1 HYPERGEOMETRIC MOTIVES

Chapter 1
HYPERGEOMETRIC MOTIVES

1.1 Introduction

Let $\vec{\alpha}, \vec{\beta} \in \mathbf{C}^n$ be n-tuples (or multisets) of complex numbers. For arithmetic applications we will eventually take them to be rationals, and for purposes of monodromy will largely need only to consider them modulo 1.

Consider the generalised hypergeometric differential equation

$$z(\theta + \alpha_1) \cdots (\theta + \alpha_n)F(z) = (\theta + \beta_1 - 1) \cdots (\theta + \beta_n - 1)F(z), \qquad \theta = z\frac{d}{dz},$$

whose only singularities are regular at 0, 1, and ∞. For simplicity of exposition, we assume that the β's are distinct modulo 1, when a basis of solutions around $z = 0$ is given by

$$z^{1-\beta_i} {}_nF_{n-1}\left(\begin{matrix} \alpha_1 - \beta_i + 1, \dots, \alpha_n - \beta_i + 1 \\ \beta_1 - \beta_i + 1, ..\vee.., \beta_n - \beta_i + 1 \end{matrix} \,\middle|\, z \right)$$

for $i = 1 \dots n$, and the ith term $\beta_i - \beta_i + 1$ is suppressed. The generalised hypergeometric function ${}_nF_{n-1}$ is given by

$${}_nF_{n-1}\left(\begin{matrix} a_1, \dots, a_n \\ b_1, \dots, b_{n-1} \end{matrix} \,\middle|\, z \right) = \sum_{k=0}^{\infty} \frac{(a_1)_k \cdots (a_n)_k}{(b_1)_k \cdots (b_{n-1})_k} \frac{z^k}{k!},$$

where the Pochhammer symbol is given by $(x)_k = (x)(x + 1) \cdots (x + k - 1)$; the $k!$ in the denominator of the above display can thus be thought of as $(1)_k$, which was the suppressed term. Note that shifting all the α and β by some fixed amount keeps the ${}_nF_{n-1}$ expression the same, while only modifying the $z^{1-\beta_i}$ term. Also, switching α and β can be envisaged in terms of the map $z \to 1/z$ that swaps 0 and ∞.

A theorem of Pochhammer says that the above differential equation has $(n - 1)$ independent *holomorphic* solutions around $z = 1$. Let G denote the fundamental group of the thrice punctured Riemann sphere, and $V_{\vec{\alpha},\vec{\beta}}$ the solution space around a base point. We have a monodromy representation $M : G \to GL_n(V_{\vec{\alpha},\vec{\beta}})$. Writing $g_0, g_1, g_\infty \in G$ for loops about 0,1,∞, we find that $M(g_0)$ has eigenvalues $e^{-2\pi i\beta_j}$ and $M(g_\infty)$ has eigenvalues $e^{2\pi i\alpha_j}$, implying that we are mainly concerned with $\vec{\alpha}$ and $\vec{\beta}$ only modulo 1. Indeed, one can note that if we take $F = {}_nF_{n-1}(\vec{a}, \vec{b}|z)$ and $\vec{x} \in \mathbf{Z}^n, \vec{y} \in \mathbf{Z}^{n-1}$, then generically ${}_nF_{n-1}(\vec{a} + \vec{x}, \vec{b} + \vec{y}|z)$ is a linear combination of rational functions times derivatives of F (this is a contiguity relation). Meanwhile, the above fact from Pochhammer implies that $M(g_1)$ must have $(n - 1)$ eigenvalues equal to 1 (all with independent eigenvectors), and so this element is a pseudo-reflection.

It turns out that if $H \subseteq GL_n(\mathbf{C})$ is generated by A and B with AB^{-1} a pseudo-reflection, the H-action on $\mathbf{C}^n$ is irreducible if and only if A and B have disjoint sets of eigenvalues. This is equivalent to all the $\alpha_i - \beta_j$ being nonintegral. Moreover, in his 1961 Amsterdam thesis, Levelt showed that, given any eigenvalues, there are (up to conjugacy) unique A and B realising these eigenvalues with AB^{-1} a pseudo-reflection. (Much of the above comes from notes of Beukers.)

For arithmetic purposes, one usually also desires that the eigenvalues be roots of unity and the sets of them be Galois-invariant. Thus we can specify hypergeometric data H by (say) two products of cyclotomic polynomials, these products being coprime and of equal degree. Given such an H, Rodriguez-Villegas conjectures the existence of a family of pure motives (defined over $\mathbf{Q}$), for which the trace of Frobenius at good primes is given by a hypergeometric sum defined by Katz [Kat90] (see also [Kat96]). For each rational $t \neq 0, 1$, there should be a motive H_t whose L-function satisfies a functional equation of a prescribed type, with the Euler factors at good primes given in terms of Gauss sums (the bad Euler factors are less understood, and depend on deformation theory).

One can also relate such motives to more traditional objects in many cases. For instance, there is one hypergeometric datum in degree 1, which can be specified by $\alpha = [\frac{1}{2}]$ and $\beta = [0]$, these being rationals corresponding to the second and first cyclotomic polynomials respectively. The L-function here corresponds to the quadratic field $\mathbf{Q}(\sqrt{t(t-1)})$. In degree 2 there are 13 such data, of which 3 are of weight 0 (see below) and give Artin representations of number fields, while the other 10 are of weight 1, and yield elliptic curves (explicitly calculated by Cohen). An example in higher degree is $\alpha = [\frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}]$ and $\beta = [0, 0, 0, 0]$, corresponding respectively to the 5th cyclotomic polynomial and the 4th power of the first cyclotomic polynomial, and this is associated to the Calabi-Yau quintic 3-fold given by

$$x_1^5 + x_2^5 + x_3^5 + x_4^5 + x_5^5 = 5t x_1 x_2 x_3 x_4 x_5.$$

The weight w of a hypergeometric motive can be defined in terms of how much the α and β interlace (considered as roots of unity). In particular, if they are completely interlacing, then the weight is 0, and the resulting motive corresponds to an Artin representation. Write $D(x) = \#\{\alpha : \alpha \leq x\} - \#\{\beta : \beta \leq x\}$. Then $w + 1 = \max_x D(x) - \min_x D(x)$, so that the above 3-fold has weight 3 (from the four β's at 0). This weight controls how large the coefficients of the Euler factors will be.

The trace at q of a hypergeometric motive (for the parameter t) is given in terms of Gauss sums g_q over $\mathbf{F}_q$. Associated to hypergeometric data is a `GammaArray`, and one defines $G_q(r) = \prod_v g_q(-rv)^{\gamma_v}$, and also the `MValue` by $M = \prod_v v^{v\gamma_v}$. For primes p with $v_p(Mt) = 0$ the hypergeometric trace is then given by

$$U_q(t) = \frac{1}{1-q}\left(\sum_{r=0}^{q-2} \omega_p(Mt)^r Q_q(r)\right),$$

where ω_p is the Teichmüller character and $Q_q(r) = (-1)^{m_0} q^{D+m_0-m_r} G_q(r)$ where m_r is the multiplicity of $\frac{r}{q-1}$ in the β and D is a scaling parameter that involves the Hodge structure (one expression is $m_0 = w+1-2D$). One uses p-adic Γ-functions to expedite the computation of the above Gauss sums (indeed, the above gives the hypergeometric trace as a p-adic number, which one recognises as an integer via sufficiently high precision). The Euler factor is given by the standard recipe $E_p(T) = \exp\bigl(-\sum_n U_{p^n}(t)T^n/n\bigr)$, and this is a polynomial that satisfies a local functional equation.

1.2 Functionality

1.2.1 Creation Functions

`HypergeometricData(A, B)`

`HypergeometricData(F, G)`

`Print` MonStgElt *Default : "cyclotomic"*

These are two of the principal ways of specifying hypergeometric data. The first takes two sequences A and B (of the same length) of rationals, which must be disjoint upon reduction modulo 1, and each of which must be Galois-invariant when taking the corresponding roots of unity (for instance, if $\frac{1}{6}$ is specified, $\frac{5}{6}$ is also given). The second takes two products F and G of cyclotomic polynomials, these products being coprime and of the same degree. Previous Magma versions could switch A and B in some cases; this is no longer the case.

The default can be specified with the `Print` vararg, the other option currently being `"alpha_beta"`.

There is now also some functionality for non-disjoint A and B, which mainly manifests itself at the L-series level.

`HypergeometricData(G)`

This is a third way to specify hypergeometric data, by giving a sequence of integers G such that $\sum_v vG[v] = 0$. Here we have $P_\alpha(T)/P_\beta(T) = \prod_v (T^v - 1)^{G[v]}$, and the polynomials can be determined via Möbius inversion.

`HypergeometricData(L)`

This is a fourth way to specify hypergeometric data, by giving a *list* L of nonzero integers (with repetition possible) corresponding to the sequence L of the previous intrinsic, with negative integers for those where $L[v]$ is negative. The sum of the members of the list must be 0.

`HypergeometricData(F, G)`

This is a fifth way to specify hypergeometric data, by giving two arrays F and G of integers, corresponding to the cyclotomic polynomials to be used.

`HypergeometricData(E)`

This is a utility intrinsic that take a sequence E of two sequences and then passes these two sequences to the intrinsics above.

`Twist(H)`

This intrinsic takes hypergeometric data H, and adds 1/2 to every element in α and β, returning new hypergeometric data. Magma no longer (ever) switches α and β when twisting.

