Tutorial for p-adics in SAGE

David Roe

March 7, 2007

1 Introduction

p-adics in SAGE are currently undergoing a transformation. Previously, SAGE has included a single class representing $\mathbb{Q}_p$, and a single class representing elements of $\mathbb{Q}_p$. Our goal is to create a rich structure of different options that will reflect the mathematical structures of the p-adics. This is very much a work in progress: some of the classes that we eventually intend to include have not yet been written, and some of the functionality for classes in existence has not yet been implemented. In addition, while we strive for perfect code, bugs (both subtle and not-so-subtle) continue to evade our clutches. As a user, you serve an important role. By writing non-trivial code that uses the p-adics, you both give us insight into what features are actually used and also expose problems in the code for us to fix.

Our design philosophy has been to get a robust, usable interface working first, with simpleminded implementations underneath. We want this interface to stabilize rapidly, so that users' code does not have to change. Once we get the framework in place, we can go back and work on the algorithms and implementations underneath. All of the current p-adic code is currently written in pure Python, which means that it does not have the speed advantage of compiled code. Thus our p-adics can be painfully slow at times when you're doing real computations. However, finding and fixing bugs in Python code is *far* easier than finding and fixing errors in the compiled alternative within SAGE (SageX), and Python code is also faster and easier to write. We thus have significantly more functionality implemented and working than we would have if we had chosen to focus initially on speed. And at some point in the future, we will go back and improve the speed. Any code you have written on top of our p-adics will then get an immediate performance enhancement.

If you do find bugs, have feature requests or general comments, please let me know at roed@math.harvard.edu.

This tutorial attempts to outline what you need to know in order to use the p-adics effectively. OUTLINE SECTIONS.

2 Terminology and types of p-adics

To write down a p-adic element completely would require an infinite amount of data. Since computers do not have infinite storage space, we must instead store finite approximations to elements. Thus, just as in the case of floating point numbers for representing reals, we have to store an element to a finite precision level. The different ways of doing this account for the different types of p-adics.

We have the following definition of the p-adic integers:

$$\mathbb{Z}_p = \lim_{\leftarrow n} \mathbb{Z}/p^n\mathbb{Z}.$$

In order to store a p-adic integer to a given finite precision level, we can merely store it as an element of $\mathbb{Z}/p^n\mathbb{Z}$ for some n.

Definition 2.1 *The* absolute precision *of a finite approximation $\bar{x} \in \mathbb{Z}/p^n\mathbb{Z}$ to $x \in \mathbb{Z}_p$ is the non-negative integer n.*