1 Overview of SQUAREM

"SQUAREM" is a package intended for accelerating slowly-convergent contraction mappings. It can be used for accelerating the convergence of slow, linearly convergent contraction mappings such as the EM (expectation-maximization) algorithm, MM (majorize and minimize) algorithm, and other nonlinear fixed-point iterations such as the power method for finding the dominant eigenvector. It uses a novel approach callled squared extrapolation method (SQUAREM) that was proposed in Varadhan and Roland (Scandinavian Journal of Statistics, 35: 335-353), and also in Roland, Vardhan, and Frangakis (Numerical Algorithms, 44: 159-172).

The functions in this package are made available with:

```
> library("SQUAREM")
```

You can look at the basic information on the package, including all the available functions with

```
> help(package=SQUAREM)
```

The package *setRNG* is not necessary, but if you want to exactly reproduce the examples in this guide then do this:

```
> require("setRNG")
> setRNG(list(kind="Wichmann-Hill", normal.kind="Box-Muller", seed=123))
```

after which the example need to be run in the order here (or at least the parts that generate random numbers). For some examples the RNG is reset again so they can be reproduced more easily.

2 How to accelerate convergence of a fixed-point iteration with SQUAREM?

2.1 Accelerating EM algorithm: Binary Poisson Mixture Maximum-Likelihood Estimation

Now, we show an example demonstrating the ability of SQUAREM to dramatically speed-up the convergence of the EM algorithm for a binary Poisson mixture estimation. We use the example from Hasselblad (J of Amer Stat Assoc 1969)

```
> poissmix.dat <- data.frame(death=0:9, freq=c(162,267,271,185,111,61,27,8,3,1))
```

Generate a random initial guess for 3 parameters

```
> y <- poissmix.dat$freq
> tol <- 1.e-08
> setRNG(list(kind="Wichmann-Hill", normal.kind="Box-Muller", seed=123))
> p0 <- c(runif(1),runif(2,0,6))
```

The fixed point mapping giving a single E and M step of the EM algorithm

```
> poissmix.em <- function(p,y) {
  pnew <- rep(NA,3)
  i <- 0:(length(y)-1)
  zi <- p[1]*exp(-p[2])*p[2]^i / (p[1]*exp(-p[2])*p[2]^i + (1 - p[1])*exp(-p[3])*p[3]^i)
  pnew[1] <- sum(y*zi)/sum(y)
  pnew[2] <- sum(y*i*zi)/sum(y*zi)
  pnew[3] <- sum(y*i*(1-zi))/sum(y*(1-zi))
  p <- pnew
  return(pnew)
  }
```

Objective function whose local minimum is a fixed point negative log-likelihood of binary poisson mixture.

```
> poissmix.loglik <- function(p,y) {
  i <- 0:(length(y)-1)
  loglik <- y*log(p[1]*exp(-p[2])*p[2]^i/exp(lgamma(i+1)) +
                  (1 - p[1])*exp(-p[3])*p[3]^i/exp(lgamma(i+1)))
  return ( -sum(loglik) )
  }
```

EM algorithm

```
> pf1 <- fpiter(p=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik,
  control=list(tol=tol))
> pf1

$par
[1] 0.6401136 2.6634056 1.2560968

$value.objfn
[1] 1989.946

$fpevals
[1] 2909

$objfevals
[1] 0

$convergence
[1] TRUE
```

Note the slow convergence of EM, as it uses more than 2900 iterations to converge. Now, let us speed up the convergence with SQUAREM:

```
> pf2 <- squarem(p=p0, y=y, fixptfn=poissmix.em, objfn=poissmix.loglik,
  control=list(tol=tol))
> pf2

$par
[1] 0.6401146 2.6634044 1.2560951

$value.objfn
[1] 1989.946

$iter
[1] 25

$fpevals
[1] 72

$objfevals
[1] 25

$convergence
[1] TRUE
```

Note the dramatically faster convergence, i.e. SQUAREM uses only 72 fixed-point evaluations to achieve convergence. This is a speed up of a factor of 40.

