Kyle's Code

August 12, 2019

Kyle's Code

- Create K, OK, integral basis of OK, U, compute r
- Compute FFF, rootsofginFFF, OF, which are L, θ in L and OL, the splitting field of K
- Set precision and start massive precision loop
- Compute the decomposition of primes in OK with their ramification, inertial degrees, compute h_{ij}
- Compute Completions of K at prime ideals, Kpp, with map mKpp
- Generate minimal polynomial gp of mKpp(theta) in Qp
- Compute decomposition of prime ideal in OL (store only 1), find ramification, inertial degrees
- Compute the completion of FFF at one prime ideal above p in FFF, FFFppF. This is the completion of L at a single prime ideal in L over p, Lp, along with mFFFpF the map $L \mapsto Lp$

My Code

- Create K, OK, integral basis of OK, U, compute r
- Compute FFF, rootsofginFFF, OF, which are L, θ in L and OL, the splitting field of K
- Generate all automorphisms of L, AutL and set ijkL as $\sigma, \sigma^2, \mathrm{id}$
- Generate large upper bound

Kyle's Code

- Generate thetap, roots of gp in FFFppF, the completion of L at a single prime over p, Lp
- Generate ImageOfIntegralBasisElementp, the image of the integral basis for K in Kp, where the map is defined by $\theta \mapsto thetap[i][j]$
- Generate ImageOfpip, ImageOfepsp, the image of π, ε under K in Kp, where the map is defined by $\theta \mapsto thetap[i][j]$
- Compute conjugates of theta in C, thetaC
- Generate ImageOfIntegralBasisElementC, the image of the integral basis for K in C, where the map is defined by $\theta \mapsto Conjugates(\theta)$
  - This is where we invoke the use of the hom function, as Kyle's method sometimes sends elements to 0
- Generate ImageOfpiC, ImageOfepsC, the image of π, ε under K in C, where the map is defined by $\theta \mapsto Conjugates(\theta)$
- Relabel the θ in L from earlier as thetaF
- Generate ImageOfIntegralBasisElementF, the image of the integral basis for K in L, where the map is defined by $\theta \mapsto thetaF[i]$. That is, $\theta \mapsto \theta[i][j] in L$

My Code

-

Kyle's Code

- Generate ImageOfpiF, ImageOfepsF, the image of π, ε under K in L, where the map is defined by $\theta \mapsto thetaF[i]$
- Apply prime ideal removing lemma

My Code

-

Begin to iterate through the cases. For each case, we now have ζ, α. Hence, for each prime

Kyle's Code

-

My Code

- Begin p adic precision loop
- Compute completion of L at one prime ideal above p, FFFppF, with map mFFFpF, Lp, mapLLp. This is
- Compute the decomposition of primes in OK
- Compute Completions of K at prime ideals, Kpp, with map mKpp, Kp, mKp
- Generate minimal polynomial gp of mKpp(theta) in Qp (mKp(th))

$$\begin{tikzcd} L \to L \arrow[r, "\phi"] \arrow[d] & Lp \arrow[d] \\ K \arrow[r] & Kp \end{tikzcd}$$

- Take th in K into L, apply each automorphism $ijkL : L \to L$, apply $mapLLp : L \to Lp$ (mFFFpF). These are the roots of gp in Lp (thetap)
- Find which ijkL[k] map corresponds to thetap[i][j], mapsLL[i][j]

