1 の項の中での主項) の位置を返す.
主項は結果として項順序に変化する. 項順序は属性 *order* によって指定される.
(このメソッドは *NestProvider* から継承される)

4.25.1.4 nest

`nest(self, outer: integer, coeffring: CommutativeRing)
→ polynomial`

与えられた位置の変数 *outer* を引き出すことにより多項式をネスト.
(このメソッドは *NestProvider* から継承される)

4.25.1.5 unnest

`nest(self, q: polynomial, outer: integer, coeffring: CommutativeRing)
→ polynomial`

与えられた位置の変数 *outer* を挿入することによりネストされた多項式 *q* をアンネストします.
(このメソッドは *NestProvider* から継承されます)

4.25.2 DomainPolynomial

整域の係数を持つ多項式.

Initialize (Constructor)

DomainPolynomial(coefficients: *terminit*, **keywords: *dict*)
→ *DomainPolynomial*

keywords は以下を含まなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項順序 (*TermOrder*)

このクラスは **RingPolynomial** と **PseudoDivisionProvider** を継承する.

Operations

operator	explanation
f / g	除算 (結果は有理関数)

Methods

4.25.2.1 pseudo_divmod

pseudo_divmod(self, other: *polynomial*) → *polynomial*

以下となる多項式 Q, R を返す:

$$d^{\deg(\text{self})-\deg(\text{other})+1}\text{self} = \text{other} \times Q + R$$

固定値として other の主係数である d .

結果として主係数は項の係数に変わる. 項順序は属性 order によって指定される.

(このメソッドは PseudoDivisionProvider から継承される.)

4.25.2.2 pseudo_floordiv

pseudo_floordiv(self, other: *polynomial*) → *polynomial*

以下となる多項式 Q を返す:

$$d^{\deg(\text{self})-\deg(\text{other})+1}\text{self} = \text{other} \times Q + R$$

固定値として other の主係数 d と 多項式 R .

結果として主係数は項順序に変わる. 項順序は属性 order によって指定される.

(このメソッドは PseudoDivisionProvider から継承される.)

4.25.2.3 pseudo_mod

pseudo_mod(self, other: *polynomial*) → *polynomial*

以下となる多項式 R を返す:

$$d^{\deg(\text{self})-\deg(\text{other})+1} \times \text{self} = \text{other} \times Q + R$$

d は other の主係数で Q は多項式.

結果として主係数は項の位数に変わる. 項順序は属性 order によって指定される.

(このメソッドは PseudoDivisionProvider から継承される.)

4.25.2.4 exact_division

exact_division(self, other: *polynomial*) → *polynomial*

(割り切れるときのみ) 除算で商を返す.

(このメソッドは PseudoDivisionProvider から継承される.)

4.25.3 UniqueFactorizationDomainPolynomial

一意分解整域 (UFD) 係数を持つ多項式.

Initialize (Constructor)

```
UniqueFactorizationDomainPolynomial(coefficients: terminit,  
**keywords: dict)  
    → UniqueFactorizationDomainPolynomial
```

keywords は以下を含まなければならない:

coeffring 可換環 (*CommutativeRing*)

number_of_variables 変数の数 (*integer*)

order 項順序 (*TermOrder*)

このクラスは **DomainPolynomial** と **GcdProvider** を継承する.

Methods

4.25.3.1 gcd

`gcd(self, other: polynomial) → polynomial`

gcd を返す. ネストされた多項式の gcd が使われる.
(このメソッドは GcdProvider から継承される.)

4.25.3.2 resultant

`resultant(self, other: polynomial, var: integer) → polynomial`

その位置 var によって指定された変数についての, 同じ環上の二つの多項式の終結式を返す.

4.25.4 polynomial – さまざまな多項式に対するファクトリ関数

`polynomial(coefficients: terminit, coeffring: CommutativeRing,
number_of_variables: integer=None)
→ polynomial`

多項式を返す.

† 関数が呼ばれる前に次の設定をすることにより, 係数環から多項式の型を選ぶ方法をオーバーライドできる:

```
special_ring_table[coeffring_type] = polynomial_type
```

4.25.5 prepare_indeterminates – 不定元連立宣言

`prepare_indeterminates(names: string, ctx: dict, coeffring: CoefficientRing=None)
→ None`

不定元な names によって分けられた空間から, 不定元を表す変数を用意する. 結果は辞書 ctx に格納される.

変数はすぐに用意されるべきである, さもなくば間違っただ変数のエイリアスが計算を遅くし混乱させるだろう.

もし任意引数の coeffring が与えられなければ, 不定元は整数係数多項式として初期化される.

