User Guide For Peakbagging Made Easy

Martin Bo Nielsen & Emanuele Papini

September 26, 2017

Contents

1 Introduction to PME

This is a guide for using Peakbagging Made Easy (PME). PME is meant to be an easy to use, modular peakbagging routine, for fitting the oscillation power spectra of solar-like stars. Here we define solar-like as a main-sequence star with oscillations driven and damped by convective motion in the outer layers of the stars. It may be possible to fit more evolved or hotter stars, but there is no guarantee that PME won't be confused by this choice and explode.

PME is written in Python and interfaces with FORTRAN for some of the more basic repetitive calculations. The fitting is done using maximum likelihood estimation (MLE), where the parameterspace spanned by the fit variables is sampled using an afine-invariant Markov Chain Monte Carlo sampler for Python called EMCEE [Foreman-Mackey et al., 2013].

1.1 Setup

1.1.1 Dependencies

1. PME requires NumPy, Scipy and EMCEE.
2. A fortran compiler for the system you are using (Intel/Gfortran/AMD/others?)

1.1.2 Getting ready for a peakbagging run

1. Download PME from here
2. Place the contents of PME (*.py/*.f90 files) into a directory of your choice.
3. Place the power spectrum or time series of your star into a directory of your choice. If you only have the time series, PME can compute the power spectrum for you (see below).
4. Make sure the power spectrum file has the *.pow extension, and only 1 file with this extension is in the directory. (This will be patched in future updates)
5. Make sure the power spectrum file consists of 2 columns corresponding to the frequency and power density axes respectively (additional columns are ignored).

PME is essentially just a big script with a few options (actually a lot!), and requires little in the way of installation. However, before using PME the fortran file flops.f90 must be compiled using f2py on your system (f2py is included in typical Numpy installations). Below is an example of how to compile flops.f90 using the intel fortran compiler.

```
f2py --f90exec=ifort --f90flags=-O3 -c flops.f90 -m flops
```

Note that depending on your system the choice of compiler will vary. f2py creates a dynamic library flops.so which is what Python uses. If edits to the fortran file are made, simply recompile the file. However, this should under normal circumstances not be necessary.

1.2 But first, a bit of math (eurgh!)

1.2.1 Model fitting

MLE seeks to find the set of parameters $\boldsymbol{\Theta}$ of a model $M(\boldsymbol{\Theta}, \nu_j)$ that has the highest probability of explaining a set of observations. In this case the observations consist of the power $P(\nu_j)$ of a time series, at frequency ν_j (PME assumes ν is in units of μHz). Here, $\{\nu_j\}$ is a set frequencies that span the range of interest, i.e., the p-mode envelope.

This time series can in principle be anything that reflects the oscillations of a star, i.e., photometric intensity measurements or radial velocity. However, PME assumes that the measurements in the time series are independent (no smoothing), Gaussian random variables (usually true for most types of observations) and at least somewhat regularly spaced[1].

[1] Power spectra of highly irregularly sampled time series are more complicated to fit, and at the moment PME is not setup to do so.

The above implies that the observed power is distributed according to a χ^2 distribution with 2 degrees of freedom [e.g, Woodard, 1984, Duvall and Harvey, 1986, Gizon and Solanki, 2003]. The probability of observing a given value P_j in the power spectrum is thus an exponential distribution:

$$f_j\left(P_j|\mathbf{\Theta}\right) = \frac{1}{M(\mathbf{\Theta},\nu_j)} \exp\left(-\frac{P_j}{M(\mathbf{\Theta},\nu_j)}\right) \tag{1}$$

The best-fit parameters are the parameters that maximize the log-likelihood, which is the logarithm of the joint probability function:

$$L\left(P_j|\mathbf{\Theta}\right) = \ln \prod_{j=1}^{J} f_j\left(P_j|\mathbf{\Theta}\right) = -\sum_{j=1}^{J}\left(\ln M\left(\mathbf{\Theta},\nu_j\right) + \frac{P_j}{M\left(\mathbf{\Theta},\nu_j\right)}\right). \tag{2}$$

Note that the logarithm of the joint probability is used for better numerical stability in the optimization (again, explosions may otherwise occur).