`PrimitiveData(H)`

Given hypergeometric data H, return its primitive associated data. This is most easily described in terms of `GammaList`, dividing all the elements by the gcd.

`PossibleHypergeometricData(d)`

`Weight`	RngIntElt	*Default :* `false`
`TwistMinimal`	BoolElt	*Default :* `false`
`CyclotomicData`	BoolElt	*Default :* `false`
`Primitive`	RngIntElt	*Default :* 0

Given a degree d, generate all possible examples of hypergeometric data of that degree, returned as a sequence of pairs of sequences, each sequence therein having d rationals. If `Weight` is specified, restrict to data of this weight. If `TwistMinimal` is specified, only give twist-minimal data. If `CyclotomicData` is specified, return the sequences of cyclotomic data rather than rationals. If `Primitive` is `true`, only return data that are primitive; if `Primitive` is a positive integer, return the data that have this imprimitivity.

1.2.2 Access Functions

`Weight(H)`

The weight of the given hypergeometric data H.

`Degree(H)`

The degree of the given hypergeometric data H.

`DefiningPolynomials(H)`

The (products of cyclotomic) polynomials corresponding to α and β corresponding to hypergeometric data H.

`Bezoutian(H)`

The resultant of the defining polynomials of the hypergeometric data.

`CyclotomicData(H)`

Returns two arrays of integers, specifying which cyclotomic polynomials occur for α and β corresponding to hypergeometric data H. Thus, for example, $\Phi_3\Phi_4^2\Phi_6$ would be represented by [3,4,4,6]).

`AlphaBetaData(H)`

Returns two arrays of rationals, giving the α and β of the hypergeometric data H.

`MValue(H)`

This returns the scaling parameter M of the given hypergeometric data H. This is defined by taking $M_n = \prod_{d|n} d^{d\mu(n/d)}$ for the nth cyclotomic polynomial, and combining these into the products for α and β, and then dividing these. Another definition is $M = \prod_v v^{v\gamma_v}$.

`GammaArray(H)`

This returns an array of integers corresponding to γ_v, where these are defined by $P_\alpha(T)/P_\beta(T) = \prod_v (T^v - 1)^{\gamma_v}$. We also have $\sum_v v\gamma_v = 0$.

`GammaList(H)`

This returns a *list* of integers corresponding to γ_v, where $\mathrm{sign}(\gamma_v) \cdot v$ appears in the list $|\gamma_v|$ times.

`H1 eq H2`

`H1 ne H2`

Two instances of hypergeometric data $H1$ and $H2$ are equal if they have the same α and β.

`IsPrimitive(H)`

Returns `true` if the given hypergeometric data H is primitive, and the index of imprimitivity. The latter is the gcd of the elements in the `GammaList`.

1.2.3 Functionality with L-series and Euler Factors

`HypergeometricTrace(H, t, q)`

Given a hypergeometric datum H, a rational $t \neq 0$, and a prime power $q = p^f$ for which p is good or multiplicative, return the hypergeometric trace. The intrinsic also works more generally when $v_p(Mt) = 0$, even if p is wild.

```
HypergeometricTraceK(A, B, t, q)
```



```
HypergeometricTraceK(A, B, t, q)
```


`Precision` RngIntElt *Default* : 5

Given α's and β's associated to a not-necessarily Galois datum, and a rational $t \neq 0$, and a prime power $q = p^f$ for which $v_p(t) = 0$ and p divides no denominator of the α's and β's, return the hypergeometric trace according to the p-adic Γ-function definition, namely that

$$H_q(\alpha, \beta|t) = \frac{q^D}{1-q} \sum_{r=0}^{q-2} \omega_p(t)^r \frac{q^{m_0}}{q^{m_r}} \frac{X_q(r)}{X_q(0)} \frac{(-p)^{T_f(r)}}{(-p)^{T_f(0)}}$$

where ω_p is the Teichmüller as before, m_r is the multiplicity of $-r/(q-1)$ in B (modulo 1), while

$$X_q(r) = \prod_{i=0}^{f-1} \frac{\prod_j \Gamma_p(\{p^i(\alpha_j + r/(q-1))\})}{\prod_j \Gamma_p(\{p^i(\beta_j + r/(q-1))\})}$$

and

$$T_f(r) = \sum_j \left[S_f(\alpha_j + r/(q-1)) - S_f(\beta_j + r/(q-1))\right] \quad \text{where} \quad S_f(x) = \sum_{i=0}^{f-1} \{p^i x\}.$$

In the version where t is a p-adic, it must be compatible with q. The precision of t should exceed that with the `Precision` parameter, those this is not mandated, and could cause an incompatibility problem later in the computation.

This intrinsic is (much) slower than the optimized version in the Galois case.

```
EulerFactor(H, t, p)
```

`Degree` RngIntElt *Default* : 0

`Check` BoolElt *Default* : `false`

`Fake` BoolElt *Default* : `false`

This intrinsic is the heart of the hypergeometric motive package. It takes hypergeometric data H, a rational $t \neq 0, 1$, and a prime p, and computes the pth Euler factor of the hypergeometric motive at t. This uses p-adic Γ-functions, as indicated by Cohen. The `Degree` vararg specifies how many terms in the Euler factor should be computed – if this is 0, then the whole polynomial is computed. The `Check` vararg allows one to turn off the use of the (local) functional equation that is used to expedite the computation process.

The `Fake` vararg allows one to compute the hypergeometric trace(s) for t with $v_p(Mt) = 0$ (including wild primes). Whether or not this is the actual Euler factor depends on how inertia acts. The use of this vararg inhibits the use of the local functional equation, but one can curtail via `Degree`, and apply it manually (if known).

In general, the given prime must not be wild, that is, it must not divide the denominator of any of the α or β.

At other bad primes, the Euler factor will depend upon the relevant monodromy. The primes for which $v_p(t-1) > 0$ can perhaps be called multiplicative, in that p should divide the conductor only once (this is related to the pseudoreflection). Since $v_p(Mt) = 0$ here, the Euler factor (of degree $d-1$) can be recovered by the p-adic Γ-function methods (even when $p = 2$). Also it is often possible to relate the (presumed) hypergeometric motive to objects from a deformation. When $v_p(t-1)$ is even and the weight is also even, the prime p is actually good and has a degree d Euler factor, even though the hypergeometric trace only gives one of degree $(d-1)$. In such a case, the `EulerFactor` intrinsic with the `Fake` vararg will return the part from the hypergeometric trace.

The p with $v_p(1/t) > 0$ correspond to monodromy at ∞. The associated inertia is given by the roots of unity with the β, with maximal Jordan blocks when eigenvalues are repeated. The same is true for p with $v_p(t) > 0$, where the monodromy is around 0 (so that the α are used). In Example H1E8), an instance is given where the inertia is trivialised due to having $\zeta^{v_p(t)} = 1$.

We can compute the Euler factors at such tame primes as follows. Suppose that $t = t_0 p^{vm}$ with $v > 0$, where m occurs as a denominator of the α (similarly with $v < 0$ and β). Then one takes the smallest $q = p^f$ that is 1 mod m, and from the hypergeometric trace formula extracts the terms

$$\omega_p(Mt_0)^{j(q-1)/m} Q_q\left(\frac{j(q-1)}{m}\right)$$

for $0 \le j < m$ with $\gcd(j, m) = 1$. Denoting these by η, we then have that $\prod_\eta (1 - \eta T^f)$ is an fth power (due to repetitions in the above extraction), and the fth root of this is the desired Euler factor of degree $\phi(m)$.

When m does not divide $v_p(t)$, the Euler factor from it is trivial. One then multiplies together all such Euler factors corresponding to the m from the α and β. Each m is only considered once, even if it appears multiple times in the `CyclotomicData`, as the Jordan blocks of the eigenvalues are maximal. Note that the local functional equation is not used for tame primes, though the computation should not be too onerous unless $q = p^f$ is large.

`LSeries(H, t)`

`BadPrimes`	SeqEnum	*Default :* []
`HodgeStructure`	HodgeStruc	*Default :* `false`
`GAMMA`	SeqEnum	*Default :* []
`Identify`	BoolElt	*Default :* `true`
`Precision`	RngIntElt	*Default :* 0

`Weight01`	BoolElt	*Default :* `false`
`QuadraticTwist`	Any	*Default :* `false`
`PoleOrder`	RngIntElt	*Default :* 0
`SaveEuler`	RngIntElt	*Default :* `false`

Given hypergeometric data H and a rational $t \neq 0, 1$, try to construct the L-series of the associated motive. This will usually need the Euler factors at wild primes to be specified. Everything else, including tame/multiplicative Euler factors and γ-factors, can be computed automatically by Magma (these can also be given respectively via `BadPrimes`, and `GAMMA` and/or `HodgeStructure`).

The `Identify` vararg indicates whether an attempt should be made to identify motives of weight 0 as Artin representations, and similarly with (hyper)elliptic curves for weight 1. The `Weight01` vararg when `true`will `Translate` the L-series (essentially a Tate twist) so that the weight is 0 or 1. Setting `Weight01` to an (odd) integer r will `Translate` so that the (motivic) weight is r plus the number of zero entries in the `AlphaBetaData`. A typical choice is $r = -1$.