Examples

```
>>> prepare_indeterminates("X Y Z", globals())
>>> Y
UniqueFactorizationDomainPolynomial({(0, 1, 0): 1})
```


4.26 poly.multivar – 多変数多項式

- Classes
 - †**PolynomialInterface**
 - †**BasicPolynomial**
 - **TermIndices**

4.26.1 PolynomialInterface – 全ての多変数多項式の基底クラス

インターフェースが抽象クラスなのでインスタンスは作らない.

4.26.2 BasicPolynomial – 多項式の基本的な実装

基本的な多項式の種類.

4.26.3 TermIndices – 多変数多項式の項のインデックス

タプルのようなオブジェクト.

Initialize (Constructor)

`TermIndices(indices: tuple) → TermIndices`

コンストラクタは整数性や非負性などのインデックスの正しさを調べない.

Operations

operator	explanation
<code>t == u</code>	等しいかどうか
<code>t != u</code>	等しくないかどうか
<code>t + u</code>	(componentwise) 和
<code>t - u</code>	(componentwise) 差
<code>t * a</code>	(componentwise) スカラーによる積 <code>a</code>
<code>t <= u, t < u, t >= u, t > u</code>	位数
<code>t[k]</code>	<code>k</code> 番目のインデックス
<code>len(t)</code>	長さ
<code>iter(t)</code>	イテレータ
<code>hash(t)</code>	ハッシュ

Methods

4.26.3.1 pop

pop(self, pos: *integer*) → (*integer*, *TermIndices*)

pos におけるインデックスと pos のインデックスを除いた新しい *TermIndices* オブジェクトを返す.

4.26.3.2 gcd

gcd(self, other: *TermIndices*) → *TermIndices*

二つのインデックスの “gcd” を返す.

4.26.3.3 lcm

lcm(self, other: *TermIndices*) → *TermIndices*

二つのインデックスの “lcm” を返す.

4.27 poly.ratfunc – 有理関数

- Classes

- **RationalFunction**

有理関数は二つの多項式の, 分数として書けるもの.

このモジュールが役に立つと期待しないこと. ただ多項式の除算のための無難なコンテナを提供するもの.

4.27.1 RationalFunction – 有理関数クラス

Initialize (Constructor)

```
RationalFunction(numerator: polynomial, denominator: polynomial=1)
→ RationalFunction
```

与えられた `numerator` と `denominator` を持つ有理関数を作る。もし `numerator` が `RationalFunction` のインスタンスで、`denominator` が与えられなければコピーを作る。もし `numerator` が多項式なら、`numerator` が与えられた有理関数を作る。さらに、もし `denominator` がすでに与えられていたら、分母はその値で設定され、さもなければ分母は 1。

Attributes

`numerator` :
多項式。

`denominator` :
多項式。

Operations

operator	explanation
<code>A==B</code>	A と B が等しいかどうか返す。
<code>str(A)</code>	読みやすい文字列を返す。
<code>repr(A)</code>	A の構造表現文字列を返す。

Methods

4.27.1.1 `getRing` – 有理関数体を得る

`getRing(self)` \rightarrow **RationalFunctionField**

有理関数が所属する有理関数体を返す.

4.28 poly.ring – 多項式環

- Classes
 - **PolynomialRing**
 - **RationalFunctionField**
 - **PolynomialIdeal**

4.28.1 PolynomialRing – 多項式環

uni-/multivariate polynomial rings のためのクラス. **CommutativeRing** のためのサブクラス.

Initialize (Constructor)

```
PolynomialRing(coeffring: CommutativeRing, number_of_variables:  
integer=1)  
→ PolynomialRing
```

coeffring は係数環. number_of_variables は変数の数. もしその値が 1 より大きければ, その環は多変数多項式に対するもの.

Attributes

zero :
環上の 0.

one :
環上の 1.

Methods

4.28.1.1 getInstance – クラスメソッド

`getInstance(coeffring: CommutativeRing, number_of_variables: integer)`
→ *PolynomialRing*

係数環 `coeffring` と変数の数 `number_of_variables` を持つ多項式環のインスタンスを返す.

4.28.1.2 getCoefficientRing

`getCoefficientRing()` → *CommutativeRing*

4.28.1.3 getQuotientField

`getQuotientField()` → *Field*

4.28.1.4 issubring

`issubring(other: Ring)` → *bool*

4.28.1.5 issuperring

`issuperring(other: Ring)` → *bool*

4.28.1.6 getCharacteristic

`getCharacteristic()` → *integer*

4.28.1.7 createElement

`createElement(seed)` → *polynomial*

多項式を返す. `seed` は多項式, 係数環の元, または一変数/多変数多項式の最初の引数に適した他のデータであり得る.

