Parallel computations with R

Søren Højsgaard

Aalborg University, Denmark

October 15, 2012

Printed: October 15, 2012 File: parallel-slides.tex

Contents

1 Introduction

- For some years now, computers have not become much faster (it seems that in practice there is an upper limit of about 3 GHz).
- Instead, most (all?) computers today come with multiple cores and hence the ability to make parallel computations.
- There are various `R` packages for parallel computations. The **parallel** package is shipped with the `R` distribution and this is the package we illustrate here.

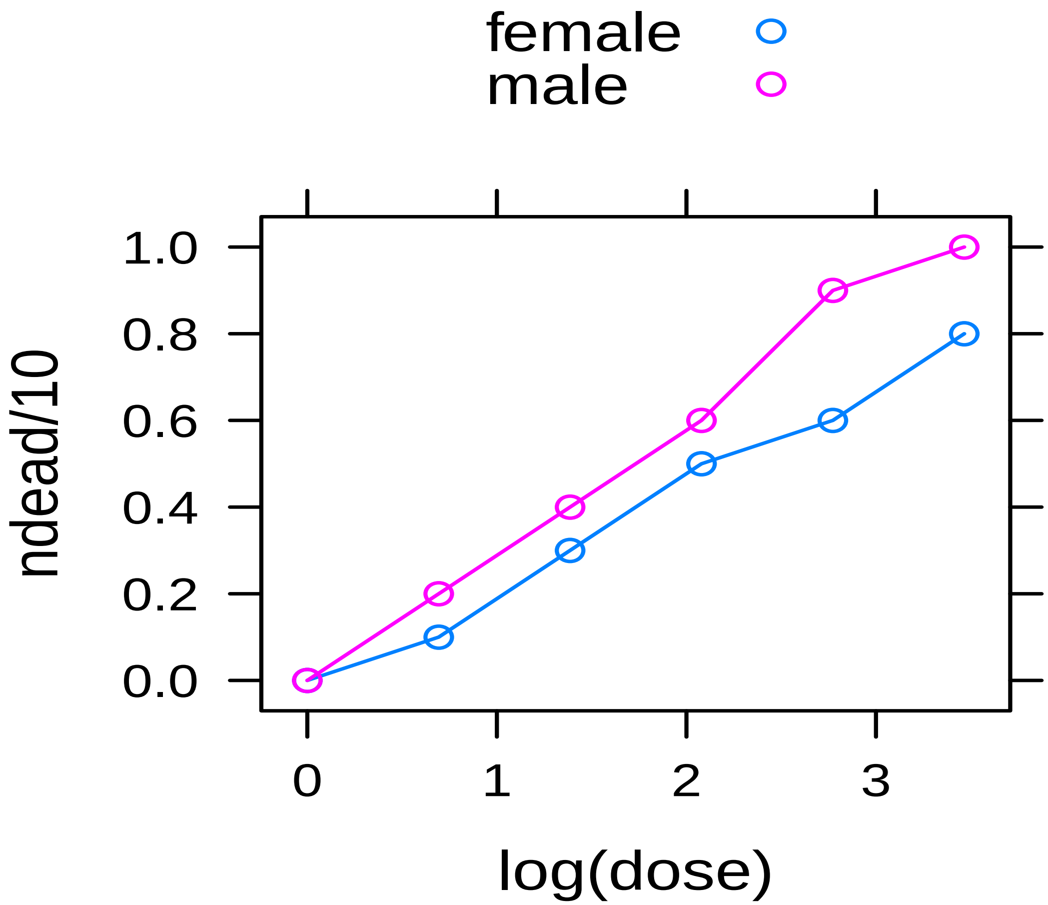
2 Parametric bootstrap

2.1 A logistic regression

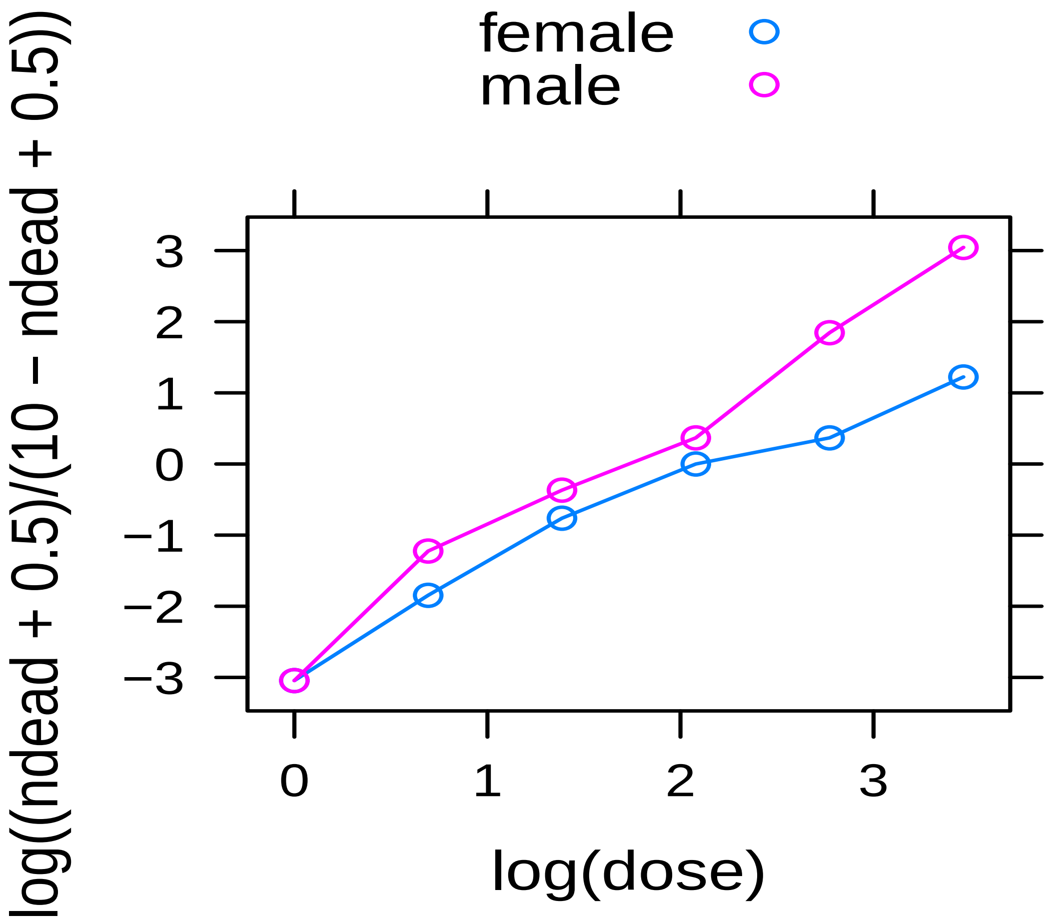
```
R> budworm

      sex dose ndead ntotal
1    male    1     0     10
2    male    2     2     10
3    male    4     4     10
4    male    8     6     10
5    male   16     9     10
6    male   32    10     10
7  female    1     0     10
8  female    2     1     10
9  female    4     3     10
10 female    8     5     10
11 female   16     6     10
12 female   32     8     10
```

```
R> library(lattice)
R> print(xyplot(ndead/10~log(dose), groups=sex, data=budworm,
+                 type="b", auto.key=T))
```



```