The `QuadraticTwist` vararg can be used to take the tensor product with the given quadratic Dirichlet character. This can be given as a nonzero rational or as a character, or alternatively can be set to `true`, when Magma will use a default twisting factor. However, this option can conflict with `BadPrimes` (Magma does not know whether to apply such primes to the original L-function or the twist), and so should be used sparingly.

The `PoleOrder` vararg allows the user to specify that the given power of the (shifted) Riemann ζ-function is expected to divide the L-series of the hypergeometric motive. The routines will then act accordingly, decomposing the L-series into a factorisation and moving the poles into the ζ-function part.

Finally, the `SaveEuler` option takes a nonnegative integer (or a boolean), and indicates how large of primes should have their `EulerFactor` saved when computed. This is useful when (say) dealing with such L-function constructs as `Symmetrization`, for which the underlying work with hypergeometric traces is being done on one L-function, and then used multiple times. This option will be (silently) ignored is Magma is able to `Identify` the L-function as coming from somewhere else.

The intrinsic actually returns two L-series, the first corresponding to the disjoint parts of A and B, and the second an Artin part (weight 0) corresponding to the common part. For most purposes the second can be ignored (it will typically be the trivial `LSeries`).

1.2.3.1 Identification of Hypergeometric Data as Other Objects

`ArtinRepresentation(H, t)`

`Check`	BoolElt	*Default :* `true`

Given hypergeometric data H of weight 0 and a rational $t \neq 0, 1$, try to determine the associated Artin representation. This is implemented for all such H of degree 3 or less, and for some of degree 4 and higher. The condition needed is that `GammaList(H)` have cardinality 3. When `Check` is true, good primes up to 100 have their Euler factors checked for correctness.

```
EllipticCurve(H)
```

```
EllipticCurve(H, t)
```

Given hypergeometric data H of degree 2 and weight 1 (there are 10 such families) and a rational $t \neq 0, 1$, return the associated elliptic curve, as catalogued by Cohen. When t is not given, return the result over a function field.

For each of the 10 families, the same function can be called for the corresponding imprimitive data of index r, and the result will generically be an elliptic curve over an extension of degree r. However, when the $x^r - 1/Mt$ splits, the intrinsic will return an array of elliptic curves corresponding to this splitting.

```
HyperellipticCurve(H)
```

```
HyperellipticCurve(H, t)
```

Given hypergeometric data H of degree 4 and weight 1 and a rational $t \neq 0, 1$, return the associated hyperelliptic curve, if this data is known to correspond to such. When t is not given, return the result over a function field. There are 18 cases where one gets a genus 2 curve from the `CanonicalCurve` (making 36 cases when twisting is considered), and a few others where `CanonicalCurve` gives a higher genus curve and there is a genus 2 quotient. In general, one can try to call `IsHyperelliptic` on the `CanonicalCurve`.

```
Identify(H, t)
```

Given hypergeometric data H and a rational $t \neq 0, 1$, return any known associated object (else returns `false`). The return value can (currently) be: an Artin representation (weight 0); an elliptic curve over $\mathbf{Q}$ (weight 1 in degree 2); an elliptic curve over a number field (weight 1 in degree $2r$ with imprimitivity r), or possibly multiple such curves; or a hyperelliptic curve over $\mathbf{Q}$ (weight 1 in degree 4).

1.2.4 Associated Schemes and Curves

```
CanonicalScheme(H)
```

```
CanonicalScheme(H, t)
```

Given hypergeometric data H, this constructs a canonical associated scheme. When the parameter t is given, the specialization is returned, otherwise the result returned will be a scheme over a function field.

The scheme is determined from the `GammaList`, with a variable (X_i or Y_j) for every element in the list. The scheme is the intersection of $\sum_i X_i = \sum_j Y_j = 1$ with

$$\prod_i X_i^{g_i^+} \prod_j Y_j^{g_j^-} = \frac{1}{Mt},$$

where the g_i^+ are the positive elements in the GammaList and the g_j^- are the negative ones (one usually moves the latter to the other side of the equation, to make the exponents positive).

`CanonicalCurve(H)`

`CanonicalCurve(H, t)`

Given suitable hypergeometric data H, this tries to construct an associated plane curve. When the parameter t is given, the specialization at t is returned, otherwise the return value will be a plane curve over a function field. The curve is constructed using the GammaList (which indicates the Jacobi sums that need to be taken). When this list has four elements, it is always possible to get a curve. When the list has six elements, it is sometimes possible, depending on whether the largest element (in absolute value) is the negation of the sum of two of the other elements. If it is not possible to construct such a curve, the intrinsic returns false.

1.2.5 Utility Functions

`HypergeometricMotiveSaveLimit(n)`

`HypergeometricMotiveClearTable()`

These are utility intrinsics that will cache the pre-computation of p-adic Γ-functions. The first indicates to save all computed values when the prime power is less than n, and the second clears the table. The qth table entry will have $(q-1)$ elements in it.

`pPart(H, p)`

`pParts(H)`

Given a hypergeometric datum, reduce the cyclotomic indices modulo either the given prime or all wild primes.

1.3 Examples

Example H1E1

Our first example constructs some hypergeometric motives, and recognises them as being related to elliptic curves or Artin representations.

```
> H := HypergeometricData([1/2],[0]); // weight 0
> t := 3/5;
> A := ArtinRepresentation(H,t);
> D := Discriminant(Integers(Field(A))); // -24
> assert IsSquare(D/(t*(t-1))); // Q(sqrt(t(t-1)))
> R := ArtinRepresentationQuadratic(-24);
> assert A eq R;
> //
> H := HypergeometricData([1/4,3/4],[0,0]);
> Weight(H);
```

```
1
> DefiningPolynomials(H);
y^2 + 1, y^2 - 2*y + 1
> t := 3/2;
> E := EllipticCurve(H,t); E;
Elliptic Curve defined by y^2 + x*y = x^3 + 1/96*x over Q
> P := PrimesInInterval(5,100);
> &and[EulerFactor(E,p) eq EulerFactor(H,t,p) : p in P];
true
> //
> f := CyclotomicPolynomial(6)*CyclotomicPolynomial(2);
> g := CyclotomicPolynomial(1)^3;
> H := HypergeometricData(f,g); H;
Hypergeometric data given by [ 2, 6 ] and [ 1, 1, 1 ]
> Weight(H);
2
> GammaList(H);
[* -1, -1, -1, -3, 6 *]
> GammaArray(H);
[ -3, 0, -1, 0, 0, 1 ]
> [EulerFactor(H,4,p) : p in [5,7,11,13,17,19]];
[ 125*y^3 + 20*y^2 + 4*y + 1, 343*y^3 - 42*y^2 - 6*y + 1,
  -1331*y^3 - 22*y^2 + 2*y + 1, -2197*y^3 - 156*y^2 + 12*y + 1,
  4913*y^3 + 323*y^2 + 19*y + 1, 6859*y^3 - 57*y^2 - 3*y + 1 ]
> //
> _<u> := FunctionField(Rationals());
> H := HypergeometricData([-2,0,0,-1,0,1] : Print:="alpha_beta");
> H; // weight 1
Hypergeometric data given by [1/6,1/3,2/3,5/6] and [0,0,1/4,3/4]
> HyperellipticCurve(H); // defined over Q(u)
Hyperelliptic Curve defined by y^2 = 4*x^6 - 8*x^5 + 4*x^4 - 64/729/u
> t := 4;
> C := Specialization($1,t); // only works over Q(u)
> &and[EulerFactor(C,p) eq EulerFactor(H,t,p) : p in P];
true
> //
> H := HypergeometricData([0,-1,0,1,0,1,0,-1] : Print:="alpha_beta");
> H; // weight 1
Hypergeometric data given by [1/6,1/3,2/3,5/6] and [1/8,3/8,5/8,7/8]
> MValue(H);
729/4096
> t := 3; // could alternatively specialize later
> E := EllipticCurve(H,t); aInvariants(E);
[ 0, 0, -s, -s, 0] where s^2 is 4096/2187
> &and[EulerFactor(E,p) eq EulerFactor(H,t,p) : p in P];
true
```

Example H1E2

This is a simple example of twisting hypergeometric data, showing that a related Artin motive is obtained for the given weight 0 data.

```
> f := CyclotomicPolynomial(6);
> g := CyclotomicPolynomial(1)*CyclotomicPolynomial(2);
> H := HypergeometricData(f,g); H; assert(Weight(H)) eq 0;
Hypergeometric data given by [ 6 ] and [ 1, 2 ]
> A := ArtinRepresentation(H,-4/5);
> K := OptimisedRepresentation(Field(A));
> DefiningPolynomial(K);
y^6 - 3*y^5 + 3*y^4 - y^3 + 3*y^2 - 3*y + 1
> T := Twist(H); T;
Hypergeometric data given by [ 3 ] and [ 1, 2 ]
> A := ArtinRepresentation(T,-4/5);
> L := OptimisedRepresentation(Field(A));
> IsSubfield(L,K), DefiningPolynomial(L);
true Mapping from: L to K,  y^3 + 3*y - 1
```

The same can be said for twisting for (hyper)elliptic curves.

```
> H := HypergeometricData([2,2],[3]); // Phi_2^2 and Phi_3
> E := EllipticCurve(H,3);
> T := EllipticCurve(Twist(H),3);
> IsQuadraticTwist(E,T);
true -4/9
> //
> H := HypergeometricData([5],[8]); // Phi_5 and Phi_8
> C := HyperellipticCurve(H);
> t := 7;
> S := Specialization(C,t);
> T := HyperellipticCurve(Twist(H),t);
> Q := QuadraticTwist(T,5*t); // get right parameter
> assert IsIsomorphic(Q,S);
```