1.2.2 The model

The model, or expectation value of the spectrum is given by [Harvey et al., 1988, Anderson et al., 1990]:

$$M\left(\mathbf{\Theta},\nu\right) = \sum_{n}\sum_{l=0}\sum_{m=-l}^{l} \frac{S_{nlm}/\Gamma_{nlm}\,\mathscr{E}_{lm}\left(i_{nlm}\right)}{1+\left(2/\Gamma_{nlm}\right)^2\left(\nu-\nu_{nlm}\right)^2} + \sum_{k=1}^{K}\frac{A_k/F_k}{1+\left(2\pi\nu\tau_k\right)^{\alpha_k}} + W. \tag{3}$$

In the sums over n,l, and m, S_{nlm} is the mode height, Γ_{nlm} is the full-width at half maximum of the Lorentzian profile, and ν_{nlm} is the mode frequency. $\mathscr{E}_{lm}\left(i\right)$ is the modulation of the azimuthal components of a given multiplet, due to the inclination of the rotation axis of the star. PME can in principle handle as high l as you want, but be warned: if you try $l > 3$ on a *Kepler* time series, people will probably look at you a bit funny.

The sum over k contains the frequency dependent background terms, with amplitude A_k, characteristic frequency F_k and slope α_k. W is the frequency independent white noise level. PME can handle up to $K = 3$, but usually $K = 2$ is sufficient.

In its most basic form, PME would use each of the independent variables in Eq. 3 as fit parameters. However, this is not recommended as the fit will most likely explode, and in fact it doesn't always make sense to do this (i_{nlm} for example, is highly illogical). This setup was chosen to make the code as flexible as possible, at the cost of efficiency and computational speed.

It is often far more beneficial to parameterize the fit parameters as a function of, e.g., frequency. PME treats each parameter in terms of sets, and the user can select a parametrization for each set. These sets are:

1. Mode frequencies (ν_{nlm})
2. Mode heights (S_{nlm})
3. Mode widths (Γ_{nlm})
4. Mode inclination (i_{nlm})
5. Background power-law exponent (α_k)
6. Background characteristic frequency (F_k)
7. Background amplitude (A_k)
8. Background white noise (W)

Everyone has their favorite parametrization, and so PME was designed to make the implementation of new parametrizations easy and fairly straight-forward, however this does require some basic knowledge of Python scripting (see section 4.1).

1.2.3 MCMC sampling

PME uses the ensemble afine-invariant sampler EMCEE by [Foreman-Mackey et al., 2013]. The principle behind ensemble MCMC sampling of a likelihood space can perhaps best be described by the term 'walkers'. The algorithm uses an group of samplers, or 'walkers', that take steps through the parameterspace and record the likelihood at each step. Each step is a linear combination of a walkers current position and that of another randomly chosen walker, which is then stretched by a factor randomly chosen from a suitable distribution. If the change in likelihood from the walkers current position to the new one is sufficiently large, the step is approved with some probability (large steps are likely to be accepted, small are not). This eventually means that the walkers will converge to a maximum somewhere in the likelihood space. Importantly, if all the walkers are stuck in a local maximum, this algorithm makes sure that there is a non-zero chance that one walker might find its way out and on to the global maximum. This will eventually lead to the other walkers finding their way there as well. This algorithm has very few tunable parameters, and for the purposes of PME these are left as the default settings for EMCEE. The two most important parameters for PME are the number of walkers and the number of steps that they will take before the script ends. See 2.2 for more details.

2 PME Basics

PME is broken down into several main functions, which are chosen as options when calling PME in the command line from the directory where you placed it.

The functions are:

- **-autoguess** provides automatic initial guesses for the fit
- **-peakbagging** performs a maximum likelihood fit to the power spectrum using MCMC, **-autoguess** must be run prior to using this option
- **-plot** plots the result of the fit. **-peakbagging** must be run prior to using this option (duh!)
- **-simulation** provides options for simulating random spectra using the results of **-peakbagging**
- **-make_powerspectrum** computes the power spectrum from of a given time series

The number of options available in PME have over time become quite substantial, and as such it is beyond the scope of this document to go through every one them. For a complete list of options for PME do

```
python PME.py --help
```

The main functionality of PME however, lies in the parameterization of the variables, and the user should see the individual **parameterization.py* files for information about each variable set.