4.28.1.8 gcd

`gcd(a, b)` → *polynomial*

(可能ならば) 与えられた多項式の最大公約数を返す. 多項式は多項式環に入っていないなければならない. もし係数環が体ならば, その結果はモニック多項式.

4.28.1.9 isdomain

4.28.1.10 iseclidean

4.28.1.11 isnoetherian

4.28.1.12 ispid

4.28.1.13 isufd

CommutativeRing から継承された.

4.28.2 RationalFunctionField – 有理関数体

Initialize (Constructor)

```
RationalFunctionField(field: Field, number_of_variables: integer)  
→ RationalFunctionField
```

有理関数体に関するクラス. **QuotientField** のサブクラス.

field は **Field** のオブジェクトであるべきである係数体. number_of_variables は変数の数.

Attributes

zero :
体上の 0.

one :
体上の 1.

Methods

4.28.2.1 getInstance – クラスメソッド

```
getInstance(coefffield: Field, number_of_variables: integer)  
→ RationalFunctionField
```

係数体 `coefffield` と変数の数 `number_of_variables` を持つ `RationalFunctionField` のインスタンスを返す.

4.28.2.2 createElement

```
createElement(*seedarg: list, **seedkwd: dict) → RationalFunction
```

4.28.2.3 getQuotientField

```
getQuotientField() → Field
```

4.28.2.4 issubring

```
issubring(other: Ring) → bool
```

4.28.2.5 issuperring

```
issuperring(other: Ring) → bool
```

4.28.2.6 unnest

```
unnest() → RationalFunctionField
```

もし `self` が `RationalFunctionField` にネストされていたら, すなわちその係数体もまた `RationalFunctionField` なら, メソッドは一段階アンネストされた `RationalFunctionField` を返す.

例えば:

Examples

```
>>> RationalFunctionField(RationalFunctionField(Q, 1), 1).unnest()  
RationalFunctionField(Q, 2)
```

4.28.2.7 gcd

```
gcd(a: RationalFunction, b: RationalFunction) → RationalFunction
```

Field から継承される.

4.28.2.8 isdomain

4.28.2.9 iseclidean

4.28.2.10 isnoetherian

4.28.2.11 ispid

4.28.2.12 isufd

CommutativeRing から継承される.

4.28.3 PolynomialIdeal – 多項式環のイデアル

多項式環のイデアルを表す **Ideal** のサブクラス.

Initialize (Constructor)

PolynomialIdeal(generators: *list*, polyring: *PolynomialRing*)
→ *PolynomialIdeal*

generators によって生成される多項式環 polyring のイデアルを表す新しいオブジェクトを作成.

Operations

operator	explanation
in	含まれているかのテスト
==	同じイデアルか?
!=	異なるイデアルか?
+	和
*	積

Methods

4.28.3.1 reduce

`reduce(element: polynomial) → polynomial`

イデアルを法とする `element` の剰余.

4.28.3.2 issubset

`issubset(other: set) → bool`

4.28.3.3 issuperset

`issuperset(other: set) → bool`

4.29 poly.termorder – 項順序

- **Classes**
 - †**TermOrderInterface**
 - †**UnivarTermOrder**
 - **MultivarTermOrder**
- **Functions**
 - **weight_order**

4.29.1 TermOrderInterface – 項順序のインターフェース

Initialize (Constructor)

`TermOrderInterface(comparator: function) → TermOrderInterface`

項順序は主に二つの項 (または単項式) の優先順位を決定する関数. 優先順位により, 全ての項は順序付けられる.

より正確に言うと, Python の形式では, 項順序は整数での二つのタプルをとり, そのそれぞれのタプルは項のべき指数を表す. そして組み込み関数の `cmp` のようにただ 0, 1 または -1 を返す.

A `TermOrder` オブジェクトは優先順位関数だけでなく, 次数や主係数などが記された, 多項式のフォーマットされた文字列を返すメソッドも提供.

`comparator` は整数での二つのタプルのようなオブジェクトをとり, それぞれのタプルは項のべき指数を表す. そして組み込み関数 `cmp` のようにただ 0, 1 または -1 を返す.

このクラスは抽象クラスでインスタンスが作られるべきではない. `k` のメソッドは下にオーバーライドされなければならない.

Methods

4.29.1.1 cmp

`cmp(self, left: tuple, right: tuple) → integer`

二つのインデックスタプル `left` と `right` を比較し優先順位を決定.

4.29.1.2 format

`format(self, polynom: polynomial, **keywords: dict) → string`

多項式 `polynom` のフォーマットされた文字列を返す.