Example H1E3

This example exercises the primitivity functionality.

```
> H := HypergeometricData([3],[4]); // Phi_3 and Phi_4
> GammaList(H);
[* -1, 2, 3, -4 *]
> H2 := HypergeometricData([* -2, 4, 6, -8 *]);
> IsPrimitive(H2);
false 2
> PrimitiveData(H2) eq H;
true
> H3 := HypergeometricData([* -3, 6, 9, -12 *]);
> IsPrimitive(H3);
```

```
false 3
> PrimitiveData(H3) eq H;
true
> aInvariants(EllipticCurve(H));
[ 0, 0, -64/27/u, -64/27/u, 0 ] where u is FunctionField(Q).1
> aInvariants(EllipticCurve(H2));
[ 0, 0, -s, -s, 0 ] where s^2=(-64/27)^2/u
> aInvariants(EllipticCurve(H3));
[ 0, 0, -s, -s, 0 ] where s^3=(-64/27)^3/u
```

Example H1E4

Here is an example with the canonical schemes and curves associated to various hypergeometric data.

```
> _<u> := FunctionField(Rationals());
> H := HypergeometricData([* -2, 3, 4, -5 *]); // degree 4
> C := CanonicalScheme(H);
> _<[X]> := Ambient(C); C;
Scheme over Univariate rational function field over Q defined by
X[1] + X[2] - 1, X[3] + X[4] - 1,
X[1]^2*X[2]^5 - 3125/1728/u*X[3]^3*X[4]^4
> Dimension(C), Genus(Curve(C)); // genus 2 curve
1 2
> assert IsHyperelliptic(Curve(C));
> CC := CanonicalCurve(H);
> _<x,y> := Ambient(CC); CC;
Curve over Univariate rational function field over Q defined by
x^7 - 2*x^6 + x^5 + 3125/1728/u*y^7 - 3125/576/u*y^6 +
 3125/576/u*y^5 - 3125/1728/u*y^4
> b, C2 := IsHyperelliptic(CC); assert b;
> HyperellipticCurve(H); // in the degree 4 catalogue
Hyperelliptic Curve defined over Univariate function field
 over Q by y^2 = 4*x^5 - 3125/432/u*x^3 + 9765625/2985984/u^2
> assert IsIsomorphic(HyperellipticCurve(H),C2);
> // and an example where the curve is reducible
> H := HypergeometricData([* 6,6,-8,-4 *]); // weight 1
> C := CanonicalCurve(H);
> A := AlgorithmicFunctionField(FunctionField(C));
> E<s> := ExactConstantField(A);
> CE := BaseChange(C,E);
> I := IrreducibleComponents(CE); assert #I eq 2;
> _<x,y> := Ambient(I[1]); I[1];
Scheme over E defined by [ where s^2 = 1048576/531441/u ]
 x^6 - 2*x^5 + x^4 - s*y^6 + 3*s*y^5 - 3*s*y^4 + s*y^3
> b, C2 := IsHyperelliptic(Curve(I[1])); assert b;
```

Example H1E5

Here is an example in degree 4 and weight 3. It turns out that the motive from $t = -1$ has complex multiplication, and the L-series appears to be the same as that of a Siegel modular form given by [vGvS93, §8.7]. (This was found by Cohen and Rodriguez-Villegas). This L-series also appears in Example **????** .

```
> H := HypergeometricData([1/2,1/2,1/2,1/2],[0,0,0,0]);
> L := LSeries(H,-1 : BadPrimes:=[<2,9,1>]); // guessed
> CFENew(L);
-5.91645678915758854058796423962E-31
> LGetCoefficients(L,100);
[* 1, 0, 0, 0, -4, 0, 0, 0, -6, 0, 0, 0, -84, 0, 0, 0, 36, 0, 0, 0, 0,
   0, 0, 0, 146, 0, 0, 0, 140, 0, 0, 0, 0, 0, 0, 0, 60, 0, 0, 0, -140,
   0, 0, 0, 24, 0, 0, 0, -238, 0, 0, 0, 924, 0, 0, 0, 0, 0, 0, 0, -820,
   0, 0, 0, 336, 0, 0, 0, 0, 0, 0, 0, -396, 0, 0, 0, 0, 0, 0, 0, -693,
   0, 0, 0, -144, 0, 0, 0, -300, 0, 0, 0, 0, 0, 0, 0, -252, 0, 0, 0 *]
> // compare to the Tensor product way of getting this example
> E := EllipticCurve("32a");
> NF := Newforms(ModularForms(DirichletGroup(32).1,3)); // wt 3 w/char
> L1 := LSeries(E); L2 := LSeries(ComplexEmbeddings(NF[1][1])[1][1]);
> TP := TensorProduct(L1, L2, [ <2, 9> ]); // conductor 2^9 (guessed)
> [Round(Real(x)) : x in LGetCoefficients(TP,100)];
[ 1, 0, 0, 0, -4, 0, 0, 0, -6, 0, 0, 0, -84, 0, 0, 0, 36, 0, 0, 0, 0,
  0, 0, 0, 146, 0, 0, 0, 140, 0, 0, 0, 0, 0, 0, 0, 60, 0, 0, 0, -140,
  0, 0, 0, 24, 0, 0, 0, -238, 0, 0, 0, 924, 0, 0, 0, 0, 0, 0, 0, -820,
  0, 0, 0, 336, 0, 0, 0, 0, 0, 0, 0, -396, 0, 0, 0, 0, 0, 0, 0, -693,
  0, 0, 0, -144, 0, 0, 0, -300, 0, 0, 0, 0, 0, 0, 0, -252, 0, 0, 0 ]
```

Example H1E6

We go through an example working with hypergeometric traces when the underlying datum is not Galois. First we check that the relevant intrinsic `HypergeometricTraceK` gives the same answers as `HypergeometricTrace`.

```
> PHD := PossibleHypergeometricData(4);
> t := 2; p := 13;
> Z := Integers();
> for h in PHD do H:=HypergeometricData(h); A:=H`alpha; B:=H`beta;
>    assert HypergeometricTrace(H,t,p) eq Z!HypergeometricTraceK(A,B,t,p);
>    end for;
```

Next we consider the "Klein quartic" datum defined over $\mathbf{Q}(\sqrt{-7})$. For split primes, its trace will be a quadratic surd.

```
> A := [1/14,9/14,11/14];
> B := [1/4,3/4,1];
> p := 29; // a split prime in Q(sqrt(-7))
> e := HypergeometricTraceK(A,B,2,p : Precision:=10);
> PowerRelation(e,4);
```

```
x^2 - x + 2
```

In more generality, one can expect elements in cyclotomic fields.

```
> e := HypergeometricTraceK([1/8,7/8],[0,1/4],2,17 : Precision:=10);
> PowerRelation(e,4);
17*x^4 + 8*x^3 + 24*x^2 + 32*x + 16
> assert IsIsomorphic(NumberField($1),CyclotomicField(8));
> //
> e := HypergeometricTraceK([1/8,7/8],[0,1/4],2,97 : Precision:=10);
> PowerRelation(e,4);
97*x^4 + 56*x^3 - 232*x^2 + 224*x + 784
> assert IsIsomorphic(NumberField($1),CyclotomicField(8));
> // and a non-split prime, with q = 1 mod 8
> e := HypergeometricTraceK([1/8,7/8],[0,1/4],2,7^2 : Precision:=20);
> PowerRelation(e,2); // generates Q(sqrt(2))
7*x^2 - 4*x - 4
```

The K-trace property can also be used to compute at t-values which are not rational. For instance, work of Guillera [Gui12, AG12] implies that $t = 8/(15\sqrt{5}-33)^2$ should give a pole for $(\Phi_2^3\Phi_3, \Phi_1^5)$, and we can verify numerically that the (normalized) traces have average 1 for small primes (and also that the second moment is approximately 2, indicating a splitting).

```
> H := HypergeometricData([2,2,2,3],[1,1,1,1,1]); // weight 4
> A := H`alpha; B := H`beta;
> Z := Integers();
> f := func<p|(1/((15*Sqrt(pAdicField(p,20)!5)-33)/2)^3)>;
> P := [p : p in PrimesUpTo(200) | p^2 mod 5 eq 1];
> R := [<p,Z!HypergeometricTraceK(A,B,Z!f(p),p)> : p in P];
> &+[r[2]/r[1]^2*1.0 : r in R]/#P;
1.13162808398732021466387060353
> &+[r[2]^2/r[1]^4*1.0 : r in R]/#P; // second moment
2.32631764414942310383344594579
```

Comparatively, most t-values have mean zero and second moment of 1, though again by [Gui12] we expect that $t = (4/3)^3$ is exceptional.

```
> MomentData(LSeries(H,5),P,2); // t = 5
0.14376 [ 0.84480 ]
> MomentData(LSeries(H,11),P,2); // t = 11
0.17961 [ 0.68179 ]
> MomentData(LSeries(H,(4/3)^3),P,2); // t = (4/3)^3
0.81211 [ 1.2554 ] // P is not that robust
> MomentData(LSeries(H,(4/3)^3),PrimesUpTo(1000),2);
0.98494 [ 1.9196 ]
```