R> print(xyplot(log((ndead+.5)/(10-ndead+.5))~log(dose), groups=sex, data=budworm,
+                type="b", auto.key=T))
```



Very vanilla logistic regression. Is there an effect of sex?

```
R> budworm <- transform(budworm, logdose=log(dose))
R> mm1  <- glm(cbind(ndead,ntotal-ndead)~sex+logdose,
+           data=budworm, family=binomial(link=logit))
R> mm0  <- update(mm1, .~. - sex)
R> anova(mm1, mm0, test="Chisq")

Analysis of Deviance Table


Model 1: cbind(ndead, ntotal - ndead) ~ sex + logdose
Model 2: cbind(ndead, ntotal - ndead) ~ logdose
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1         9     5.0466
2        10     8.7883 -1  -3.7417  0.05307 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2.2 Parametric bootstrap

Alternative to $-2\log Q$ test: Parametric bootstrap – seems attractive when large sample asymptotics is questionable:

- Compare two models M_1 and M_0 where $M_0 \subset M_1$.
- Fit both models to data; Gives $\hat{\theta}_1$, $\hat{\theta}_0$ and t_{obs} (test statistic)
- Draw B parametric bootstrap sample datasets $y^1, \dots y^B$ from $p_0(\cdot|\hat{\theta}_0)$. For each simulated dataset y^b, calculate test statistic t^b.
- Evaluate how extreme t_{obs} is in $\{t^1, \dots t^B\}$.

2.3 Non–parallel version

Need to refit `glm` to new response variable:

