Ten years of work in *Wiadomości Matematyczne* — an adventure with LaTeX and *Emacs* hacking

Marcin Borkowski

Abstract

Since 2007 I have been working for the "Wiadomości Matematyczne" journal (`http://wydawnictwa.ptm.org.pl/index.php/wiadomosci-matematyczne`), where I am responsible for — among other things — LaTeX-based typesetting. This is enough time to form some habits, and also to make some predictions. I would like to share them with my TeX friends.

1 Introduction

This year marks the tenth anniversary of my work in *Wiadomości Matematyczne*, a (sort of) newsletter of the Polish Mathematical Society. ("Sort of" meaning it appears twice a year, so the "news" in "newsletter" is sometimes more like "olds".) Together with a friend, we were appointed "secretaries", which (at varying points in time) meant everything from handling email, to designing the look-and-feel of the printed issues, to hacking together LaTeX classes to accomplish that design, to proofreading and typesetting papers, making corrections suggested by the authors (or argue why they cannot be made), to actually driving to the post office to send out freshly printed issues to people.

Needless to say, using LaTeX (but also other tools, most notably *Emacs*) is a large part of this undertaking. In the present paper, I would like to share some experiences and thoughts on the matter.

This paper — or tale, if you will — is organized as follows. First, I explain some of the assumptions and policies we set up at the very beginning (surprisingly, many of them did not change). Then I proceed to what is perhaps the most interesting for the typesetting-oriented readers: the LaTeX classes we developed to prepare our pdfs. For the most part I abstain from quoting actual source code; an interested reader may find it on my website at `http://mbork.pl/wiadmatfiles.zip`, so that this paper does not get too TeXnical. Next, I describe some *Emacs* functions which we use in our editing work; some of them are general enough to be of use for the wider public. Finally, I try to summarize my experience with a few general rules of thumb I found out to be useful.

In the paper I sometimes say "I" and sometimes "we". The former means the author; the latter usually means the author together with the other secretary, Paweł Mleczko.

2 Assumptions and policies

When we started working on *Wiadomości Matematyczne*, we did not know much about what we are exactly expected to do. We knew that proofreading would be our primary duty. Soon it turned out that it was quite frustrating when the proofreaders and the typesetter did not remain in strict contact; in fact, the person responsible for typesetting lived in another city. We quickly figured out that if *we* did the typesetting (which we were confident we were capable of, and which we both liked to do), things would go much more smoothly. We also suggested to the editors that a visual overhaul would not be a bad idea.

That meant that we needed to do a few things, even before we dived into coding our LaTeX classes. One of them was deciding how we are going to keep all the incoming files in order. To this end, we adopted a very strict set of rules. It turned out that this was a good idea (and that an even stricter one would be even better!). First, I estimated the overall number of papers we may be dealing with over the course of several years; my estimation was that the number should not exceed a thousand within foreseeable future. Therefore we settled on 4-digits identifiers, just in case. (As of this writing, we are at 521.) Each paper was to get an identifier of the form `art-0000-name`, with the right number in place of the zeros (so we started with `art-0001-...`), and `name` was to be the family name of the (alphabetically) first author, folded to pure ASCII and in lowercase. That way, the identifiers are more-or-less human-readable, too — we humans are not as good as computers in remembering much numerical stuff.

Further, this means that each paper lands in a separate directory called `art-0000-name`, and this directory should contain a LaTeX file (unsurprisingly called `art-0000-name.tex`) and possibly other files, like `art-0000-name-photo-1`, etc. Also, each of these directories is a repository for a version control system (we settled on Mercurial). Finally, alongside those directories, we create directories named like `wm-53-1`, containing issue 1 of volume 53 of *Wiadomości Matematyczne*, with an appropriately named (and version-controlled) LaTeX file inside. With the help of a web-savvy friend we set up a server for all these repositories, so that we could easily pull each other's changes and push our own. After some time, I also wrote a few shell scripts: one for executing some action (like *pull* or *update*) on all (appropriately named) repositories in a local directory, another one for cloning all repositories not present on my computer, etc.

Having that (and a few other things, like deciding on the encoding system — we started with `cp-1250` and at some point moved to `UTF-8`), we started to think about what we would need our LaTeX classes to do. (It turned out much later that many of our assumptions were wrong.) The initial list looked more or less like this.

- Each paper should be typesettable separately, but we need a way to typeset the whole issue, too.
- We need to collect a very specific set of metadata about the paper and the authors, much beyond LaTeX's default trio of title, author's name and date.
- We really want grid typesetting, so we need e.g. heavily customized sectioning, enumerate and theorem-like commands.
- We want LaTeX to do as much as possible for us, but we want a way to influence things manually if needed.