Finally, we demonstrate how to work with p-adic inputs for the t-parameter.

```
> A := [1/4]; B := [1/3];
> Qp := pAdicField(5,20);
> E := ext<Qp|Polynomial([-2,0,1])>; // x^2-2, unramified
```

```
> HypergeometricTraceK(A,B,4+E.1,5);
1190*E.1 + 1258 + O(5^5)
> HypergeometricTraceK(A,B,4-E.1,5);
-1190*E.1 + 1258 + O(5^5)
```

Example H1E7

Here is an example showing how to handle bad primes in some cases. The Euler factors at $\{3, 5, 17\}$ [where $p|(t-1)$] were determined via a recipe from deformation theory by Rodriguez-Villegas, while at $p = 2$, Roberts suggested a t-value that would trivialise the conductor (from a number field analogy), and Tornaría then computed the full degree 4 factor (at $p = 2$) for $t = 2^8$.

```
> H := HypergeometricData([1/2,1/2,1/2,1/2],[0,0,0,0]);
> Lf := LSeries(Newforms(ModularForms(8,4))[1][1]);
> T := PolynomialRing(Integers()).1; // dummy variable
> f3 := EulerFactor(Lf,3 : Integral)*(1-3*T); // make it a poly
> f5 := EulerFactor(Lf,5 : Integral)*(1-5*T); // via Integral
> f17 := EulerFactor(Lf,17 : Integral)*(1-17*T);
> f2 := 1+T+6*T^2+8*T^3+64*T^4; // determined by Tornaria
> BP := [<2,0,f2>,<3,1,f3>,<5,1,f5>,<17,1,f17>];
> L := LSeries(H,256 : BadPrimes:=BP);
> Conductor(L);
255
> assert Abs(CFENew(L)) lt 10^(-28);
```

One need not specify all the bad prime information as in the above example. Here is a variation on it, with $t = 1/2^8$ (note that this actually gives the same L-series, as the data is a self-twist, with the character induced by twisting being trivial for this choice of t). Note that only the information at 2 is given to LSeries.

```
> H := HypergeometricData([1/2,1/2,1/2,1/2],[0,0,0,0]);
> MValue(H);
256
> t := 1/2^8; // makes v_2(Mt)=0
> f2 := EulerFactor(H,t,2 : Fake);
> f2;
64*T^4 + 8*T^3 + 6*T^2 + T + 1
> L := LSeries(H,t : BadPrimes:=[<2,0,f2>]);
> Conductor(L);
255
> assert Abs(CFENew(L)) lt 10^(-28);
```

Example H1E8

Here is an example with the quintic 3-fold. The deformation theory at $p = 11$ here is related to the Grössencharacter example over $\mathbf{Q}(\zeta_5)$ given in Example **????** . The action on inertia at $p = 11$ involves ζ_5 when $11|t$, and here it is raised to the 5th power, thus trivialising it. As with previous example, the deformation theory also involves a weight 4 modular form, here of level 25.

```
> f := CyclotomicPolynomial(5); g := CyclotomicPolynomial(1)^4;
```

```
> H := HypergeometricData(f,g : Print:="alpha_beta");
> H, Weight(H); // weight 3
Hypergeometric data given by [1/5,2/5,3/5,4/5] and [0,0,0,0]
3
> t := 11^5; // 11 is now good, as is raised to 5th power
> T := PolynomialRing(Rationals()).1;
> f2 := (1-T+8*T^2)*(1+2*T); // could have Magma compute these
> f3221 := (1-76362*T+3221^3*T^2)*(1-3221*T); // wt 4 lev 25
> // degree 4 factor at 11 comes from Grossencharacter
> // in fact, this is the t=0 deformation: sum_i x_i^5 = 0
> K<z5> := CyclotomicField(5);
> p5 := Factorization(5*Integers(K))[1][1]; // ramified
> G := HeckeCharacterGroup(p5^2);
> psi := Grossencharacter(G.0,[[3,0],[1,2]]);
> f11 := EulerFactor(LSeries(psi),11 : Integral); f11;
1771561*x^4 - 118459*x^3 + 3861*x^2 - 89*x + 1
> BP := [<2,1,f2>,<5,4,1>,<11,0,f11>,<3221,1,f3221>];
> L := LSeries(H,t : BadPrimes:=BP);
> Conductor(L); // 2*5^4*3221, 5^4 is somewhat guessed
4026250
> LSetPrecision(L,5);
> LCfRequired(L); // approx with old CheckFunctionalEquation
12775            //
> time CFENew(L); // actually needs much fewer now
1.5259E-5
Time: 4.290
```

Again one need not specify all the bad prime information, as Magma can automatically compute it at multiplicative and tame primes (however, the local conductor at 5 must be specified).

```
> EulerFactor(H,t,11); // tame
1771561*T^4 - 118459*T^3 + 3861*T^2 - 89*T + 1
> EulerFactor(H,t,2); // multiplicative
16*T^3 + 6*T^2 + T + 1
> EulerFactor(H,t,3221); // multiplicative
-107637325775281*T^3 + 33663324863*T^2 - 79583*T + 1
```

One can also choose t so as to trivialise the wild prime 5.

```
> MValue(H); // 5^5;
3125
> t := 11^5/5^5;
> f5 := EulerFactor(H,t,5 : Fake); // v_5(Mt)=0
> f5;
15625*T^4 - 125*T^3 - 45*T^2 - T + 1
> L := LSeries(H,t : BadPrimes:=[<5,0,f5>]);
> Conductor(L); // 2*3*26321, Magma computes Euler factors
157926
> LSetPrecision(L,9); // about 4000 terms
> CFENew(L);
```

```
-2.32830644E-10
> t := -11^5/5^5; // another choice with v_5(Mt)=0
> f5 := EulerFactor(H,t,5 : Fake); // v_5(Mt)=0
> f5; // four possible Euler factors, one for each Mt mod 5
15625*T^4 + 1750*T^3 + 230*T^2 + 14*T + 1
> L := LSeries(H,t : BadPrimes:=[<5,0,f5>]);
> Conductor(L); // 2*31*331, Magma computes Euler factors
20522
> LSetPrecision(L,9); // about 1300 terms
> CFENew(L);
4.65661287E-10
```

Example H1E9

Here is an example with tame primes. This derives from comments of Rodriguez-Villegas. The idea is to take hypergeometric data that has weight 0 or 1, and compare it to Artin representations or hyperelliptic curves.

```
> T := PolynomialRing(Rationals()).1; // dummy variable
> H := HypergeometricData([3,4,6,12],[1,1,5,5]); // degree 10
> b, HC := IsHyperelliptic(CanonicalCurve(H)); // genus 5
> assert b; Genus(HC);
5
> EulerFactor(Specialization(HC,13^12),13); // 13 becomes good
371293*T^10 - 285610*T^9 + 125229*T^8 - 31096*T^7 + 4810*T^6
 - 540*T^5 + 370*T^4 - 184*T^3 + 57*T^2 - 10*T + 1
> EulerFactor(H,13^12,13); // use hypergeometric methods
371293*T^10 - 285610*T^9 + 125229*T^8 - 31096*T^7 + 4810*T^6
 - 540*T^5 + 370*T^4 - 184*T^3 + 57*T^2 - 10*T + 1
> assert &and[EulerFactor(Specialization(HC,p^12),p)
>              eq EulerFactor(H,p^12,p) : p in [11,13,17,19]];
> assert &and[EulerFactor(Specialization(HC,t0*13^12),13)
>              eq EulerFactor(H,t0*13^12,13) : t0 in [1..12]];
```

One can take a smaller power than the 12th, but then the curve will not become completely good at the prime. However, the hypergeometric calculations will still be possible.

```
> EulerFactor(H,17^4,17);
17*T^2 + 2*T + 1
> EulerFactor(H,19^9,19); // takes the Phi_3 term
19*T^2 + 7*T + 1
> EulerFactor(H,19^6,19);
361*T^4 + 114*T^3 + 31*T^2 + 6*T + 1
> EulerFactor(H,1/11^5,11); // degree is phi(1)+phi(5)
-T^5 + 1
> EulerFactor(H,4/11^5,11); // degree is phi(1)+phi(5)
```

```
-T^5 + 5*T^4 - 10*T^3 + 10*T^2 - 5*T + 1
```

A similar exploration is possible with a weight 0 example. Here the Artin representation machinery is better able to cope with partially good primes.

```
> H := HypergeometricData([2,3,6],[1,5]); // degree 5
> Q := Rationals();
> EulerFactor(ArtinRepresentation(H,7^6),7 : R:=Q);
-T^5 - T^4 - T^3 + T^2 + T + 1
> EulerFactor(ArtinRepresentation(H,7^3),7 : R:=Q);
T^2 + T + 1
> EulerFactor(ArtinRepresentation(H,7^2),7 : R:=Q);
-T + 1
> EulerFactor(ArtinRepresentation(H,2/11^5),11 : R:=Q);
-T^5 + 5*T^4 - 10*T^3 + 10*T^2 - 5*T + 1
> EulerFactor(H,7^6,7); // compute it directly from H
-T^5 - T^4 - T^3 + T^2 + T + 1
> EulerFactor(H,7^3,7);
T^2 + T + 1
> EulerFactor(H,7^2,7);
-T + 1
> EulerFactor(H,2/11^5,11);
-T^5 + 5*T^4 - 10*T^3 + 10*T^2 - 5*T + 1
```