4.29.1.3 leading_coefficient

`leading_coefficient(self, polynom: polynomial) → CommutativeRingElement`

多項式 `polynom` の項順序についての主係数を返す.

4.29.1.4 leading_term

`leading_term(self, polynom: polynomial) → tuple`

多項式 `polynom` の主項を項順序についてのタプル (`degree index`, `coefficient`) として返す.

4.29.2 UnivarTermOrder — 一変数多項式に対する項順序

Initialize (Constructor)

`UnivarTermOrder(comparator: function) → UnivarTermOrder`

一変数多項式に対しては一意的な項順序がある. 次数として知られている.

一変数の場合への特別なことは, べき数はタプルではなく, 単なる整数であるということである. このことから, メソッド signatures もまた `TermOrderInterface` 内の定義から変換する必要があるが, それは容易なため説明は省略.

`comparator` は二つの整数をとり, `cmp` のようにただ 0, 1 または -1 を返すために呼ばれ得る, すなわち, もしそれらが 0 を返す, 最初は 1 より大きい, そしてさもなれば -1. 理論上は期待できる比較関数は `cmp` 関数のみ.

このクラスは `TermOrderInterface` を継承する.

Methods

4.29.2.1 format

```
format(self, polynom: polynomial, varname: string='X', reverse:  
bool=False)  
    → string
```

多項式 *polynom* のフォーマットされた文字列を返す.

- *polynom* は一変数多項式でなければならない
- *varname* は変数名の設定ができる.
- *reverse* は *True* と *False* のどちらかになり得る. もしそれが *True* なら, 項は逆 (降) 順で現れる.

4.29.2.2 degree

```
degree(self, polynom: polynomial) → integer
```

多項式 *polynom* の次数を返す.

4.29.2.3 tail_degree

```
tail_degree(self, polynom: polynomial) → integer
```

polynom の全ての項の中での最小次数を返す.

このメソッドは *experimental* です.

4.29.3 MultivarTermOrder – 多変数多項式に対する項順序

Initialize (Constructor)

```
MultivarTermOrder(comparator: function) → MultivarTermOrder
```

このクラスは **TermOrderInterface** を継承する.

Methods

4.29.3.1 format

```
format(self, polynom: polynomial, varname: tuple=None, reverse:
bool=False, **kws: dict)
    → string
```

多項式 `polynom` のフォーマットされた文字列を返す.

追加の引数である `varnames` は変数名が必要とされる.

- `polynom` は多変数多項式です.
- `varnames` は変数名の列.
- `reverse` は `True` と `False` のどちらかになり得る. もしそれが `True`, 項は逆 (降) 順で現れる.

4.29.4 weight_order – 重み付き順序付け

```
weight_order(weight: sequence, tie_breaker: function=None)
    → function
```

`weight` による重み付き順序の比較関数を返す.

w を `weight` をします. 重み付き順序付けは引数 x と y によって定義され, それらは以下を満たす. もし $w \cdot x < w \cdot y$ なら $x < y$ であり, また $w \cdot x == w \cdot y$ かつ `tie breaker` が $x < y$ と出したら $x < y$.

オプション `tie_breaker` は, もし重み付きベクトルのドット積が引数 `tie` と等しいままなら使われるもう一つの比較関数. もしそのオプションが `None` (初期設定) で, 与えられた引数を順序付けするため `tie breaker` が本当に必要なら, `TypeError` が起こる.

Examples

```
>>> w = termorder.MultivarTermOrder(
...     termorder.weight_order((6, 3, 1), cmp))
>>> w.cmp((1, 0, 0), (0, 1, 2))
1
```

4.30 poly.uniutil – 一変数多項式のためのユーティリティ

- **Classes**

- **RingPolynomial**
- **DomainPolynomial**
- **UniqueFactorizationDomainPolynomial**
- **IntegerPolynomial**
- **FieldPolynomial**
- **FinitePrimeFieldPolynomial**
- OrderProvider
- DivisionProvider
- PseudoDivisionProvider
- ContentProvider
- SubresultantGcdProvider
- PrimeCharacteristicFunctionsProvider
- VariableProvider
- RingElementProvider

- **Functions**

- **polynomial**

4.30.1 RingPolynomial – 可換環上の多項式

Initialize (Constructor)

```
RingPolynomial(coefficients: terminit, coeffring: CommutativeRing,  
**keywords: dict)  
    → RingPolynomial object
```

多項式を与えられた係数環 `coeffring` で初期化.

このクラスは **SortedPolynomial**, **OrderProvider** そして **RingElement-Provider** から継承.