With those (and a few other) things in mind, I started developing the classes. It took maybe a few weeks to get some working prototypes/proofs of concept, and soon we had working LaTeX classes. Obviously, in the course of actually using them, it turned out that they were not exactly ideal. After about five years we decided that our technological debt had risen to an unacceptable level and I decided to rewrite the classes (almost) from scratch. That turned out to be a good decision — the "new" classes, while still fairly complex, are much better to handle. The main goal when writing the "new" classes was simplicity. It turns out that if some obscure case comes up once in, say 20 or 50 articles (or even 10), coding a dozen or more lines of code to cater for that was a mistake. It's much better to deal with such rare cases manually. Of course, if we aimed at total automation of typesetting, the situation would be different; but since we carefully proofread each and every article ourselves anyway, it's more effective to have a simpler, more manageable codebase with clear ways to manually override the default behavior instead of some clever way for TeX to do it itself without any way to influence its decisions.

3 LaTeX classes

As I said, I will not go through the code of our classes *in extenso*; they are shy of 1800 lines of LaTeX code, and it would be too boring anyway. Instead, I am going to highlight a few issues I think are interesting, in a kind of broad view perspective. Anyone wishing to see the nitty-gritty details is invited to look at the class code. It should be available on CTAN at some point; meanwhile, I uploaded all code discussed here on my personal web page.

3.1 Documenting classes

From the very beginning I knew that I'd like my classes well-documented. The first version was written using the `gmdoc` class by Grzegorz Murzynowski. It later had some issues with pdfLaTeX (as opposed to XƎLaTeX), so I dropped it in favor of the classical `ltxdoc`. This turned out to be a not-so-good decision; `doc` is rather unwieldy with the four spaces before `\end{macrocode}`, etc. I would gladly return to `gmdoc` at some point in time.

3.2 One or more?

At first, I decided to have one class for each type of article (a regular paper, an obituary, a book review, etc.) and one for the whole issue. This turned out to be a bad decision. (I was so eager to try out the `docstrip`'s selective inclusion of various source file fragments in various resulting files that I apparently didn't think that through well enough . . .) The benefits of this approach were infinitesimal (in fact, I can think of one only: compiling individual articles must have been faster by a fraction of a second), and the resulting complexity was very difficult to handle. In the "new" classes, I reduced the number of classes dramatically, and instead decided to select the article type with a class option, which works much better — especially since all the code specific to any type of article must be present in the whole-issue class anyway.

3.3 Packages we use

We rely heavily on a number of packages. Here's what they are with a short explanation/rationale for using each of them.

- `xparse`, which helps define commands with complicated syntax,
- `etoolbox`, which makes `\expandafter` and company almost unnecessary,
- `amsmath`, which is fairly obvious for the mathematical content. We use the `intlimits` option with it to preserve Polish typesetting tradition,
- `mathtools`, which I find essential (in fact, I don't know why it's not part of `amsmath`!),
- `pgfkeys` and `pgfopts`, which help define the class options,
- `amsrefs`, which is nicer to use for bibliographies than BibTeX or BibLaTeX: it lets us keep the bibliography in the same file as the rest of the paper, and more importantly, changing the look of the bibliography with it is very easy. We use

the nobysame and initials options, which for some strange reason are not the default,

- MinionPro, since this is the font we use,
- polski, since we typeset in Polish,
- geometry, which is a pretty obvious choice,
- ifpdf, so that the journal logo in eps format can be used when producing a dvi file (does anyone still use dvi, by the way?)
- graphicx, since we often include pictures, and tikz, since we often create them ourselves,
- multicol, since book reviews are typeset in two columns,
- fancyhdr, which (again, for some strange reason) is not included in LaTeX itself,
- enumitem, for obvious reasons (and more on that later),
- booktabs, for obvious reasons,
- adforn, since we want some decorations,
- microtype — very useful, especially that the pages are rather narrow,
- upref, for obvious reasons,
- nicefrac, which is occasionally useful,
- url — we need urls in bibliographies and sometimes elsewhere,
- pdfpages and hyperref, which are needed to prepare pdfs with separate articles for the website (they are made from the whole issue's pdf).

As you can see, the list is quite impressive. This really shows that bare-bones LaTeX is not extremely useful without a host of packages; I feel that many of them should in fact be part of the LaTeX core.

3.4 Docstrip guards

As I mentioned, in the first iteration of the classes we used docstrip to generate separate classes for various article types. Currently, however, we have basically two classes: wm-art (for typesetting a single article) and wm-issue (for typesetting the whole issue). (There is one more, but we will cover that later.) We use article and issue docstrip guards to differentiate the code suitable for only one of those. Most of the code is shared between the two.

3.5 Class options

I decided to use pgfopts for options support. Even though we don't actually use key/value type options, it's nice to use pgf even for simple options which should just execute some piece of code when used. It turns out that option handling with pgf is pretty clever, and even though the learning curve is a bit steep, it is definitely worth the effort.