Example H1E10

In general one should be able to relate the tame Euler factors to Grössencharacters. This is partially considered (in a different guise) in [Wei74] and [Wei76]. In particular, for primes p that are 3 mod 4, we can take the hypergeometric data given by the squares modulo p, and find that at a prime l that is 1 mod p with a suitably normalised t-value whose valuation $v_l(t)$ at l is p, the Euler factor is a power (and Tate twist) of that for the canonical Grössencharacter of $\mathbf{Q}(\sqrt{-p})$.

```
> p := 11; assert p mod 4 eq 3 and p ne 3 and IsPrime(p);
> SQ := [-Integers()!x : x in GF(p) | IsSquare(x) and x ne 0];
> H := HypergeometricData([* x : x in [-&+SQ] cat SQ *]);
> GammaList(H);
[* -1, -3, -4, -5, -9, 22 *]
> Weight(H); assert Weight(H) eq (p-5)/2;
3
> K := QuadraticField(-p);
> I := Factorization(p*Integers(K))[1][1];
> G := HeckeCharacterGroup(I); // get Tate twist of canonical GR
> u := (Weight(H)+ClassNumber(K)) div 2;
> v := (Weight(H)-ClassNumber(K)) div 2; //u+v=wt, u-v=classno
> u,v;
2 1
> GR := Grossencharacter(G.0,[[u,v]]); // [6,3] for sqrt(-23)
> for l in [l : l in PrimesUpTo(1000) | l mod p eq 1] do
>   ef := EulerFactor(H,l^p/MValue(H),l);
```

```
>   F := Factorization(ef);
>   assert #F eq 1 and F[1][2] eq (p-1)/2;
>   assert F[1][1] eq EulerFactor(GR,l : Integral);
>   printf "%o %o\n",l,F[1][1];
>   end for;
23 12167*x^2 + 207*x + 1
67 300763*x^2 - 871*x + 1
89 704969*x^2 + 801*x + 1
199 7880599*x^2 + 3980*x + 1
```

Example H1E11

Here is an example of the use of PossibleHypergeometricData, enumerating the number of possibilities in small degree. A speed test is also done for the save-limit code.

```
> for d in [1..8] do
>    [#PossibleHypergeometricData(d : Weight:=w) : w in [0..d-1]];
>    end for;
[ 1 ]
[ 3, 10 ]
[ 3, 0, 10 ]
[ 11, 74, 30, 47 ]
[ 7, 0, 93, 0, 47 ]
[ 23, 287, 234, 487, 84, 142 ]
[ 21, 0, 426, 0, 414, 0, 142 ]
[ 51, 1001, 1234, 3247, 894, 1450, 204, 363 ]
> D4w1 := PossibleHypergeometricData(4 : Weight:=1);
> D := [HypergeometricData(x) : x in D4w1]; // 12 are self-twists
> #[x : x in D | Twist(x) eq x or Twist(x) eq SwapAlphaBeta(x)];
12
> #PossibleHypergeometricData(4 : Weight:=1,TwistMinimal);
43
> #PossibleHypergeometricData(4 : Weight:=1,Primitive);
64
> // speed test for SaveLimit
> H := HypergeometricData([1/2,1/2,1/2,1/2],[0,0,0,0]);
> HypergeometricMotiveSaveLimit(2000);
> time _:=LGetCoefficients(LSeries(H,-1),2000);
Time: 1.040
> time _:=LGetCoefficients(LSeries(H,-1),2000);
Time: 0.540
> HypergeometricMotiveClearTable();
> time _:=LGetCoefficients(LSeries(H,-1),2000);
Time: 1.030
```

1.3.1 Special Hypergeometric Motives

As indicated partially in the previous description, one can specialize the (singular) parameter $t = 1$ in a family of a hypergeometric motives, and still get a arithmetic object. In doing this, the degree drops by 1, or 2 if the original datum has even degree and odd weight. However, the weight stays the same. All primes that are not involved with the cyclotomic data can be considered "multiplicative" in that $p|(t-1)$. Magma contains a "database" of bad Euler information for all HGMs up through degree 6. Some examples are given below.

Example H1E12

Note that any degree 2 weight 1 hypergeometric datum will have a $t = 1$ specialization with a degree 0 L-function, that is, the trivial L-function that is identically 1. One of the degree 2 weight 0 data gives the Riemann ζ-function, and the other two give Artin representations (which could be written as Dirichlet characters).

```
> H := HypergeometricData([6],[1,2]);
> L := LSeries(H,1); L;
L-series of Artin rep C2: (1,-1) of ext<Q|x^2-3>, conductor 12
> H := HypergeometricData([3],[1,2]); // twist of above
> L := LSeries(H,1); L;
L-series of Riemann zeta function
> H := HypergeometricData([4],[1,2]);
> L := LSeries(H,1); L;
L-series of Artin rep C2: (1,-1) of ext<Q|x^2-2>, conductor 8
```

Magma automatically identifies the examples where the resulting degree is 2 or less, and also some additional Artin representations. However, for (Φ_5, Φ_6^2) and its twist, the resulting newform has level 5400, and as the identification in this space might be expensive, it is not made.

```
> H := HypergeometricData([12],[1,2,3]); // degree 4
> L := LSeries(H,1); L; // drops to degree 3
L-series of Artin rep S4: (3,-1,1,0,-1) of ext<Q|x^4-4*x-6>, cond 6912
> // both the next two drop to degree 2 (odd wt)
> H := HypergeometricData([2,2,6],[10]);
> L := LSeries(H,1); L; CremonaReference(L`parent);
L-series of Elliptic Curve y^2 + y = x^3 - 75*x + 256 over Q
225e1
> H := HypergeometricData([2,2,3],[4,4]);
> L := LSeries(H,1); L; Conductor(L);
L-series of q - 2*q^5 + 12*q^7 - 60*q^11 + O(q^12)
288
```

In general, the returned object is just an L-series label, with the correct bad Euler data attached. There are also various examples (necessarily in even weight) where a translate of the Riemann ζ-function is a factor of L-series.

```
> H := HypergeometricData([3,8],[1,2,2,2,6]);
> Degree(H), Weight(H);
6 2
```

```
> L:=LSeries(H,1);
> Degree(L); // drops by 1
5
> L;
(L-function for param t=1 of HG data given by [3,8] and [1,2,2,2,6] /
 Translation by 1 of L-series of Riemann zeta function) *
(Translation by 1 of L-series of Riemann zeta function)
> BadPrimeData(L); // conductor 2^7 * 3^2
[
    <2, 7, 8*x^3 - 4*x^2 - 2*x + 1>,
    <3, 2, -27*x^3 - 3*x^2 + x + 1>
]
> CFENew(L); // check the bad Euler info
0.000000000000000000000000000000
```

In general, the local conductors bad Euler factors (possibly trivial) were found by a process of trial and error, aided by ideas from David Roberts and Fernando Rodriguez-Villegas.

```
> H := HypergeometricData([5,6],[1,1,2,2,3]); // deg 6 wt 1
> L := LSeries(H,1); // degree 4
> BadPrimeData(L);
[
    <2, 1, -2*x^3 + x + 1>, // (1-x)*(1+2x+2x^2)
    <3, 1, -3*x^3 + 2*x^2 + 1>, // (1-x)*(1+x+3x^2)
    <5, 4, 1>
]
> LSetPrecision(L,15);
> CFENew(L);
0.000000000000000
> //
> H := HypergeometricData([14],[1,1,1,2,3]); // deg 6 wt 2
> L := LSeries(H,1); // degree 5
> BadPrimeData(L); // cond too large to check here
[
    <2, 10, 1>,
    <3, 3, 3*x^2 - 4*x + 1>, // (1-x)(1-3x), mixed weight
    <7, 5, 1>
]
```

There are some exotic examples where the specialized L-function factors into degree 2 L-functions of modular forms (or elliptic curves), or at least has an identifiable factor of smaller degree.

```
> H := HypergeometricData([2,2,4,6],[3,10]);
> L := LSeries(H,1); // reduce to deg 4 and wt 1
> E1 := EllipticCurve("400c");
> E2 := EllipticCurve("1200l");
> X := LSeries(E1)*LSeries(E2);
> &and[EulerFactor(L,p) eq EulerFactor(X,p) : p in PrimesUpTo(100)];
true
> //
```

```
> H := HypergeometricData([2,2,2,2,2,2],[1,1,1,1,1,1]);
> L := LSeries(H,1); // reduce to deg 4 and wt 5
> f4 := Newforms(CuspForms(8,4))[1][1]; // modular forms
> f6 := Newforms(CuspForms(8,6))[1][1]; // of level 8
> Y := Translate(LSeries(f4),1)*LSeries(f6);
> &and[EulerFactor(L,p) eq EulerFactor(Y,p) : p in PrimesUpTo(100)];
true
> //
> H := HypergeometricData([3,3,3,3],[6,6,6,6]);
> L := LSeries(H,1); // deg 6 weight 7
> P := PrimesInInterval(5,100);
> &+[1.0*Coefficient(EulerFactor(L,p : Degree:=1),1)^2/p^7 : p in P]/#P;
1.99744382297573720150050401754 // imprimitive, 2 factors
> f6 := Newforms(CuspForms(36,6))[2][1]; // level 36 modwt 6
> LQ := Translate(LSeries(f6),1); // make it weight 7
> P30 := PrimesInInterval(5,30);
> &and[IsDivisibleBy(EulerFactor(L,p),EulerFactor(LQ,p)) : p in P30];
true
```

1.4 Jacobi Motives

A topic related to hypergeometric motives is that of Jacobi sum motives. These are indeed simpler, and in fact the tame prime information for hypergeometric motives can be determined from Jacobi motives, possibly twisted by Kummer and Tate characters.