`coefficients` の型は **terminit**. `coeffring` は **CommutativeRing** のサブクラスのインスタンス.

Methods

4.30.1.1 getRing

getRing(self) → *Ring*

多項式の所属する *Ring* のサブクラスのオブジェクトを返す.
(このメソッドは *RingElementProvider* 内の定義をオーバーライドする)

4.30.1.2 getCoefficientRing

getCoefficientRing(self) → *Ring*

全ての係数が所属する *Ring* サブクラスのオブジェクトを返す.
(このメソッドは *RingElementProvider* 内の定義をオーバーライドする)

4.30.1.3 shift_degree_to

shift_degree_to(self, degree: *integer*) → *polynomial*

次数が与えられた *degree* である多項式を返す. より正確に, $f(X) = a_0 + \dots + a_n X^n$ とすると, `f.shift_degree_to(m)` は以下を返す:

- もし *f* が零多項式なら, 零多項式を返す
- $a_{n-m} + \dots + a_n X^m$, ($0 \leq m < n$)
- $a_0 X^{m-n} + \dots + a_n X^m$, ($m \geq n$)

(このメソッドは *OrderProvider* から継承される)

4.30.1.4 split_at

split_at(self, degree: *integer*) → *polynomial*

与えられた次数で分割された二つの多項式のタプルを返す. 与えられた次数の項は, もし存在するなら, 下の次数の多項式の側に属する.
(このメソッドは *OrderProvider* から継承される)

4.30.2 DomainPolynomial – 整域上の多項式

Initialize (Constructor)

DomainPolynomial(coefficients: *terminit*, coeffring: *CommutativeRing*, **keywords: *dict*)
→ *DomainPolynomial object*

与えられた整域 `coeffring` に対し多項式を初期化.

基本的な多項式の演算に加え, それは擬除算を持つ.

このクラスは **RingPolynomial** と **PseudoDivisionProvider** を継承.
`coefficients` の型は **termint**. `coeffring` は `coeffring.isdomain()` を満たす
CommutativeRing のサブクラスのインスタンス.

Methods

4.30.2.1 pseudo_divmod

pseudo_divmod(self, other: *polynomial*) → tuple

以下のような多項式 Q, R のタプル (Q, R) を返す:

$$d^{\deg(f)-\deg(\text{other})+1}f = \text{other} \times Q + R,$$

d は other の主係数.

(このメソッドは PseudoDivisionProvider から継承される)

4.30.2.2 pseudo_floordiv

pseudo_floordiv(self, other: *polynomial*) → *polynomial*

以下のような多項式 Q を返す:

$$d^{\deg(f)-\deg(\text{other})+1}f = \text{other} \times Q + R,$$

d は other の主係数.

(このメソッドは PseudoDivisionProvider から継承される)

4.30.2.3 pseudo_mod

pseudo_mod(self, other: *polynomial*) → *polynomial*

以下のような多項式 R を返す:

$$d^{\deg(f)-\deg(\text{other})+1}f = \text{other} \times Q + R,$$

d は other の主係数.

(このメソッドは PseudoDivisionProvider から継承される)

4.30.2.4 exact_division

exact_division(self, other: *polynomial*) → *polynomial*

(割り切れるとき) 除算の商を返す.

(このメソッドは PseudoDivisionProvider から継承される)

4.30.2.5 scalar_exact_division

**scalar_exact_division(self, scale: *CommutativeRingElement*)
→ *polynomial***

各係数を割り切る scale による商を返す.

(このメソッドは PseudoDivisionProvider から継承される)

4.30.2.6 discriminant

`discriminant(self) → CommutativeRingElement`

多項式の判別式を返す.

4.30.2.7 to_field_polynomial

`to_field_polynomial(self) → FieldPolynomial`

整域 D 上の多項式環を D の商体へ埋め込むことにより得られる `FieldPolynomial` オブジェクトを返す.

4.30.3 UniqueFactorizationDomainPolynomial – UFD 上の多項式

Initialize (Constructor)

`UniqueFactorizationDomainPolynomial(coefficients: terminit,
coeffring: CommutativeRing, **keywords: dict)
→ UniqueFactorizationDomainPolynomial object`

与えられた UFD `coeffring` において多項式を初期化.

このクラスは `DomainPolynomial`, `SubresultantGcdProvider` そして `ContentProvider` から継承する.

`coefficients` の型は `terminit`. `coeffring` は `coeffring.isufd()` を満たす `CommutativeRing` のサブクラスのインスタンス.

4.30.3.1 content

`content(self) → CommutativeRingElement`

多項式の内容を返す.