We can also run SQUAREM without specifying an objective function, i.e. the log-likelihood. *This is usually the most efficient way to use SQUAREM.*

```
> pf3 <- squarem(p=p0, y=y, fixptfn=poissmix.em, control=list(tol=tol))
> pf3

$par
[1] 0.6401146 2.6634044 1.2560951

$value.objfn
[1] NA

$iter
[1] 25

$fpevals
[1] 72

$objfevals
[1] 0

$convergence
[1] TRUE
```

2.2 Accelerating the Power Method for Finding the Dominant Eigenpair

The power method is a nonlinear fixed-point iteration for determining the dominant eigenpair, i.e. the largest eigenvalue (in terms of absolute magnitude) and the corresponding eigenvector, of a square matrix A. The iteration can be programmed in R as:

```
> power.method <- function(x, A) {
  # Defines one iteration of the power method
  # x = starting guess for dominant eigenvector
  # A = a square matrix
  ax <- as.numeric(A %*% x)
  f <- ax / sqrt(as.numeric(crossprod(ax)))
  f
  }
```

We illustrate this for finding the dominant eigenvector of the Bodewig matrix which is a famous matrix for which power method has trouble converging. See, for example, Sidi, Ford, and Smith (SIAM Review, 1988). Here there are two eigenvalues that are equally dominant, but have opposite signs! Sometimes the power method finds the eigenvector corresponding to the large positive eigenvalue, but other times it finds the eigenvector corresponding to the large negative eigenvalue

```
> b <- c(2, 1, 3, 4, 1,  -3,   1,   5,  3,   1,   6,  -2,  4,   5,  -2,  -1)
> bodewig.mat <- matrix(b,4,4)
> eigen(bodewig.mat)

$values
[1]  7.932905  5.668864 -1.573191 -8.028578

$vectors
           [,1]       [,2]       [,3]       [,4]
[1,] -0.5601445  0.3787027  0.6880479  0.2634624
[2,] -0.2116328  0.3624190 -0.6241229  0.6590407
[3,] -0.7767083 -0.5379352 -0.2598009 -0.1996335
[4,] -0.1953816  0.6601988 -0.2637503 -0.6755734
```

Now, let us look at power method and its acceleration using various SQUAREM schemes:

```
> p0 <- rnorm(4)
> # Standard power method iteration
> ans1 <- fpiter(p0, fixptfn=power.method, A=bodewig.mat)
> # re-scaling the eigenvector so that it has unit length
> ans1$par <- ans1$par / sqrt(sum(ans1$par^2))
> # dominant eigenvector
> ans1
```

```
$par
[1]  0.2634624  0.6590407 -0.1996335 -0.6755734

$value.objfn
[1] NA

$fpevals
[1] 5000

$objfevals
[1] 0

$convergence
[1] FALSE

> # dominant eigenvalue
> c(t(ans1$par) %*% bodewig.mat %*% ans1$par) / c(crossprod(ans1$par))

[1] -8.028578
```

Now, we try first-order SQUAREM with default settings:

```
> ans2 <- squarem(p0, fixptfn=power.method, A=bodewig.mat)
> ans2

$par
[1] -0.2634624 -0.6590407  0.1996335  0.6755734

$value.objfn
[1] NA

$iter
[1] 751

$fpevals
[1] 1500

$objfevals
[1] 0

$convergence
[1] TRUE

> ans2$par <- ans2$par / sqrt(sum(ans2$par^2))
> c(t(ans2$par) %*% bodewig.mat %*% ans2$par) / c(crossprod(ans2$par))

[1] -8.028578
```

The convergence is still slow, but it converges to the dominant eigenvector. Now, we try with a minimum steplength that is smaller than 1.

```
> ans3 <- squarem(p0, fixptfn=power.method, A=bodewig.mat, control=list(step.min0 = 0.5))
> ans3

$par
[1] 0.5601445 0.2116328 0.7767083 0.1953816

$value.objfn
[1] NA

$iter
[1] 10

$fpevals
[1] 27

$objfevals
[1] 0

$convergence
[1] TRUE

> ans3$par <- ans3$par / sqrt(sum(ans3$par^2))
> # eigenvalue
> c(t(ans3$par) %*% bodewig.mat %*% ans3$par) / c(crossprod(ans3$par))

[1] 7.932905
```

The convergence is dramatically faster now, but it converges to the second dominant eigenvector. We try again with a higher-order SQUAREM scheme.

```
> # Third-order SQUAREM
> ans4 <- squarem(p0, fixptfn=power.method, A=bodewig.mat, control=list(K=3, method="rre"))
> ans4

$par
[1] 0.5601445 0.2116328 0.7767083 0.1953816

$value.objfn
[1] NA

$iter
[1] 5

$fpevals
```

```
[1] 29

$objfevals
[1] 0

$convergence
[1] TRUE

> ans4$par <- ans4$par / sqrt(sum(ans4$par^2))
> # eigenvalue
> c(t(ans4$par) %*% bodewig.mat %*% ans4$par) / c(crossprod(ans4$par))

[1] 7.932905
```

Once again we obtain the second dominant eigenvector.