Kyle's Code

- Compute images of zeta, alpha in Qp
- Generate ImageOfzetap, ImageOfalphap, the image of ζ, α under K in Kp, where the map is defined by $\theta \mapsto thetap[i][j]$
- Generate ImageOfzetaC, ImageOfalphaC, the image of ζ, α under K in C, where the map is defined by $\theta \mapsto Conjugates(\theta)$
- Determine i_0, j, k by computing δ_1, δ_2 in FFFppF, the completion of L at prime ideal, Lp (using thetap)
- Verify Special Case 1
- If Special Case 1 holds, recompute α, ImageOfalphap, ImageOfalphaC
- Generate LogarithmicAlphap, the pAdicLof of δ_1, ImageOfpip[k]/ImageOfpip[j], ImageOfepsp[k]/ImageOfepsp[j], etc
- Verify Special Case 2 and see if we can find i_0, j, k for which it holds
- Compute $\hat{i}$
- Generate betas:= -LogarithmicAlphap[i] /LogarithmicAlphap[ihat]
- Compute rootsofginFFFppF[l][i]:= mFFFppF(theta[i][j]), taking theta[i][j], the roots θ in L, into Lp

$$\begin{tikzcd} L \arrow[r, "\phi"] \arrow[d] & Lp \arrow[d] \\ K \arrow[r] & Kp \end{tikzcd}$$

My Code

- Define thetaL (thetap) as ijk[k](L!th)
- Find i0,jj,kk among ijkL using thetaL
- Compute tauL, gammalistL, epslistL, the image of $\alpha * \zeta$, gammalist, epslist using mapsLL[i][j]
- Compute δ_1, δ_2 in L
- Compute δ_1, δ_2 in Lp via mapLLp
- Check Special Case 1, Special Case 2; determine small bound if true
- Generate heuristic precision
- Compute pAdicLog of gammalistL,epslistL in Lp, mapped there via mapLLp
- Compute ihat
- Compute beta:= -LogList[i]/Loglist[ihat]
- Generate ellipsoid [DETAILS]
- Generate pAdicLattice [DETAILS]

Kyle's Code

- In L, we have rootsofginFFF, theta[i][j]. To compute thetap, generate minimal polynomial of theta in Kp, compute roots in Lp. Now, when generating Lp, we also generated the map $\phi : L \to Lp$. Applying $\phi(theta[i][j])$ gives us the same roots theta[i][j] in possibly different order. Hence thetap are the same as rootsofginFFFppF
- Find which rootsofginFFFppF[l][i] corresponds to thetap[i0], thetap[j], thetap[k]; label these as ii0, ijjj, ikkk
- Generate the preimage of ζ, α in L corresponding to i0, j,k using the above ii0,ijjj,ikkk
- Now, we have thetaF, ImageOfpiF, PreimageOfalpha, PreimageOfzeta, ImageOfepsF
- Generate alphaALGEBRAIC, images of δ_1, ImageOfpiF[ikkk]/ImageOfpiF[ijjj] in L

My Code

-

Kyle's Code

-

My Code

-

- Choose one conjugate of $L.1$ in C, giving a map $L \mapsto C$. Use this to compute rootsofginC, theta[i][j] under this map

$$\begin{tikzcd} L \arrow[r, "\phi"] \arrow[d] & \mathbb{C} \arrow[d] \\ K \arrow[r] & \mathbb{C} \end{tikzcd}$$

- In C, we have Conjugates(theta), also known as thetaC. We now have theta[i][j] in L mapped into C, rootsofginC, and these items should be the same
- Find which rootsofginC[i] correspond to thetaC[i0], thetaC[j], thetaC[k]; label these as ii0,ijjj,ikkk
- Generate the preimage of ζ, α in L corresponding to i0, j,k using the above ii0,ijjj,ikkk
- Generate alphaALGEBRAIC, images of δ_1, ImageOfpiF[ikkk]/ImageOfpiF[ijjj] in L
- Compute upper bound $C22$ for cases $s = 0, 1, 2, \geq 3$, use this to determine UpperBoundForn

- alphaALGEBRAIC (and therefore preimages) are only used to generate the upper bound

Begin basic p-adic reduction (LLL)

- Use betas:= -LogarithmicAlphap[i] /LogarithmicAlphap[ihat] to generate p-adic approximation matrix; from imagesinp

Begin basic real reduction (LLL)

- Use LogarithmicAlphaC to generate complex approximation matrix; from imagesinC