2.1 Autoguess

To start using PME, run it with the **-autoguess** option:

```
python PME.py /folder/with/spectrum starID -autoguess
                                           -my_additional_options
```

where 'starID' is a prefix that will be used by PME to find and store the correct files in the given directory. Following this is the main function, in this case **-autoguess**. In general for PME, any additional options (or flags) are simply added in sequence:

```
python PME.py /folder/with/spectrum starID -autoguess
                                           -flag
                                           -numerical_input 1234
                                           -list_input [1,2,3,4]
                                           -string_input mystring
```

The above options are of course just examples to give you a general idea of what inputs can look like. More real-world examples are given below.

2.1.1 A typical run with -autoguess

-autoguess will do its very best to provide initial guesses for the background noise in the spectrum, as well as any oscillation modes that may be present. The background fit is usually very robust, so manual interaction is rarely needed. However, some manual interaction is probably necessary for the mode identification routine, which only tends to pick up the highest amplitude modes.

A typical run with **-autoguess** might look like this:

1. Start by pointing PME at the directory that contains the power spectrum of your star and use the **-autoguess** option

   ```
   python PME.py /folder/with/spectrum kplr001234567 -autoguess
   ```

2. **-autoguess** will now start spewing out lots of info about how it is doing. Currently **-autoguess** is not the most reliable piece of software in the world, which is why there are so many print outs, hopefully this will help find out what the problem is when **-autoguess** eventually fails. The output will look something like this:

   ```
   Trying to read /folder/with/spectrum/kplr001234567.pow.
   Success.
   Starting PME in autoguess mode on kplr004914923
   Attempting to find nu_max and the large separation
   Guessed nu_max = 1794.52 muHz and large separation = 88.73 muHz
   Finding initial guesses for the background terms
   Noise level estimate: 1.110
   Saving background results to /folder/with/spectrum/kplr001234567_output.npz
   Starting mode identification... wish me luck!
   Attempting to define the mode search interval...
   Mode search interval: 895muHz to 2669muHz
   Finding the peaks in the spectrum...
   Found a total of 28 peaks
   Assigning first small separation as l=2,0 pair.
   Working upwards in frequency...
   Identifying highest-frequency mode...
   Working downwards in frequency...
   Identifying lowest-frequency mode...
   Assigning n values.
   Saving diagnostics figures for the background fit.
   Autoguess is done. Have a nice day
   ```

   This is of course under ideal conditions. In this case **-autoguess** finds a total of 28 peaks, and among them at least one $l = 2, 0$ pair. This pair is then used to count sequentially upward and then downward in frequency until no more modes are left in the peak list.

   Typical sources of error are an incorrect identification of ν_{max}, or that **-autoguess** does not find the initial $l = 2, 0$ pair. A value of ν_{max} can be given to **-autoguess** in the command line call, as well as a list of frequencies. The l of these mode frequencies can also be given, but it is not stictly required.

3. Once **-autoguess** has run two files will appear in the star directory, *starID_initialguesses.txt* and *starID_output.npz*. For now we will only discuss the former. This is a plain text file containing a list of the modes that **-autoguess** found with columns containing: a 'pseudo-radial order' (starting at 0 at the lowest frequency mode), a guess at the l as well as frequency, peak height and width. The most important columns are the l and frequency columns.

   It is advisable that you check **initialguesses.txt* file after every run of **-autoguess**, to make sure that the mode parameters look vaguely correct. **-autoguess** will most likely not pick up all the modes in the spectrum, so you may wish to add these manually. This can of course be done with the command line call, but it is often more convenient to do this directly in the **initialguesses.txt* file.

   To check which modes are missing, run **-autoguess** with the **-check** flag. This will show a plot of the power spectrum and the modes currently in the **initialguesses.txt* list. Now simply add the missing modes on separate lines in the **initialguesses.txt* file. The mode height and width are not so important to get correct, just pick something vaguely in the ballpark. Note also that the modes don't have to be ordered in the list, they will be sorted automatically by frequency.

4. Now run **-autoguess** again with the **-refit** option. This will go through all the modes in **initialguesses.txt* and perform the whole mode identification rigmarole again, but with your inputs as override. **-refit** also updates the fit dictionary (see below).