Each article type has its own option, setting a relevant Boolean switch. There are also other class options, like proof (which enables cropmarks), ebook (which sets the page geometry with minimal margins), etc.

3.6 Macros for typesetting the whole issue

We define two hooks named \AtBeginArticle and \AtEndArticle. In the case of a single article, they just fall back to etoolbox's \AfterEndPreamble and \AtEndDocument; in the case of the whole issue, they add their argument to macros \everybeginarticle and \everyendarticle (using etoolbox's \appto). Also, we define two macros named \ArticleOnly and \IssueOnly, each accepting one argument and expanding to that argument or nothing in respective situations.

Next, we define (only in wm-issue) commands like \Year and \Volume so that we can typeset these in each article's header.

Now comes the fun part. We cannot assume that two articles will not have the exact same \labels, so we have to do something about this. We use \let to save the original meaning of \label and \ref and define our versions, using a prefix \subjobname. That prefix identifies the article (it is set in the wm-issue class to be the article filename sans extension). Also, we make sure that the references use lining (i.e., non-oldstyle) numerals. Of course, we also handle \pageref, \eqref and \cite, all in a similar manner. (The citations are slightly more difficult due to some amsrefs quirks.)

Since at the beginning of each article we want to typeset a header containing (among others) the page range for this article, we need the number of its *last* page. This amounts (more or less) to

```
\AtEndArticle{\origlabel{\subjobname:end}}
```

Again, in reality this is slightly more complicated, since the \origlabel must be put somewhere else for reviews (they are typeset in two columns).

Next up are the macros facilitating loading articles into the whole issue. This is a bit tricky, since each article has its own \documentclass and \begin{document} ... \end{document}. (Nowadays, we have things like docmute and combine, which help with such issues. When I was writing the classes for *Wiadomości Matematyczne*, they didn't exist, or at least I didn't know about them.)

First we define the macros \wmdocumentclass, \wmdocument, \wmenddocument and \wmusepackage, which will be substituted for the respective built-ins. The \wmdocumentclass must set up the article type, reset all the counters like section, footnote, etc., and make sure we start a new page. (The new page thing again is quite tricky, since one of the quirks of *Wiadomości Matematyczne* is that an article can start on the same page as the previous one's end. We achieve this by setting a Boolean switch `newpage` to `true`, but only for those articles that actually need it.) Since we usually need `\DeclareMathOperator` (which is normally defined as a preamble-only command) in individual articles, we reset it to the previously remembered value. (Preamble-only commands work by being set to a special command telling the user that this can only be used in the preamble. To mitigate that, one needs to save the original command to something (with `\let`), then — after the preamble — `\let` the command back to what it was saved to.)

The commands `\wmdocument`, `\wmenddocument` and `\wmusepackage` are pretty simple. The first two just typeset everything saved with `\AtBeginArticle` and `\AtEndArticle` respectively; `\wmusepackage` is just a no-op (articles actually needing external packages are a very rare thing, and in such cases we `\usepackage` them in the preamble of the whole issue manually).

As mentioned, one serious complication arises from the fact that there are some "article groups" with individual articles *not* starting on a new page. We handle that by means of a `Group` environment and two Boolean switches, whose names are self-explanatory: `\ifingroup` and `\iffirstingroup`.

The next big thing is the table of contents. This is basically a big mess (just like in LaTeX itself, although I redid it from scratch insead of trying to coerce the original to work my way). One reason is that we have to cater for the article groups. Another one is that we actually want *two* tables of contents: a Polish one and an English one. Of course, the title version in the ToC may be different than the one in the article and/or its running head. Yet another complication is connected with the so-called "vacats". These are empty pages before an article (they arise naturally when each new article starts on the right, i.e., odd-numbered page, and when the previous article had an odd number of pages). A long-standing tradition in our journal is to put various things on those pages, ranging from pictures of mathematicians' monuments to conference posters to funny quotations. They appear in the ToC, at its end, under a "Miscellanea" section, with all the page numbers put together. Finally, since we may have a lot of UTF-8-only characters in the title, and we do not want `inputenc`'s macro expansions to get into the `.toc` file, we have to make sure that the `\@title` macros and their like are expanded exactly once. All that means a pile of `\expandafter`s, `\unexpanded`s, etc., which could probably be simplified a lot — but it ain't broken, so I'm rather hesitant to fix it. (I assume LaTeX3 might help *a lot* with these issues, but I was not brave enough to use it.)