```
R> refit_glm <- function(mm, yy){
+    ff <- update.formula(formula(mm),yy ~ .)
+    ee <- local({ yy=yy; environment()})
+    environment(ff) <- ee
+    cl <- getCall(mm)
+    cl$formula <- ff
+    eval(cl)
+  }
```

Need to do so many times and calculate reference distribution:

```
R> pboot <- function(lg, sm, nsim=10){
+    simdata <- simulate(sm, nsim)
+    ref <- rep.int(NA,nsim)
+    for (ii in 1:nsim){
+      y.new <- simdata[,ii]
+      ref[ii] <- 2*(logLik(refit_glm(lg, y.new))-logLik(refit_glm(sm, y.new)))
+    }
+    ref
+  }
```

```
R> anova(mm1, mm0, test="Chisq")

Analysis of Deviance Table

Model 1: cbind(ndead, ntotal - ndead) ~ sex + logdose
Model 2: cbind(ndead, ntotal - ndead) ~ logdose
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1         9     5.0466
2        10     8.7883 -1  -3.7417  0.05307 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R> set.seed(123)
R> Nsim <- 2000
R> (obsdev <- c(2*(logLik(mm1)-logLik(mm0))))

[1] 3.741666

R> system.time({   refdev <- pboot(mm1, mm0, nsim=Nsim) })

   user  system elapsed
  10.51    0.02   10.55

R> sum(refdev > obsdev) / length(refdev)

[1] 0.0505
```

2.4 Parallel version

To do parallel computations, first detect the number of cores available. Next create a handle on the clusters:

```
R> library("parallel")
R> ## Number of cores:
R> (nc <- detectCores())

[1] 8

R> ## Create clusters
R> cl <- makeCluster(rep("localhost", nc))
```

When done with the parallel computations, close the clusters before leaving R:

```
R> ## Remember to shut down clusters before quitting R
R> stopCluster(cl)
```

```
R> t0 <- proc.time()
R> Nsim <- 2000
R> (nsim2 <- round(Nsim / nc))

[1] 250

R> ## Export global environment to each cluster
R> clusterExport(cl, ls(envir=.GlobalEnv), envir = .GlobalEnv)
R> ## Spec for random number generator
R> clusterSetRNGStream(cl)
R> ## Time to get things up and running
R> proc.time()-t0

   user  system elapsed
   0.02    0.03    0.05
```

```
R> t0 <- proc.time()
R> ## Do the simulations
R> xxx    <- clusterCall(cl, pboot, mm1, mm0, nsim2)
R> refdev <- c(xxx, recursive=TRUE)
R> ## Time to do the simulations
R> proc.time()-t0

   user  system elapsed
   0.01    0.00    2.43

R> (p.PB <- sum(refdev > obsdev) / length(refdev))

[1] 0.0585
```

Notice:

- We get a substantial saving in computing time, but not by a factor equal to the number of cores.
- There is some overhead in setting things up for parallel computations.

3 Matrix multiplication

Multiplication of a $p \times q$ and a $q \times r$ matrix has complexity of the order pqr.