(このメソッドは `ContentProvider` から継承される)

4.30.3.2 primitive_part

`primitive_part(self) → UniqueFactorizationDomainPolynomial`

多項式の原始的部分を返す.

(このメソッドは `ContentProvider` から継承される)

4.30.3.3 subresultant_gcd

subresultant_gcd(self, other: *polynomial*) → *UniqueFactorizationDomainPolynomial*

与えられた多項式の最大公約数を返す。これらは多項式環に入っていない
ならず、その係数環は UFD でなければならない。

(このメソッドは SubresultantGcdProvider から継承される)

Reference: [13]Algorithm 3.3.1

4.30.3.4 subresultant_extgcd

subresultant_extgcd(self, other: *polynomial*) → *tuple*

$A \times self + B \times other = P$ である (A, B, P) を返す。P は与えられた多項式
の最大公約数。これは多項式環に入っていないならず、その係数環は UFD で
なければならない。

参考: [21]p.18

(このメソッドは SubresultantGcdProvider から継承される)

4.30.3.5 resultant

resultant(self, other: *polynomial*) → *polynomial*

self と other の終結式を返す。

(このメソッドは SubresultantGcdProvider から継承される)

4.30.4 IntegerPolynomial – 有理整数環上の多項式

Initialize (Constructor)

IntegerPolynomial(coefficients: *terminit*, coeffring: *CommutativeRing*, **keywords: *dict*)
→ *IntegerPolynomial object*

与えられた可換環 coeffring において多項式を初期化。

組み込みの int/long へ特別な初期化がされなければならないので、このクラス
は必要とされる。

このクラスは **UniqueFactorizationDomainPolynomial** から継承。

coefficients の型は **terminit**。coeffring は **IntegerRing** のインスタンス。冗
長のように思えるが、有理整数環を与える必要がある。

4.30.5 FieldPolynomial – 体上の多項式

Initialize (Constructor)

```
FieldPolynomial(coefficients: terminit, coeffring: Field, **keywords:
dict)
    → FieldPolynomial object
```

与えられた体 `coeffring` において多項式を初期化.

体上の多項式環はユークリッド整域なので, 除算が提供される.

このクラスは **RingPolynomial**, **DivisionProvider** そして **ContentProvider** から継承.

`coefficients` の型は **terminit**. `coeffring` は **Field** のサブクラスのインスタンス.

Operations

operator	explanation
<code>f // g</code>	切り捨て除算の商
<code>f % g</code>	余り
<code>divmod(f, g)</code>	商と余り
<code>f / g</code>	有利関数体上での除算

Methods

4.30.5.1 content

content(self) → *FieldElement*

多項式の内容を返す.
(このメソッドは `ContentProvider` から継承される)

4.30.5.2 primitive_part

primitive_part(self) → *polynomial*

多項式の原始的部分を返す.
(このメソッドは `ContentProvider` から継承される)

4.30.5.3 mod

mod(self, dividend: *polynomial*) → *polynomial*

dividend mod *self* を返す.
(このメソッドは `DivisionProvider` から継承される)

4.30.5.4 scalar_exact_division

scalar_exact_division(self, scale: *FieldElement*) → *polynomial*

各係数を割り切る *scale* による商を返す.
(このメソッドは `DivisionProvider` から継承される)

4.30.5.5 gcd

gcd(self, other: *polynomial*) → *polynomial*

self と *other* の最大公約数を返す.

返される多項式はすでにモニック多項式です.
(このメソッドは `DivisionProvider` から継承される)

4.30.5.6 extgcd

extgcd(self, other: *polynomial*) → *tuple*

タプル (*u*, *v*, *d*) を返す; 二つの多項式 *self* と *other* の最大公約数 *d* と以下となる *u*, *v* である

$$d = self \times u + other \times v$$

extgcd を参照.

(このメソッドは `DivisionProvider` から継承される)

4.30.6 `FinitePrimeFieldPolynomial` – 有限素体上の多項式

Initialize (Constructor)

```
FinitePrimeFieldPolynomial(coefficients: terminit, coeffring:
    FinitePrimeField, **keywords: dict)
    → FinitePrimeFieldPolynomial object
```

与えられた可換環 `coeffring` において多項式を初期化.

このクラスは **`FieldPolynomial`** と **`PrimeCharacteristicFunctionsProvider`** から継承する.

`coefficients` の型は **`terminit`**. `coeffring` は **`FinitePrimeField`** のサブクラスのインスタンス.

Methods

4.30.6.1 mod_pow – モジュロとべき乗

`mod_pow(self, polynom: polynomial, index: integer)`
→ *polynomial*

$\text{polynom}^{\text{index}} \bmod \text{self}$ を返す.