3.7 Gathering metadata

In standard LaTeX, you have the `\title`, `\author` and `\date` macros. This is far from enough in *Wiadomości Matematyczne*. While we actually do not need the date, we need a lot of additional stuff. For "regular" articles, we need the English title (and also we occasionally want to differentiate between the "normal" title, the title in the running head and the version in the ToC). Also, we have many other types of articles, such as obituaries (where we need the name of the late person, their date of birth and death, a picture and a scan of their signature), articles about prize laureates (which is more or less similar — without the dates, of course, and the signature, but with the prize name instead) and book reviews (with lots of data about the book itself, including a scan of the cover). Since we need a lot of similar `\title`-like macros, I decided to write a macro to write them for me:

```
\newcommand{\DefineDataGrabber}[1]{%
  \csdef{#1}##1{\csdef{@#1}{##1}}}
```

Now, issuing a command like `\DefineDataGrabber {title}` is more or less equivalent to `\def\title#1 {\def\@title{#1}}`, which is kind of cool (and quite Lispy, in fact).

Another interesting thing about gathering article metadata is the collection of author names. Obviously, we need support for more than one author, but how their data are typeset may differ in various places. We need at least four ways of doing that. At the beginning of each article, we just list the author names together with their *cities* (another tradition of the journal). If any author has a nonempty `\authornote`, we include a footnote mark here, too. Also on the first page we want to actually put the author footnotes. At the very end of each article we typeset their names, institutions and emails. Finally, we need the author names (in their "short" form) in the ToC.

The way we handle this is as follows. We define a command `\makeauthorlist`, which (when run) creates an auxiliary macro `\authorlist`, containing first an invocation of `\firstauthor` (with nine parameters corresponding to the author's data), then (if needed) subsequent invocations of an analogous macro `\nextauthor`. Then, `\makeauthorlist` is called `\AtBeginArticle`. When we want to typeset something for each author, we define `\firstauthor` and `\nextauthor` to do what we need (e.g., typeset the names and the cities, or typeset the author footnotes, etc.) and run the `\authorlist` command.

3.8 Design and implementation

With the design, I figured that the hard part was the design itself. When that is done, the LaTeX side of things is usually rather easy.

Well, I was wrong.

The first thing is the font choice. We typeset with *Minion Pro*, but we make our class available to authors, who don't necessarily have that font installed. Hence we check (using `\IfFileExists` for the existence of the file `MinionPro.sty`, and only use that font if this file is present.

We use oldstyle numerals in text, but references are typeset with lining numerals and sometimes their "tabular" version, so we need commands to turn them on when needed. Also, we use Polish-style inequalities (with slanted lines).

We redefine quite a few of LaTeX's skips, like `\baselineskip`. (For some reason, LaTeX redefines the skips `\AtBeginDocument`, so we need to give them twice: once within that command and once without it, since we sometimes need to typeset some material *before* `\begin{document}`.) We redefine `\smallskip`, `\medskip` and `\bigskip` to be equal to a quarter, a half and a whole `\baselineskip`. We also define `\smallskipneg`, `\medskipneg` and `\bigskipneg` — just in case — and much more seldom used "stretch" and "shrink" variants (i.e., zero-length skips with possibility of stretching or shrinking the same amounts). Finally, we redefine skips around displayed equations to be much smaller than the defaults. Again, this must be done `\AtBeginDocument`.

Actually, vertical skips were one of the hardest parts of our classes. You may ask, what is difficult with that? Well, we try to typeset on a grid. This means that we take some care for many things to have the height equal to some multiple of `\baselineskip`. This includes section titles (moderately easy), theorems (rather easy), figures (difficult); when we have displayed equations or quotations, we drop the grid requirement on that page. Since many articles in *Wiadomości Matematyczne* do not contain a lot of math, this works quite well. Unfortunately, vertical skips, page layout and page breaking are one of the darker TeX corners, and I have to admit that TeX behavior in that regard is often a mystery to me.

The next thing in the design department is the page layout. This is accomplished with the help of the `geometry` package. A small complication arises from the fact that — depending on the class options — we want the options to be slightly different. It turned out that a simple `\ifbool` *within the options* works well. (For clarity, I avoid plain TeX's `\ifs`, using `etoolbox`'s `\ifbool`'s instead.)

Each article has a special "header" on its first page. It contains the journal logo, journal name, volume and issue numbers and page numbers for the article. This is all simple: page numbers are there thanks to `\labels` at the beginning and end, the logo is in a pdf file (we use `\IfFileExists` so that the authors using our class who do not have that file can use the class anyway), and absolute positioning on the page is done by the wonderful `tikz` package.

The title and authors are a bit more work, but this work is not difficult, only tedious: we put all that stuff in a box of fixed height (equal to `24\baselineskip`) so that the grid won't be disturbed. One nice touch is that "author footnotes" are distinguished from the usual footnotes by the numbering system: they are "numbered" with Greek letters. Here we also make use of the `\authorlist` mentioned earlier. One thing worth noting is that we need to test if some data (like the author footnotes, for example) is undefined or empty; `etoolbox`'s `\ifdefvoid` is very useful for that.