```
R> nr <- 3
R> (A <- matrix(round(rnorm(nr^2),1),nr=nr))

     [,1] [,2] [,3]
[1,] -0.2  0.4 -1.6
[2,]  0.8  0.2  0.5
[3,] -1.2  1.0  0.3

R> (B <- t(A) + 4)

     [,1] [,2] [,3]
[1,]  3.8  4.8  2.8
[2,]  4.4  4.2  5.0
[3,]  2.4  4.5  4.3

R> A %*% B

      [,1]  [,2]  [,3]
[1,] -2.84 -6.48 -5.44
[2,]  5.12  6.93  5.39
[3,]  0.56 -0.21  2.93
```

Idea: Split A by rows, do multiplications and stack the results:

```
R> A[1,,drop=FALSE] %*% B

      [,1]  [,2]  [,3]
[1,] -2.84 -6.48 -5.44

R> A[2,,drop=FALSE] %*% B

     [,1] [,2] [,3]
[1,] 5.12 6.93 5.39

R> A[3,,drop=FALSE] %*% B

     [,1]  [,2] [,3]
[1,] 0.56 -0.21 2.93

R> rbind(
+  A[1,,drop=FALSE] %*% B,
+  A[2,,drop=FALSE] %*% B,
+  A[3,,drop=FALSE] %*% B )

      [,1]  [,2]  [,3]
[1,] -2.84 -6.48 -5.44
[2,]  5.12  6.93  5.39
[3,]  0.56 -0.21  2.93
```

```
R> ## Use 2 clusters
R> cl2 <- cl[1:2]
R> (idx   <- splitIndices(nrow(A), length(cl2)))

[[1]]
[1] 1


[[2]]
[1] 2 3

R> (Alist <- lapply(idx, function(ii) A[ii,,drop=FALSE]))

[[1]]
     [,1] [,2] [,3]
[1,] -0.2  0.4 -1.6


[[2]]
     [,1] [,2] [,3]
[1,]  0.8  0.2  0.5
[2,] -1.2  1.0  0.3
```

```
R> ans   <- clusterApply(cl2, Alist, function(aa, BB) aa %*% BB, B)
R> do.call(rbind, ans)

      [,1]  [,2]  [,3]
[1,] -2.84 -6.48 -5.44
[2,]  5.12  6.93  5.39
[3,]  0.56 -0.21  2.93

R> A %*% B

      [,1]  [,2]  [,3]
[1,] -2.84 -6.48 -5.44
[2,]  5.12  6.93  5.39
[3,]  0.56 -0.21  2.93
```

3.1 Parallel version

Define parallel matrix multiplication function:

```
R> matprod.par <- function(cl, A, B){
+    if (ncol(A) != nrow(B)) stop("Matrices do not conforme")
+    idx   <- splitIndices(nrow(A), length(cl))
+    Alist <- lapply(idx, function(ii) A[ii,,drop=FALSE])
+    ## ans   <- clusterApply(cl, Alist, function(aa, B) aa %*% B, B)
+    ## Same as above, but faster:
+    ans   <- clusterApply(cl, Alist, get("%*%"), B)
+    do.call(rbind, ans)
+  }
```

Notice:

```
R> get("%*%")

function (x, y)  .Primitive("%*%")
```

```
R> nr <- 5
R> A <- matrix(round(rnorm(nr^2),1),nr=nr)
R> B <- t(A) + 4
R> matprod.par(cl, A, B)

      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] -1.65 -2.09 -6.30 -3.94 -5.51
[2,] -0.49  6.05 -1.91 -3.04 -1.22
[3,]  4.90  7.69 12.41  5.21 10.40
[4,] 13.26 12.56 11.21 17.60 12.50
[5,]  2.09  4.78  6.80  2.90  6.30