The classical Jacobi sums were indicated by Weil to come from Grössencharacters [Wei52], and this functionality is also included, with it indeed being the preferred method to compute Euler factors and the L-series, once the reciprocity correspondence has been established and the Grössencharacter identified.

1.4.1 Background

Let $n_j \in \mathbf{Z}$ and $x_j \in \mathbf{Q}/\mathbf{Z}$ with $\theta = \sum_j n_j \langle x_j \rangle$ an element of the free group on $\mathbf{Q}/\mathbf{Z}$ with $\sum n_j x_j \in \mathbf{Z}$.

Letting m be the least common multiple of the denominators of the x_j, the field of definition K_θ is a subfield of $\mathbf{Q}(\zeta_m)$, corresponding by class field theory to quotienting out by $(\mathbf{Z}/m\mathbf{Z})^\star$ by elements which leave θ fixed when scaling by them. When scaling by -1 fixes θ this field K_θ is totally real, and otherwise it is a CM field.

For primes p with $\gcd(p, m) = 1$, we consider Gauss sums corresponding for prime ideals $\mathfrak{p}$ in $\mathbf{Q}(\zeta_m)$, defined by

$$G^{\psi}_{a/m}(\mathfrak{p}) = -\sum_{x \in \mathbf{F}^\star_{\mathfrak{p}}} \left(\frac{x}{\mathfrak{p}}\right)^a_m \psi\left(\mathrm{Tr}^{\mathbf{F}_{\mathfrak{p}}}_{\mathbf{F}_p} x\right),$$

where ψ is a nontrivial additive character on $\mathbf{F}_p$ and the power residue symbol takes values in the roots of unity of $\mathbf{Q}(\zeta_m)$ with

$$\left(\frac{x}{\mathfrak{p}}\right)_m^a \equiv x^{(q-1)a/m} \pmod{\mathfrak{p}}.$$

The associated Jacobi sum evaluation for θ at $\mathfrak{p}$ is then given by $\prod_j G_{x_j}(\mathfrak{p})^{n_j}$ with the result being independent of the choice of additive character ψ. This defines the Jacobi sum for good primes $\mathfrak{p}$ up to a choice of ζ_m into $\mathbf{C}$.

If one is just interested in Euler factors over $\mathbf{Q}$ and not K_θ, then a p-adic method using the Gross-Koblitz formula can also be used. There are known bounds on the conductor of the resulting L-function, the first being that of Weil [Wei52].

1.4.2 Kummer and Tate Twists

A Jacobi motive can also be Kummer twisted by t^ρ for some rational ρ and nonzero rational t. This can increase the field of definition so that m includes the denominator of ρ. This corresponds to multiplying the various Jacobi sum evaluations by suitable roots of unity.

Often one wants to Tate twist the Jacobi sum to gets its effective weight, and for this reason the full unit is sometimes called a Jacket motive (for Jacobi, Kummer, and Tate).

1.5 Jacobi Motive Functionality

1.5.1 Creation Functions

`JacobiMotive(A, B)`

`Kummer`	SeqEnum	*Default :* $[1,0]$
`Tate`	RngIntElt	*Default :* 0
`Weight`	RngIntElt	*Default :*

Given two sequences of rationals, corresponding to positive and negative elements in the free group on $\mathbf{Q}/\mathbf{Z}$, create the resulting Jacobi motive. This requires that the signed sum of the rationals is an integer. The optional `Kummer` vararg can specify a Kummer twist, and similarly with the `Tate` vararg. Alternatively, the desired Tate twist can be obtained by giving and integral argument to the `Weight` vararg, and the effective Tate twist can be obtained by setting `Weight` as `true`. A variant with only one argument (B is empty) is also available.

`JacketMotive(A, B, t, rho, j)`

Similar to above, this intrinsic spells out the Jacobi summands (A, B), the Kummer twist t^ρ, and the Tate twist j explicitly.

`KummerTwist(J, t, rho)`

Given a Jacobi motive J, return its Kummer twist by t^ρ, where ρ is rational and t is a nonzero rational.

`TateTwist(J, j)`

Given a Jacobi motive J and an integer j, return the jth Tate twist of J.

1.5.2 Operations

`J1 * J2`

Given two Jacobi motives, take their tensor product, eliminating any rationals common to the positive and negative parts.

`J1 / J2`

Given two Jacobi motives, take their tensor quotient, eliminating any rationals common to the positive and negative parts.

`J1 eq J2`

`J1 ne J2`

Whether two Jacobi motives are equal, that is, whether they have the same positive and negative parts, their t^ρ Kummer twists are the same, and they have the same Tate twist parameter.

`Scale(J, q)`

Given a Jacobi motive, scale all the rational numbers defined the datum by the given rational q. The denominator of q must be coprime to m, and q must be invertible mod m. The resulting motive will be identical over $\mathbf{Q}$ but need only be conjugate over K_θ.

1.5.3 Attributes

`Field(J)`

The field of definition of a Jacobi motive.

`Weight(J)`

The motivic weight of a Jacobi motive.

`EffectiveWeight(J)`

The effective motivic weight of a Jacobi motive, that is, the width of its Hodge structure.

`HodgeStructure(J)`

`HodgeVector(J)`

`EffectiveHodgeStructure(J)`

The Hodge structure of a Jacobi motive.

1.5.4 L-function

`EulerFactor(J, p)`

Degree	RngIntElt	*Default :*
Roots	BoolElt	*Default :* false

Given a good prime p, that is, one which is coprime to m and the Kummer twisting parameter t, compute its Euler factor. The `Roots` vararg also returns as a second argument the p-adic approximations to the roots (associated to the prime ideals above p). The `Degree` vararg can be used when the full Euler factor is not needed, though it is often still just as easily computed. It is often easier to first identify the Jacobi motive as a `Grossencharacter`, and then compute its Euler factors.

`ComplexEvaluation(J, P)`

Precision	RngIntElt	*Default :*

Given a Jacobi motive and a good degree 1 prime over K_θ, compute the associated Jacobi sum as a complex number. This is used to identify motives that are equivalent over $\mathbf{Q}$ but not over K_θ in some examples.

`Grossencharacter(J)`

Given a Jacobi motive, identify it as a Grössencharacter. This uses the Weil bound on the conductor, and then tries enough good primes to distinguish the character. This is now the preferred way to compute the `LSeries` of a Jacobi motive (though the latter still exists).

1.6 Jacobi Motive Examples

Example H1E13

Here are some simple examples of Jacobi motives. The first involves the Fermat cubic, and various twists.

```
> J := JacobiMotive([2/3,2/3],[1/3]); // CM elliptic curve conductor 27
> Weight(J);
1
> Field(J);
Number Field with defining polynomial y^2 - y + 1 over Q
> P := PrimesInInterval(11,100);
> E := EllipticCurve("27a");
> &and[EulerFactor(J,p) eq EulerFactor(E,p) : p in P];
true
> K := KummerTwist(J, 2, 1/3); // twist by 2^(1/3)
> Et := EllipticCurve("108a");
> &and[EulerFactor(K,p) eq EulerFactor(Et,p) : p in P];
true
> K4 := KummerTwist(J, 4, 1/3); // twist by 4^(1/3)
> E36 := EllipticCurve("36a");
```

```
> &and[EulerFactor(K4,p) eq EulerFactor(E36,p) : p in P];
true
> K2 := KummerTwist(J, -2, 1/2); // quadratic twist by 2
> Q := QuadraticTwist(E,-2);
> &and[EulerFactor(K2,p) eq EulerFactor(Q,p) : p in P];
true
```

Example H1E14

The next example is related to the Klein quartic and the elliptic curves of conductor 49.

```
> J := JacobiMotive([1/7,2/7,4/7]);
> Scale(J,2) eq J; // scaling by 2 or 4 gives same motive
true
> Field(J);
Number Field with defining polynomial y^2 - y + 2 over the Rational Field
> Weight(J);
3
> EffectiveWeight(J);
1
> T := TateTwist(J,1); // this twist is weight 1
> Weight(T);
1
> P := PrimesInInterval(11,100);
> E := EllipticCurve("49a");
> &and[EulerFactor(T,p) eq EulerFactor(E,p) : p in P];
true
> Grossencharacter(J);
Grossencharacter of type [[ 1, 2 ]] for Hecke-Dirichlet pair (1,$.1)
with modulus of norm 7 over Number Field y^2 - y + 2
> TateTwist($1,1);
Grossencharacter of type [[ 0, 1 ]] for Hecke-Dirichlet pair (1,$.1)
with modulus of norm 7 over Number Field y^2 - y + 2
```