For articles about prize laureates, new professors and book reviews we want to wrap an image with text. This is notoriously difficult in TeX, but we went the easy way and decided that one paragraph will always be long enough to fill the space around the (rather small) picture. This way, we just set `\hangafter` and `\hangindent` to suitable values, and put the picture in a zero-height `\vbox` within `\vadjust` (taking care of vertical dimensions for everything). If the first paragraph is extremely short (which can happen from time to time), we have a simple remedy:

```
\newcommand{\fakepar}
  {\leavevmode\nobreak\hfil\break}
```

(Notice the lack of `\indent`; this is intentional. Since the `\fakepar` needs to be inserted manually anyway, I wanted it to be general enough to support unindented paragraphs, like sections, etc.)

Finally, `\AtEndArticle` we typeset the author information (it looks slightly different among the various article types, but that is trivial to accomplish).

Some additional care needs to be taken when typesetting section and subsection titles. A section title takes up space equal to three lines (or more in case of long section titles, which we discourage and which seem to never happen anyway). We want 1.5`\baselineskip` above and a half below. However, if a section begins at the top of a page, we do not want any vertical skips above it (and hence also below, because … grid!). This is hard or impossible to achieve automatically, so we have the macro `\attoppage`, setting a suitable switch to true. Also, the `\section` macro has to behave differently at the very beginning of the article: the skip above is then smaller (because we don't need it anyway). It is similar with subsections: normally, a subsection has one full `\baselineskip` above, but not if it directly follows a `\section`. All this is accomplished through special values of `\penalty`s and checks for `\lastpenalty`. (In the initial version of the classes, this mechanism was used much more often, e.g., with theorems; in the current iteration of the class, I decided that was too tricky and decided to go for a simpler solution, with a possibility of easy inserting manual skips, both positive and negative.)

Running headers is the next thing. No surprises here — we use `fancyhdr`, we set up the pagestyles for the first page of the article and for the rest of them, and we define macros `\theleftrunninghead` and `\therightrunninghead` so that the user can easily override them manually.

Theorem-like environments are a much more complicated business. For various reasons I was not satisfied with the LaTeX defaults. In my own papers I usually use `amsthm`, but for *Wiadomości Matematyczne* I decided to go my own way and do all the theorems from scratch. One reason is grid typesetting. A more important one is that by default, the *theorem*'s optional argument is typeset in a TeX box, so its spaces are fixed. With rather short lines this tends to look ugly if the rest of the line happens to be very *loose* in TeX's terms, i.e., its spaces are much wider than usual.

Since I did theorems from scratch anyway, I decided to do them my way. For starters, `\newtheorem` always creates a numbered and a non-numbered (starred) variant. Another thing is the handling of the optional argument. Oftentimes it consists only of a call to `\cite` or `\citelist`; in such a case, LaTeX puts the bracketed output of `\cite` in parentheses, which I don't like. In our case, if the optional argument begins with `\cite`, the parentheses are dropped. Again, this can be manually overridden by using the `\relax` command (which is a no-op, but is different than `\cite`, etc.). Since it is conceivable that someone might want to drop the parentheses for a different reason, we provide a `\noparen` command (a no-op again, but also recognized by the theorem environment, along with `\cite` and `\citelist`). Finally, we define a slew of theorem-like environments by default.

The next thing is enumerations, which are easy: I employ the great `enumitem` package. One unorthodox thing we do in *Wiadomości Matematyczne* is the following. We strongly discourage more than one level of enumerations (and totally forbid more than two), and instead we use various item styles to distinguish between various semantics. For instance, if the items form a conjunction, they are marked with arabic numerals in parentheses; if they form an alternative, they are marked with lowercase Roman numerals with a dot afterwards, etc. I'm not sure whether anyone notices, but I like it that way.

The next topic is bibliographies. The code responsible for them is quite large, but this is mainly because we need to define a lot of bibliography types. As I mentioned, we use `amsrefs`, which I like a lot. One of the greatest things about this package is that defining a new bibliography style is so easy. One of less great things is that it messes internally with the catcode of the apostrophe, which conflicts with the usage of the `\'` *macro*. We need to resort to some `\xdef` hackery because of that when defining the `coauthor`. (We have a special kind of a bibliography in *Wiadomości Matematyczne*: a list of publications by one person. In such a case, the `author` field is not typeset, although we introduce a `coauthor` field, which is typeset at the very end in the form of "(coauthor: ...)" or "(coauthors: ...)". The internal macro containing the `coauthor` field is called `\bib'coauthor`, with the apostrophe as a letter; on the other hand, we need the `\'` control symbol to typeset the Polish word for "coauthor", "współautor".) Finally, we define an environment `bibliography` with *two* optional arguments: the first one is the title of the bibliography (the default dependent on the article type) and the second being a *prefix*, so that we can have two bibliographies in one article, the first one with entries numbered [1], [2], etc., and the second one with entries numbered e.g. [A1], [A2], etc. (this is sometimes needed). Anyway, the main takeaway here is: `amsrefs` is very nice to use and quite hard to hack on.