R> A %*% B

      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] -1.65 -2.09 -6.30 -3.94 -5.51
[2,] -0.49  6.05 -1.91 -3.04 -1.22
[3,]  4.90  7.69 12.41  5.21 10.40
[4,] 13.26 12.56 11.21 17.60 12.50
[5,]  2.09  4.78  6.80  2.90  6.30
```

4 The heat equation

For a function $u(x, y, t)$ of spatial variables (x, y) and time variable t, the heat equation is

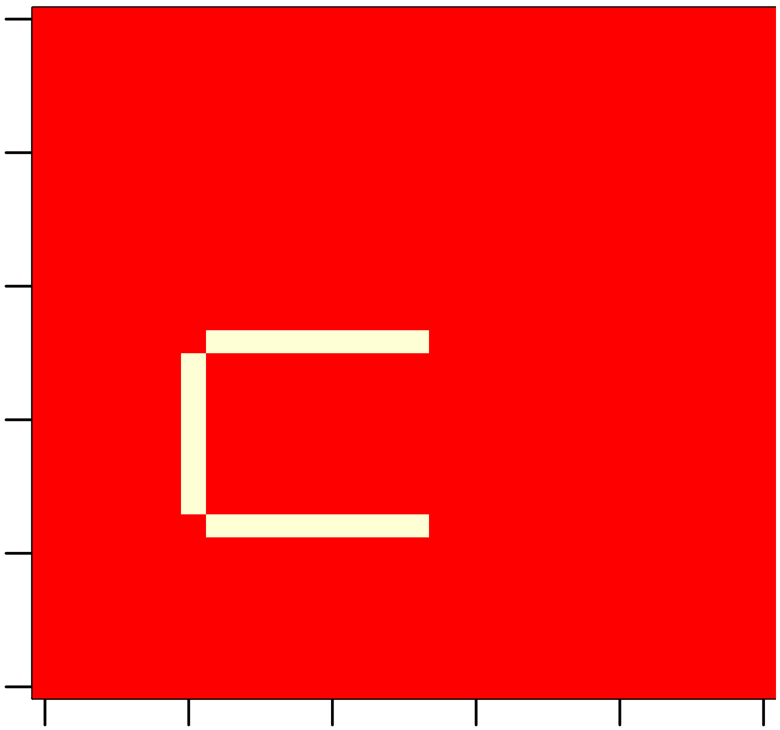
$$\frac{\partial u}{\partial t} - \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0, \quad \alpha > 0$$

This equation describes how the temperature $u(x, y, t)$ at location (x, y) changes over time t as heat spreads in space.

To solve this equation we must speficify the initial conditions $u(x, y, 0)$ and a set of boundary conditions.

Suppose we throw a heated horse shoe (with temperature 1) on a cold plate (with temperature 0):

```
R> nr <- nc <- 30
R> mm <- matrix(0,nrow=nr, ncol=nc)
R> mm[7,9:15] <- mm[8:16,16] <- mm[8:16,8] <- 1
R> image(mm)
```



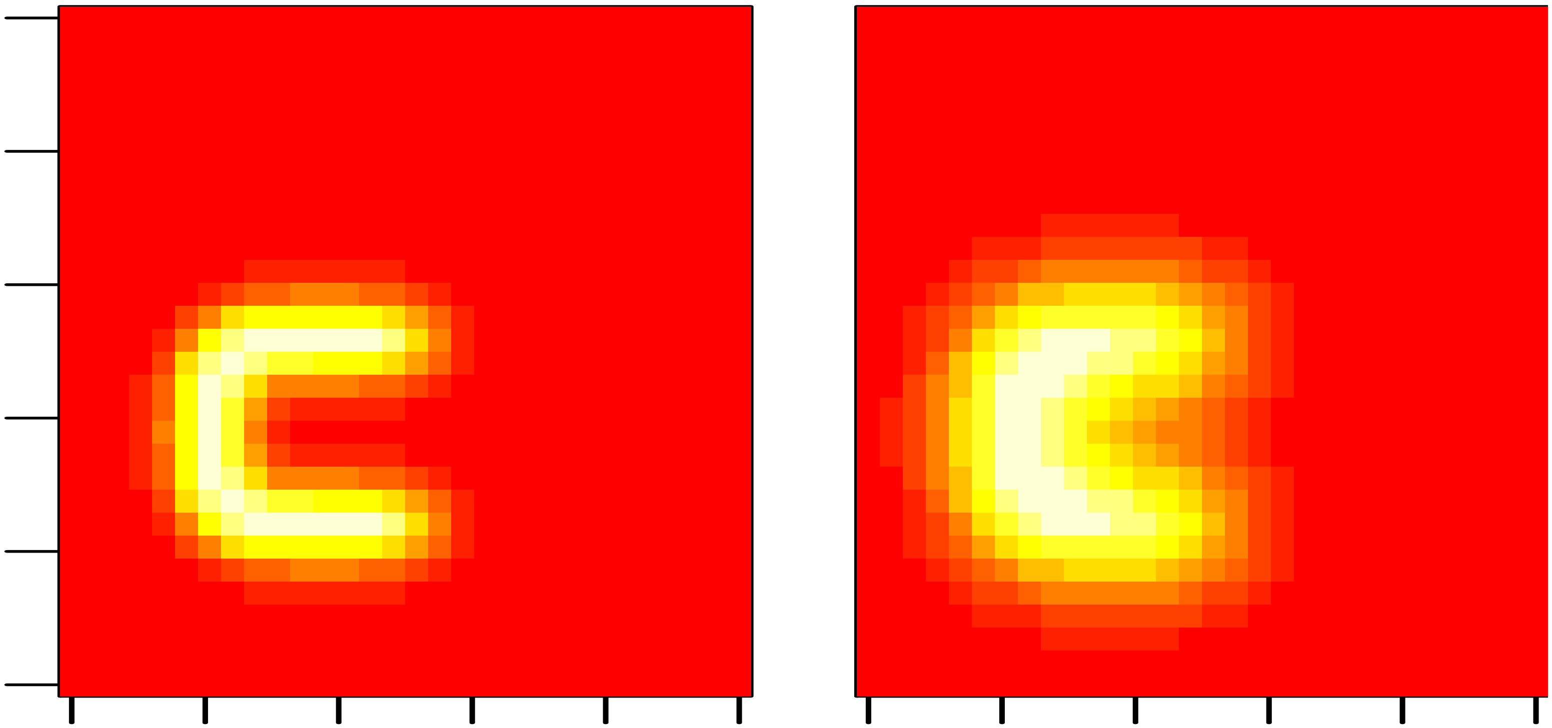
The temperature at the boundaries (outermost rows and columns) is assumed to remain constantly equal to zero. (Called a Dirichlet condition).

On a regular grid, this equation can be solved numerically as

$$\begin{aligned} u(x,y,t+1) &= u(x,y,t)+ \\ &\quad \alpha(u(x+1,y,t)+u(x-1,y,t)-2u(x,y,t))+ \\ &\quad \alpha(u(x,y+1,t)+u(x,y-1,t)-2u(x,y,t)) \end{aligned}$$