Example H1E15

We next give an example with some tensor arithmetic.

```
> J := JacobiMotive([1/3,1/3,1/3]); // weight 3, effective wt 1
> J;
Jacobi motive given by 3*[1/3]
> Grossencharacter(J);
Grossencharacter of type [[ 1, 2 ]] for Hecke-Dirichlet pair
(1,$.1*$.2^2) with modulus of norm 9 over Number Field y^2 - y + 1
> J^3;
Jacobi motive given by 9*[1/3]
> Grossencharacter(J^3); // norm 3
Grossencharacter of type [[ 3, 6 ]] for Hecke-Dirichlet pair
```

```
(1,$.1) with modulus of norm 3 over Number Field y^2 - y + 1
> Grossencharacter(J^6); // norm 1
Grossencharacter of type [[ 6, 12 ]] for Hecke-Dirichlet pair
(1,1) with modulus of norm 1 over Number Field y^2 - y + 1
> A := JacobiMotive([2/3,2/3],[1/3]);
> J*A; // one of the 1/3 cancels
Jacobi motive given by 2*[1/3]+2*[2/3]
> Grossencharacter(J*A); // trivial character
Tate twist by -2 of Hecke character 1 in Group of Hecke characters
of modulus of norm 1 over Number Field y - 1
> K := KummerTwist(J*A, 3, 1/3); K;
Jacobi motive given by 2*[1/3]+2*[2/3] with Kummer twisting
parameters (t,rho)=(3,1/3)
> Grossencharacter(K); // bad reduction at 3
Grossencharacter of type [[ 2, 2 ]] for Hecke-Dirichlet pair (1,1)
with modulus of norm 81 over Number Field y^2 - y + 1
> K2 := KummerTwist(JacobiMotive([]),3,1/3); K2;
Unital Jacobi motive with Kummer twisting parameters (t,rho)=(3,1/3)
> Grossencharacter(K2); // twist of the above
Grossencharacter of type [[ 0, 0 ]] for Hecke-Dirichlet pair
(1,$.1^2) with modulus of norm 81 over Number Field y^2 - y + 1
> K/K2 eq J*A; // removing the Kummer twist from K, gives J*A
true
```

Example H1E16

Here is an example with motives that are the same over $\mathbf{Q}$, but not over $\mathbf{Q}(\sqrt{-3})$. One of the Grössencharacters is seen to agree with the Jacobi motive at prime ideals.

```
> J := JacketMotive([], [], 2, 1/3, 0); J; // twist by 2^(1/3)
Unital Jacobi motive with Kummer twisting parameters (t,rho)=(2,1/3)
> GR := Grossencharacter(J); GR;
Grossencharacter of type [[ 0, 0 ]] for Hecke-Dirichlet pair (1,$.1)
with modulus of norm 36 over Number Field y^2 - y + 1
> ef := func<G,p | EulerFactor(G,p : Integral)>;
> P := PrimesUpTo(100);
> &and[ef(GR,p) eq ef(GR2,p) where GR2:=GR^2 : p in P]; // same over Q
true
> p7 := Ideal(Decomposition(Field(J),7)[1][1]); // norm 7, degree 1
> GR(p7);
-0.499999999999999999999999999998 - 0.866025403784438646763723170754*i
> (GR^2)(p7);
-0.500000000000000000000000000001 + 0.866025403784438646763723170752*i
> ComplexEvaluation(J,p7); // same as GR, not GR^2
-0.499999999999999999999999999998 - 0.866025403784438646763723170754*i
> PK := [p : p in PrimesUpTo(100,Field(J)) | Norm(p) ne 3];
> Max([Norm(GR(p)-ComplexEvaluation(J,p)) : p in PK | Degree(p) eq 1]);
```

```
1.01121993374575362499804556062E-59
```

Example H1E17

Jacobi motives also can be used to determine the tame prime information for a hypergeometric motive. It is easiest to write the hypergeometric datum in its `GammaList` form when doing this. The appropriate Kummer-Tate twist of the Jacobi motive gives the tame hypergeometric Euler factor corresponding to trivialised inertia near $t = 0$ or $t = \infty$.

```
> H := HypergeometricData([* 1,2,3,3,6, -5,-5,-5 *]); H;
Hypergeometric data given by [ 1, 1, 2, 2, 3, 3, 3, 6 ] and [ 5, 5, 5 ]
> Weight(H);
3
> POS := [x/5 : x in GammaList(H) | x ge 0];
> NEG := [x/5 : x in GammaList(H) | x le 0];
> J:=JacobiMotive(POS,NEG); J; // weight 5
Jacobi motive given by 2*[1/5]+[2/5]+2*[3/5]
> p := 11; // check 5th powers for beta value
> EulerFactor(H,1/p^5/MValue(H),p); // weight 3-2
121*x^4 + 11*x^3 - 9*x^2 + x + 1
> EulerFactor(TateTwist(J,2),p); // weight 5-2*2
121*x^4 + 11*x^3 - 9*x^2 + x + 1
> [EulerFactor(H,u/p^5/MValue(H),p) : u in [1..p-1]];
[
    121*x^4 + 11*x^3 - 9*x^2 + x + 1,
    121*x^4 + 121*x^3 + 51*x^2 + 11*x + 1,
    121*x^4 - 44*x^3 + 6*x^2 - 4*x + 1,
    121*x^4 - 99*x^3 + 41*x^2 - 9*x + 1,
    121*x^4 + 11*x^3 + 21*x^2 + x + 1,
    121*x^4 + 11*x^3 + 21*x^2 + x + 1,
    121*x^4 - 99*x^3 + 41*x^2 - 9*x + 1,
    121*x^4 - 44*x^3 + 6*x^2 - 4*x + 1,
    121*x^4 + 121*x^3 + 51*x^2 + 11*x + 1,
    121*x^4 + 11*x^3 - 9*x^2 + x + 1
]
> [EulerFactor(JacketMotive(POS,NEG,u,1/5,2),p) : u in [1..p-1]];
[
    121*x^4 + 11*x^3 - 9*x^2 + x + 1,
    121*x^4 + 121*x^3 + 51*x^2 + 11*x + 1,
    121*x^4 - 44*x^3 + 6*x^2 - 4*x + 1,
    121*x^4 - 99*x^3 + 41*x^2 - 9*x + 1,
    121*x^4 + 11*x^3 + 21*x^2 + x + 1,
    121*x^4 + 11*x^3 + 21*x^2 + x + 1,
    121*x^4 - 99*x^3 + 41*x^2 - 9*x + 1,
    121*x^4 - 44*x^3 + 6*x^2 - 4*x + 1,
    121*x^4 + 121*x^3 + 51*x^2 + 11*x + 1,
    121*x^4 + 11*x^3 - 9*x^2 + x + 1
]
```

```
> p := 31;
> &and[EulerFactor(H,u/p^5/MValue(H),p) eq
>      EulerFactor(JacketMotive(POS,NEG,u,1/5,2),p) : u in [1..p-1]];
true
```

The same can be done for the α parameters, with positive valuation in the t-values. Here there can be various contributions depending which of the positive γ-values (1,2,3,3,6) divide the valuation.

```
> p := 7; // again take a prime that is 1 mod 6
> d := 6;
> D := Divisors(d);
> POS := [[x/e : x in GammaList(H) | x ge 0] : e in D];
> NEG := [[-x/e : x in GammaList(H) | x le 0] : e in D];
> A, B := CyclotomicData(H);
> for e in D do e,Multiplicity(A cat B,e); end for;
1 2 // weight should be 3 + 1 - 2
2 2 // weight should be 3 + 1 - 2
3 3 // weight should be 3 + 1 - 3
6 1 // weight should be 3 + 1 - 1
> EulerFactor(H,p^2/MValue(H),p);
49*x^2 - 14*x + 1
> f1 := EulerFactor(JacobiMotive(POS[1],NEG[1] : Weight:=2),p);
> f2 := EulerFactor(JacobiMotive(POS[2],NEG[2] : Weight:=2),p);
> f1 * f2;
49*x^2 - 14*x + 1
> function ef(t0,v) ans:=PolynomialRing(Integers())!1;
>   for i in [1..#D] do
>     e := D[i]; if Gcd(v,e) ne e then continue; end if;
>     w := Weight(H)+1-Multiplicity(A cat B,e);
>     J := JacobiMotive(POS[i],NEG[i] : Kummer:=[t0,1/e], Weight:=w);
>     ans := ans*EulerFactor(J,p); end for;
>   return ans; end function;
> for i in [1..100] do v:=Random([1..12]); t0:=Random([1..p-1]);
>   assert EulerFactor(H,t0*p^v/MValue(H),p) eq ef(t0,v); end for;
```

1.7 Bibliography

[**AG12**] Gert Almkvist and Jesús Guillera. Ramanujan-like series for $1/\pi^2$ and String Theory. *Experimental Math.*, 21:223–234, 2012.

[**Gui12**] Jesús Guillera. Collection of Ramanujan-like series for $1/\pi^2$. 2012.

[**Kat90**] N. M. Katz. *Exponential Sums and Differential Equations*, volume 124. Annals of Math. Studies., 1990.

[**Kat96**] N. M. Katz. *Rigid Local Systems*, volume 139. Annals of Math. Studies., 1996.

[**vGvS93**] B. van Geemen and D. van Straten. The cusp forms of weight 3 on $\Gamma_2(2,4,8)$. *Math. Comp.*, 61(204):849–872, 1993.

[**Wei52**] A. Weil. Jacobi Sums as "Grössencharaktere". *Trans. AMS*, 73:487–495, 1952.

[**Wei74**] A. Weil. Sommes de Jacobi et caractères de Hecke. *Göttingen Nachr.*, 1:1–14, 1974.

[**Wei76**] A. Weil. Sur les périodes des intégrales abéliennes. *Comm. Pure Appl. Math.*, XXIX:813–819, 1976.