2.2 Peakbagging

Once you have obtained a satisfactory list of modes to fit, the next step is running PME with the **-peakbagging** option:

```
python PME.py /folder/with/spectrum starID -peakbagging
                                            -variable1 my_parameterization1
                                            -variable2 my_parameterization2
                                            -mcmc_option1
                                            -mcmc_option2
                                            ...
```

-peakbagging takes the list of initial guesses from *starID_initialguesses.txt* for the modes and *starID_output.npz* for the background terms. These initial guesses are parameterized according to the scheme that is chosen for **-peakbagging**, an example could be running with -mode_splittings radial_step_profile. See –help for a complete list of options. If no other options are giving to PME, it will perform a short run of 20 steps, which can be used to see that everything runs properly, before launching a longer run.

Apart from the parameterizations a few extra options should be considered here concerning the MCMC part of the routine:

- **-mcmc_walkers XX**: sets the number of MCMC walkers (or samplers) in the fit. This should be at least twice the number of parameters (after the parameterization has been chosen), and preferably somewhat more than this. PME will warn you if the number of walkers is too low. Because PME uses the parallel ensemble afine invariant sampler from EMCEE, the number of walkers must be even. Again, PME will let you know if it disagrees with your choice of walkers.

- **-mcmc_steps XX**: the number of steps through parameterspace that the walkers should take. Currently I am not aware of any metric that will tell you exactly when the MCMC walkers will converge on the maximum likelihood. The choice of this number should be large (at least a few thousand), and the **-plot** option should be used regularly to check the condition of walkers (see below).

- **-mcmc_burnt**: This option can be used in the event that you chose a value of -mcmc_steps that was too small. When using this option, the fitting will pick up where the previous fit left off, and let the walkers continue on their merry way.

- **-mcmc_substeps**: saves the progress of the walkers for every chosen number of sub-steps. This option is useful for particularly long runs in case the computer crashes. It also allows you to view the current progress of the walkers using the **-plot** option.

2.3 Plot

After **-peakbagging** has run, or the first subsection of the run has been saved using the -substep option, you can use the **-plot** option to check the state of the fit:

```
python PME.py /folder/with/spectrum starID -plot
```

Note that there are no additional options for **-plot**. Be aware that **-plot** will display a large number of plots, so machine memory may be an issue. (an option to limit plot output will be implemented)

The point of this option is to provide diagnostic plots that will show if something is wrong in the fit, and potentially when the fit has converged. The vast majority of these are of the MCMC chains projected onto each axis of the parameter space. In addition to this, the likelihoods of all the walkers at each step is also plotted. This is the main indicator of convergence, as the likelihood will stop changing once the maximum likelihood region is reached.

It is important to stress that 'convergence' of an MCMC chain is a somewhat subjective term, in the sense that it is difficult (sometimes impossible) to know if the maximum likelihood region has been reached. However, a general rule of thumb is that if the median and the variance of the walker positions along each axis, as well as the likelihoods, have remained constant for several thousand steps, the fit has most likely converged.

In addition to the above, the best-fit model is also plotted, as well as an eschelle diagram, which will helps identify problems with the mode ID.

2.4 PME Output

The main output of PME is the *starID_output.npz* file. This is a zipped Python dictionary, which is created when using the **-autoguess** option. This dictionary contains all the information relevant to the fit (that we can think of), the power spectrum itself, the star, as well as the settings used when running **-peakbagging**. This should in principle allow you or someone else without complete knowledge of the current fitting run, to reconstruct all or parts of the fit.

Note that certain parts of the output file point to functions in *PME_FUNCTIONS.py* and some of the other *.py files. So make sure Python can see these files if you want to manually access the dictionary. Not sure how to suppress this...

It is possible to modify the fit on the fly by editing the dictionary, but this is for advanced users only, so best to not fiddle with it too much.

3 Additional Options and Tools

3.1 Simulation

This option is only for experimental purposes. It is run using:

```
python PME.py /folder/with/spectrum starID -simulation
                                           -additional_options
```

This option generates a random noise realization with a mean equivalent to the best-fit model from the **-peakbagging** option. Currently the only accepted options are the simulation ID, a chosen value of the rotational splitting parameter(s) and the chosen value of the stellar inclination angle.

3.2 Make power spectrum

The **-make_powerspectrum** option takes any given time series and computes the power spectrum. The power spectrum is computed by the Lomb-Scargle method [see, e.g., Frandsen et al., 1995] and so is rather slow for long, densely sampled time series (10^5 samples or more). It is parallelized so you should take advantage of this if possible. The benefit however is that you won't have to worry about whether your data is regularly sampled or not (don't forget about your window function though).