The last *difficult* thing is inserting figures and tables. We do not use floats, since we prefer to have full manual control over the placement of "floating" material. Therefore we redefine the `figure` and `table` environments. While we are at it, we provide some machinery to control the captions: the `figure*` environment makes them unnumbered, and the figure prefix (like "Fig.") empty by default, but redefinable through `\renewcommand`. We put the figure (with caption) into a `\vbox` of depth zero, measure its height, round it up to the nearest multiple of `\baselineskip` and repackage it into another `\vbox` of the computed size. This way we can retain grid typesetting. If the `figure` environment (or similar) starts in horizontal mode, we use `\vadjust`.

The rest is, happily, much easier. We slightly modify the default design of footnotes, we define a custom `quotation` environment, a simple modification of equation numbers (we want them in tabular numerals, and we provide an `\eqrefr` macro for equation ranges, like "(1–3)"), we define some Polish-specific dashes, etc. One interesting thing is that we have `\emergencystretch` set to 1 pt. This is very useful for narrow columns. We also define some very narrow horizontal skips (half of `\,` and its negative counterpart). We also have a few last-emergency macros for influencing the typesetting. These are `\manualshortenthispage` (expanding by default to `\enlargethispage{-1\baselineskip}`), `\manuallooser` (basically, setting `\looseness`) and the following two:

```
\newcommand{\manualfillpar}
  {{\setlength{\parfillskip}{0pt}\par}}
\newcommand{\manualindentfillpar}
  {{\setlength{\parfillskip}{\parindent}%
  \par}}
```

which allow us to fight very short or very long last lines of a paragraph.

3.9 Making pdf files for individual articles

After typesetting, printing and sending out the whole issue, when the dust settles, we need to prepare a pdf file of every article to put on the website and send to the authors. This is not as easy as typesetting every article separately, since we want the page numbers to be exactly like in the printed issue; also, that would not work in case of articles beginning in the middle of a page. Hence we use `pdfpages` to include the relevant pages from the pdf of the issue.

Our solution is as follows: when the issue is typeset with a special option, `generatefiles`, we invoke a special macro `\generatefile` for each article (using `\AtBeginArticle`). This macro takes the page numbers from the labels and writes out a file named `wm-11-2-333-444.tex`, where `11` stands for the volume number, `2` for the issue number and `333` and `444` for the begin and end pages of the article. This file is very short and consists of setting the pdf metadata and including the relevant pages from `wm-11-2.pdf`. (Also, it uses a special, very simple third class generated from the dtx file.)

There are two main difficulties here. One is that the second issue each year has page numbers resuming where the first one ends, so we need to take care of page number arithmetic. Another is that we don't want to have e.g. ties in pdf title, so we redefine a few standard commands to generate their ASCII equivalents (most notably, both `~` and `\\` expand to a space in this context). Last but not least, we `\usepackage{hyperref}` so that the page numbers in the pdf file matches the ones printed on each page (instead of starting from 1).

4 *Emacs* editing functions

LaTeX classes and general workflow is one thing. Actually editing files is another. We use *Emacs* to get the full editing power available to humanity. Even though it has its flaws, it is an extremely flexible tool. Customizing *Emacs* to work better for *Wiadomości Matematyczne* is an ongoing effort; currently, I am writing functions to automatically generate article templates, help with filling them with some metadata and upload them to our Mercurial repository, all with minimal manual intervention.

In this section, I would like to briefly describe the *Emacs* tools I've made so far, which turn out to be quite general and possibly useful for others.

4.1 Automatic replacement of strings

The first thing is automatic replacement of strings. There are some things that just need to be changed to reasonable defaults. One of them is Polish diacritical signs. We clearly do not want our source code to be littered with things like `\.z\'o\l{}w` instead of proper UTF-8 "żółw". Another is dollar signs (single and double), which we want to be converted to `\(`, `\)`, `\[` and `\]`. Yet another is `\-`, which should be just deleted everywhere, or ties, which need to be inserted after one-letter words and in a few other places and deleted from math mode.