`self` を法としていることに注意.
(このメソッドは PrimeCharacteristicFunctionsProvider から継承される)

4.30.6.2 pthroot

`pthroot(self)` → *polynomial*

X^p を X に渡すことにより得られる多項式を返す. p は標数. もし多項式が p 乗された項のみ成さなければ, 結果は無意味.
(このメソッドは PrimeCharacteristicFunctionsProvider から継承される)

4.30.6.3 squarefree_decomposition

`squarefree_decomposition(self)` → *dict*

平方因子を含まない多項式分解を返す.

返される値は, `keys` が整数で `values` が対応したべき乗因子の辞書. 例えば, もし

Examples

```
>>> A = A1 * A2**2
>>> A.squarefree_decomposition()
{1: A1, 2: A2}.
```

(このメソッドは PrimeCharacteristicFunctionsProvider から継承される)

4.30.6.4 distinct_degree_decomposition

`distinct_degree_decomposition(self)` → *dict*

多項式を相異なる次数で因数分解したものを返す.

返される値は `keys` が整数で `values` が対応した次数の因数の積である辞書. 例えば, もし $A = A1 \times A2$, で, そして $A1$ の全ての既約因子が次数 1 を持ち, $A2$ の既約因子は次数 2 を持つ, そして結果は: {1: $A1$, 2: $A2$ }.

与えられた多項式は平方因子をもないものでなければならず、その係数環は有限体でなければならない。

(このメソッドは `PrimeCharacteristicFunctionsProvider` から継承される)

4.30.6.5 `split_same_degrees`

```
split_same_degrees(self, degree: ) → list
```

多項式の既約因子を返す。

多項式は与えられた次数の既約因子の積でなければならない。

(このメソッドは `PrimeCharacteristicFunctionsProvider` から継承される)

4.30.6.6 `factor`

```
factor(self) → list
```

多項式を因数分解する。

返される値は、最初の成分は因数で次の成分はその重複度であるタブルのリストです。

(このメソッドは `PrimeCharacteristicFunctionsProvider` から継承される)

4.30.6.7 `isirreducible`

```
isirreducible(self) → bool
```

もし多項式が既約なら `True` を返し、さもなければ `False` を返す。

(このメソッドは `PrimeCharacteristicFunctionsProvider` から継承される)

4.30.7 `polynomial` – さまざまな多項式に対するファクトリ関数

```
polynomial(coefficients: terminit, coeffring: CommutativeRing)  
→ polynomial
```

多項式を返す。

† 関数を呼ぶ前に以下を設定することにより、係数環から多項式の型を選ぶ方法をオーバーライドすることができる:

```
special_ring_table[coeffring_type] = polynomial_type
```

.

4.31 poly.univar – 一変数多項式

- Classes
 - †**PolynomialInterface**
 - †**BasicPolynomial**
 - **SortedPolynomial**

この `poly.univar` は以下の型を使っている:

polynomial :

`polynomial` はこの文脈では **PolynomialInterface** のサブクラスのインスタンス.

4.31.1 PolynomialInterface – 全ての一変数多項式に対する基底クラス

Initialize (Constructor)

抽象クラスなのでインスタンスは作らない.
このクラスは **FormalSumContainerInterface** から派生される.

Operations

operator	explanation
f * g	乗法 ¹
f ** i	べき乗

Methods

4.31.1.1 differentiate – 形式微分

`differentiate(self) → polynomial`

多項式の形式微分を返す.

4.31.1.2 downshift_degree – 多項式の次数を下げる

`downshift_degree(self, slide: integer) → polynomial`

次数 `slide` を持つ全ての項を下にシフトして得られた多項式を返す.

最も次数が小さい項が `slide` より小さいとき, 結果は数学的には多項式でないことに注意. このような場合でも, このメソッドは例外は起こさない.

†`f.downshift_degree(slide)` は `f.upshift_degree(-slide)` と同等のものであります.

4.31.1.3 upshift_degree – 多項式の次数を上げる

`upshift_degree(self, slide: integer) → polynomial`

次数 `slide` を持つ全ての項を上シフトして得られた多項式を返す.

†`f.upshift_degree(slide)` は `f.term_mul((slide, 1))` と同等のものである.

4.31.1.4 ring_mul – 環上の乗法

`ring_mul(self, other: polynomial) → polynomial`

多項式 `other` との乗法の結果を返す.

4.31.1.5 scalar_mul – スカラーの乗法

`scalar_mul(self, scale: scalar) → polynomial`

スカラー `scale` による乗法の結果を返す.