Note that the

3.3 Tools for Data analysis in Python

With this package comes PME_tools.py, which contains the class PME_LOAD for loading and handling the output from PME, and do some simple plot. This module is currently in development. The class can in principle load multiple PME output, thus allowing for straightforward comparisons.

Below some examples involving the use of all current methods implemented.

Import the module and load a PME fit output

Load the data from a fit of the spectrum contained in the file StarID.pow:

```
import PME_tools as PT

pme_out = PT.PME_LOAD(files = 'StarID_output.npz')
```

There are other ways of loading a PME output, using different keyword than files, however they require to be in the same folder as StarID_output.npz. Multiple PME outputs are loaded if a list/tuple of files is given. In that case the full file path may be specified and files do not need to be in the same folder.

The object `pme_out` contains the following methods/data

- `pme_out.fit_dict[]` : type NumberedDict. This is the main dictionary, that contains the output dictionaries from PME. The keys of the dictionary are the wild_cards of the fit(s), e.g., `pme_out.fit_dict['StarID']` contains the output dictionary from `StarID_output.npz`.
- `pme_out.PME_PLOT()` : method that make a series of plots: same as python PME.py ./ StarID -plot
- `pme_out.get_best_fit_model()` : extracts the best fit model together with the 16th and 84th percentiles for all the dictionaries and put the result in `pme_out.best_fit`.
- `pme_out.best_fit[]` : type NumberedDict. Dictionary, e.g., `pme_out.best_fit['StarID']` contains ['bestfit','16th','84th'] percentiles of the corresponding fit. Each key is a dictionary with the following structure: `pme_out.best_fit['StarID']['best_fit'][...]`
  - `['freq']` : type np.ndarray. Frequency array of the PSD.
  - `['fit_psd']` : type np.ndarray. Fitted model o PSD.
  - `['fit_params]` : type Dict. Dictionary with best fit values `['height', 'h_pow', 'freq', 'l', 'm', 'h_exp', 'n', 'width', 'h_freq', 'incl','w_noise']`, as obtained from the `unpack_parameters` function provided by PME.
- `pme_out.extract_params(['keys'])` : this function returns a NumberedDict of the desired parameters to extract given as a list of the keys contained in [...]['fit_params'] (see above). It is useful when dealing with more than one fit result, because it "brings together" the same parameters from different fits in one dictionary. Is Useless if only one fit is loaded.
- `pme_out.plot_best_fit()` : plot spectrum together with best fit. Several options implemented in case of multiple fits.
- `pme_out.plot_param_vs_freq(param='height', with_errors=True, palette='rainbow', subplots=True)` : plot the best value of the desired parameter vs. frequency for all the loaded fits, with 84th and 16th percentile errors. WARNING. The errors are calculated based on the 16th and 84th percentiles, e.g. using the errors in the coefficients of the polynomial fit of the width, therefore it is not the "1-sigma" error for the width of that mode.

4 Variable parameterizations in PME

One of the main goals of PME was to make it flexible, in the sense that it should be possible to swap out different ways of parameterizing variables in the peakbagging. The way PME handles variable parameterization is to initially cast the contents of **initialguesses.txt* into the chosen parameterization for each variable set.

Example: a set of 30 mode heights may be parameterized by a Guassian with only 3 variables, since we know that the mode heights of p-modes tend to take this shape as a function of frequency. Thus reducing the number of fit parameters dramatically. At the start of a peakbagging run, PME will then fit a Gaussian to the mode heights provided in **initialguesses.txt*, and use these fit parameters as initial guesses in the MCMC fit. Following this, PME then selects the appropriate so-called unpacking function, which translates the 3 parameters of the Gaussian back into the heights of the 30 modes. These 30 mode heights are then fed to the function which computes the model spectrum (Eq. 3).

4.1 Adding additional parametrizations

Adding new parameterizations is in principle very simple, but does require some knowledge of the Python programming language. It is important to keep in mind that the function that PME uses to calculate the spectrum is fixed (Eq. 3). This means that any parameterization you want to implement must eventually be translated into the variables in Eq. 3.