For this, I developed an *Emacs* command called `mrr-auto-replace`. It is configured by means of a list of lists. Each of these lists consists of a regex, optionally followed by a predicate (i.e., a function returning a Boolean value) and by one or more strings. This works as follows: *Emacs* walks through the entire file (buffer, to be more precise) looking for the given regexen, and if the predicate is satisfied in any of the places found, the part matching the regex is replaced by one of the strings given. The strings are cycled, so we can e.g. have the regex `\$\$` (matching two dollar signs) replaced alternately by `\[` and `\]`. The predicate option is useful for distinguishing between math and text modes; AUCTeX (which is an *Emacs* package for interacting with TeX and friends) has a function called `texmathp`, returning a true value if the cursor ("point", in *Emacs*-speak) is in math mode.

It seems simple (and so it is — the source code for `mrr-auto-replace` is less than 20 lines), but it is extremely useful. It is usually one of the first things we run on any LaTeX file received.

4.2 Semi-automatic replacement of strings

Sometimes, however, we cannot trust a machine to do the right thing. For such cases, we have the `mrr-replace-mode` *Emacs minor mode* (i.e., something we can turn on or off in a given buffer). It is configured by means of another list (with a similar structure as previously, though a bit more elaborate here). This time, however, whenever we find an occurrence of any of the regexen from our list, we stop, highlight it and give the user a chance to *select* one of the possible replacements. The use-case should be obvious — one of the possibilities is changing things like `-+` to one of `-`, `--` or `---`. Indeed, we have more than forty such replacement possibilities, and while going through the file using this utility is tedious, it is much less so than if we did that manually. Also, this was much more complicated to code; it takes up more than a hundred lines of code (and as of writing this, it also contains a few minor bugs).

Being a minor mode has the additional advantage that this is *non-modal* in a sense: after turning `mrr-replace-mode` on, only a few keys behave in a special way: `TAB` cycles through the possible replacements (including the original version), `RET` (*Enter* on modern keyboards) looks for the next place to replace, and `C-g` (*Control-g*) quits the mode. This way, if we decide that we have to make some edits other than the ones defined in our list of potential replacements, there is nothing to stop us without exiting the replacement mode.

4.3 Various small hacks

Apart from the two bigger things mentioned above, we have a few smaller tools. For instance, I noticed that if I insert a tie, I almost always delete any space at that point first. Hence I bound the tilde key to a custom *Emacs* command I wrote which does exactly this. Another thing I often do when editing files (as opposed to writing them from scratch) is inserting commas and dashes (the latter often in pairs!). Therefore, I modified the comma-inserting command so that if I type it when *after* a space, *Emacs* inserts it *before* that space anyway. It's very simple, but a very nice time-saver.

For dashes, I have something special. Since usually, when I insert a dash, I need to remove any punctuation in that place (usually a comma), I defined another command to do just that. But more often than not, I want to enclose a fragment of text in dashes. Therefore, when I first select some text and then invoke my command, two dashes are inserted around the selection ("region" in *Emacs* language). Yet another simple command (11 lines of code) making editing much nicer.

4.4 Plans for the future

This is of course not everything *Emacs* can do for us. I noticed that there are numerous repetitive activities I perform when working on articles for *Wiadomości Matematyczne*. One of them is sending emails to the rest of the editors with e.g. pdfs for proofreading. Since I use *Emacs* as my email client (obviously!), I plan to write a command to prepare such an email (with a template text and the pdf attached) automatically. Another one I plan to do one day is a command which would walk across all the articles we are working on and display a summary with each article's status (like "after converting to our class, but before proofreading" or "after sending to the author for proofreading/confirmation"). The possibilities are vast, *Emacs* Lisp is a nice language to work with — it is only a question of time to mold *Emacs* into a system customized to this particular journal's workflow.

5 Summary

As can be seen from this tale, working on a journal is a complicated (but rewarding!) business. From the TeXnical standpoint, there seem to be a few general recurring themes here. The most important (at least for TeXnicians) is that there is no point in trying to force TeX to do everything automatically; it is much better to cover, say, 90% of cases automatically and have facilities for manual override for the remaining 10%. Also, when writing classes for a journal, good knowledge of TeX is very useful. Expansion control and vertical mode are especially important. Moreover, it is usually a good idea to use packages instead of reinventing the wheel whenever possible. And when we have full control over whatever comes into the journal (i.e., we heavily edit all incoming files), redefining even basic LaTeX macros and environments (like, say, the `document` or `figure` environments or the `\usepackage` macro) should not scare anyone away. On the other hand, to minimize the editing effort, it is probably a good idea not to mess around with LaTeX guts. The results will be less appealing aesthetically, though, since authors often make horrible design decisions.