4.31.1.6 term_mul – 項の乗法

`term_mul(self, term: term) → polynomial`

与えられた `term` の乗法の結果を返す. `term` はタプル `(degree, coeff)` として与えられるか, `polynomial` として与えられる.

4.31.1.7 square – 自身との乗法

```
square(self) → polynomial
```

この多項式の平方を返す.

4.31.2 BasicPolynomial – 多項式の基本的実装

基本的な多項式の型. 変数名や環のような概念はない.

Initialize (Constructor)

```
BasicPolynomial(coefficients: terminit, **keywords: dict)  
→ BasicPolynomial
```

このクラスは **PolynomialInterface** を継承し実装.
coefficients の型は **terminit**.

4.31.3 SortedPolynomial – 項がソートされたままの状態に維持する多項式

Initialize (Constructor)

```
SortedPolynomial(coefficients: terminit, _sorted: bool=False,  
**keywords: dict)  
→ SortedPolynomial
```

このクラスは **PolynomialInterface** から派生される.
coefficients の型は **terminit**. 任意的に もし係数がすでにソートされた項のリストなら, _sorted は True になり得る.

Methods

4.31.3.1 degree – 次数

`degree(self) → integer`

この多項式の次数を返す。もし零多項式なら、次数は -1 となる。

4.31.3.2 leading_coefficient – 主係数

`leading_coefficient(self) → object`

最も次数が高い項の係数を返す。

4.31.3.3 leading_term – 主項

`leading_term(self) → tuple`

タプル (degree, coefficient) として主項を返す。

4.31.3.4 †ring_mul_karatsuba – Karatsuba 法による乗算

`ring_mul_karatsuba(self, other: polynomial) → polynomial`

同じ環上での二つの多項式の乗法。計算は Karatsuba 法によって実行される。これはだいたい次数が 100 以上のとき早く動くだろう。初期設定ではこの方法を用いていないので、これを使う必要があるなら自身で用いる。

Bibliography

- [1] IPython. <http://ipython.scipy.org/>.
- [2] KANT/KASH. <http://www.math.tu-berlin.de/~kant/kash.html>.
- [3] Magma. <http://magma.maths.usyd.edu.au/magma/>.
- [4] Maple. <http://www.maplesoft.com/>.
- [5] Mathematica. <http://www.wolfram.com/products/mathematica/>.
- [6] matplotlib. <http://matplotlib.sourceforge.net/>.
- [7] mpmath. <http://code.google.com/p/mpmath/>.
- [8] NZMATH. <http://tnt.math.se.tmu.ac.jp/nzmth/>.
- [9] PARI/GP. <http://pari.math.u-bordeaux.fr/>.
- [10] SIMATH. <http://tnt.math.se.tmu.ac.jp/simath/>.
- [11] Janice S. Asuncion. Integer factorization using different parameterizations of Montgomery's curves. Master's thesis, Tokyo Metropolitan University, 2006.
- [12] J. Brillhart and J. L. Selfridge. Some factorizations of $2^n \pm 1$ and related results. *Math. Comp.*, Vol. 21, pp. 87–96, 1967.
- [13] Henri Cohen. *A Course in Computational Algebraic Number Theory*. GTM138. Springer, 1st. edition, 1993.
- [14] G. E. Collins and M. J. Encarnación. Improved techniques for factoring univariate polynomials. *Journal of Symbolic Computation*, Vol. 21, pp. 313–327, 1996.
- [15] Richard Crandall and Carl Pomerance. *Prime Numbers*. Springer, 1st. edition, 2001.
- [16] Ivan Bjerre Damgård and Gudmund Skovbjerg Frandsen. Efficient algorithms for the gcd and cubic residuosity in the ring of Eisenstein integers. *Journal of Symbolic Computation*, Vol. 39, No. 6, pp. 643–652, 2005.

- [17] D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bull. Amer. Math. Soc.*, Vol. 33, pp. 327–340, 1927.
- [18] H. W. Lenstra, Jr. Miller's primality test. *Information processing letters*, Vol. 8, No. 2, 1979.
- [19] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. DISCRETE MATHEMATICS AND ITS APPLICATIONS. CRC Press, 1st. edition, 2003.
- [20] André Weiler. $(1 + i)$ -ary gcd computation in $\mathbb{Z}[i]$ as an analogue to the binary gcd algorithm. *Journal of Symbolic Computation*, Vol. 30, No. 5, pp. 605–617, 2000.
- [21] Kida Yuuji. Integral basis and decomposition of primes in algebraic fields (Japanese). <http://www.rkmath.rikkyo.ac.jp/~kida/intbasis.pdf>.