Furthermore, the variables in PME are separated into the sets (listed in section 1.2.2), and these sets are treated individually and in sequence[2]. This means that currently, you cannot use parameterizations that rely on variables that are first calculated later in the sequence (i.e., coupled equations).

4.2 Implemented Parametrizations

4.2.1 Mode linewidth parametrization

Currently,the following parametrizations for the mode linewidths are used: a 5th order polynomial in frequency and a combination of a power law with a lorentzian profile.

Appourchaux et al. parametrization: Following Appourchaux et al. [2014] we parametrize the mode linewidths as follow

$$\ln(\Gamma) = \alpha \ln(\nu/\nu_{\max}) + \ln \Gamma_\alpha - \ln(\Delta\Gamma_{\text{dip}}) \left[1 + \left(\frac{2\ln(\nu/\nu_{\text{dip}})}{\ln(W_{\text{dip}}/\nu_{\max})}\right)^2\right]^{-1}, \quad (4)$$

where ν is the mode frequency, $\nu_{\max}$ is the frequency of maximum power, α is the power law index, Γ_α is the factor multiplying the power law, $\Delta\Gamma_{\text{dip}}$ is the depth of the lorentzian dip, W_{dip} is the width of the dip and ν_{dip} is its frequency.

In the fit $\nu_{\max}$ is given by the preliminary fit (done in -autoguess mode). We now write the default limits and guesses of each parameters in the form [low limit, guess, up limit]

- $\alpha : [2 \cdot 10^{-19}, 4, 10]$
- $\Gamma_\alpha : [2 \cdot 10^{-19}, 5, 20]\mu\text{Hz}$
- $\Delta\Gamma_{\text{dip}} : [2 \cdot 10^{-19}, 3, 20]\mu\text{Hz}$
- $\nu_{\text{dip}} : [\min(\nu_{nlm}), \nu_{\max}, \max(\nu_{nlm})]\mu\text{Hz}$
- $W_{\text{dip}} : [\nu_{\max}, (\nu_{\max} + \max(\nu_{nlm}))/2, 10000.]\mu\text{Hz},$

where ν_{nlm} are the initial guesses of the mode frequencies. Those limits have been set based on the fit results over 22 kepler stars of G and F type from table 3 of Appourchaux et al. [2014]. The fit is done on the logarithmic quantities of the parameters, except for α and ν_{dip}.

Bibliography

E. R. Anderson, T. L. Duvall, Jr., and S. M. Jefferies. Modeling of solar oscillation power spectra. *ApJ*, 364:699–705, December 1990. doi: 10.1086/169452.

T. Appourchaux, H. M. Antia, O. Benomar, T. L. Campante, G. R. Davies, R. Handberg, R. Howe, C. Régulo, K. Belkacem, G. Houdek, R. A. García, and W. J. Chaplin. Oscillation mode linewidths and heights of 23 main-sequence stars observed by Kepler. *A&A*, 566:A20, June 2014. doi: 10.1051/0004-6361/201323317.

T. L. Duvall, Jr. and J. W. Harvey. Solar Doppler shifts: Sources of continuous spectra. In D. O. Gough, editor, *NATO Advanced Science Institutes (ASI) Series C*, volume 169 of *NATO Advanced Science Institutes (ASI) Series C*, pages 105–116, 1986.

[2]This will eventually be changed

D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. emcee: The MCMC Hammer. *PASP*, 125:306–312, March 2013. doi: 10.1086/670067.

S. Frandsen, A. Jones, H. Kjeldsen, M. Viskum, J. Hjorth, N. H. Andersen, and B. Thomsen. CCD photometry of the δ-Scuti star κˆ2ˆ Bootis. *A&A*, 301:123, September 1995.

L. Gizon and S. K. Solanki. Determining the Inclination of the Rotation Axis of a Sun-like Star. *ApJ*, 589:1009–1019, June 2003. doi: 10.1086/374715.

J. W. Harvey, F. Hill, J. R. Kennedy, J. W. Leibacher, and W. C. Livingston. The Global Oscillation Network Group (GONG). *Advances in Space Research*, 8:117–120, 1988. doi: 10.1016/0273-1177(88)90304-3.

M. F. Woodard. *Short-Period Oscillations in the Total Solar Irradiance.* PhD thesis, University of California, San Diego., 1984.