Two paths I *didn't* follow — and which seem worth trying — are using LaTeX3 and restricting what authors can do. The former is obvious — I would expect LaTeX3 to reduce the need for things like `etoolbox` or plain old `\expandafter`s and friends. The latter might be useful, since some authors use LaTeX in an extremely, shall we say, *creative* way (like numbering all footnotes manually or defining dozens of macros making the source file slightly shorter and totally unreadable, or using the very same LaTeX 2.09

style preamble with lots of unnecessary stuff and cargo-cult coding artifacts for all their documents, etc.). Since our authors are mathematicians, they are unfortunately accustomed to using LaTeX; if that were not the case, we might prefer *Markdown* or something similar to restrict the authors' freedom to break things.

From the point of view of an editor who works with text files written by someone else, the obvious takeaway is that you need to use a serious text editor. Which one of the two you choose may be less important: while Vim is difficult to beat in terms of using as few keystrokes as possible to achieve a given transformation of text (a sport known as *vimgolf*), *Emacs* shines in the flexibility/programmability department, and also offers a more comprehensive environment, such as email clients, shell buffers, and a time-tracking/organizational application (the famous *Org-mode*).

In any case, being a secretary of a journal and using LaTeX and *Emacs* is an ongoing adventure that I hope to last for at least another decade.

◇ Marcin Borkowski
Faculty of Mathematics
and Computer Science
Adam Mickiewicz University
ul. Umultowska 87
61-614 Poznań, Poland
`mbork (at) amu dot edu dot pl`
`http://mbork.pl`

Production notes

Karl Berry

This seems an opportune place to say a few words about *TUGboat* production. In general, our process is nothing like as regularized as that described by Marcin.

One immediate difference is that *TUGboat*, by its nature, has to handle articles using any TeX engine. We use pdf(LA)TeX by default, which can handle the majority of articles, but it's typical and reasonable for an article about LuaTeX to require LuaTeX, etc.

So, we can't create an entire *TUGboat* issue in one run. Instead, each article is processed separately into its own PDF. We then concatenate the individual PDFs to make the full-issue PDF to be uploaded to our printer.

To do the concatenation, we've used a variety of tools, most commonly Ghostscript and `pdfjam` (`ctan.org/pkg/pdfjam`) of late. ConTeXt and `pdftk` have also been useful. Different tools are needed as years go by and software and systems change (for no convincing reason).

The same tools can select PDF pages when splicing two articles together, that is, when one article ends and another begins on the same page. We try to avoid this, partly because of the extra production trouble, but primarily because it is better for readers to find new articles starting on new pages. But content must dictate form, so we make it work out when it's needed. (Incidentally, another PDF check is for all fonts being embedded, using `pdffonts` from Xpdf, `foolabs.com/xpdf`.)

The trickiest part of producing the whole issue as a concatenation is the page numbering. We have a control file which lists all the articles in the order in which they will appear, as well as the beginning page number for the issue. Then each article writes its beginning and ending (`\AtEndDocument`) page numbers into external files, where the next article can read them. The two tables of contents use the same external files, so as to ensure consistency of the page numbers.

Unfortunately, nothing comparable keeps titles and authors consistent among the tables of contents and articles. Partly this is due to inertia, partly because it would be hard to implement in full generality, and partly because sometimes there are intentional differences among the three places — forced line breaks, abbreviations, etc.

Back to issue production: the compilation of each article, and the overall process, is done with GNU Make, via a single included Makefile fragment which defines nearly all needed actions. The per-article Makefiles merely give the name of the file, the engine to use (if not `pdflatex`), etc.; the goal being, naturally, to eliminate redundancy wherever possible.

We use GNU Aspell (`gnu.org/s/aspell`) with some `sed` preprocessing to do spell checking: `aspell list \ --mode=tex --add-extra-dicts=‘pwd‘/.dict.pws\ | sort -fu`. The idea being that a given article can have a `.dict.pws` file with the spelling exceptions needed that don't make sense to add to the global exception list (unusual proper names, one-off neologisms, etc.).

Besides spell checking, we've implemented several custom checks across an entire issue, again done in the central Makefile: doubled words (`math.utah.edu/~beebe/software/file-tools.html#dw`), lowercase letters inside `\acro`, tripled letters ("eee"), etc. More globally, we check that the tables of contents aren't missing an article processed in the central control file. Of course, besides the automated checks, humans review each and every word, line, and page that goes out.

Character encodings are an unending hassle. We receive many articles in UTF-8 these days, often with confusion or incorrect usage of accents, dashes, etc., or garbled in transmission. Other articles still use Latin-1 or similar. For articles which have only a few "special" characters, we strongly recommend taking advantage of TeX's inherent capability, and sticking to 7-bit ASCII.

One final point is that all production work is done on Unix (CentOS 7 these days), using TeX Live. Thanks to the well-known portability of TeX documents, there is rarely a problem with an author obtaining different results than the production run, with one glaring exception: when fonts are found by XeTeX or LuaTeX via system lookup, instead of by filename. This makes the document immediately and completely unportable — so I implore everyone, please don't do this in *TUGboat* articles!