```
R> ## Update in one time step
R> heat <- function(mm, cx=.1, cy=.1, nr=nrow(mm), nc=ncol(mm)){
+    mm2 <- mm
+    for (ii in 2:(nr-1)){
+      for (jj in 2:(nc-1)){
+        mm2[ii,jj] <- mm[ii,jj] +
+          cy * (mm[ii-1,jj]+mm[ii+1,jj]-2*mm[ii,jj]) +
+            cx * (mm[ii,jj-1]+mm[ii,jj+1]-2*mm[ii,jj])
+      }
+    }
+    mm2
+  }
R> library(compiler) ## Byte compiling gives a factor 5 in speed
R> heat.cmp <- cmpfun(heat)
R> ## Iterate over time
R> heat.iter <- function(mm, n=1, cx=.1, cy=.1){
+    for (ii in 1:n){
+      mm <- heat.cmp(mm, cx, cy)
+    }
+    mm
+  }
```

```
R> mm.10 <- heat.iter(mm,n=10)
R> mm.30 <- heat.iter(mm.10,n=20)
R> image(mm.10)
R> image(mm.30,yaxt='n')
```



4.1 Numeric estimation of second derivatives

Suppose that we want to estimate the second derivative of a function f at a point x and that we know the values of the function at three points

$$(x-h, f(x-h)), \quad (x, f(x)), \quad (x+h, f(x+h))$$

Since $f''(x)$ is the derivative of the function f' at the point x we can estimate $f''(x)$ by

$$f''(x) \approx \frac{f'\left(x+\frac{h}{2}\right) - f'\left(x-\frac{h}{2}\right)}{h}$$

We can estimate $f'(x-h/2)$ and $f'(x+h/2)$ by

$$\begin{aligned} f'(x - \frac{h}{2}) &\approx \frac{f(x) - f(x-h)}{h} \\ f'(x + \frac{h}{2}) &\approx \frac{f(x+h) - f(x)}{h} \end{aligned}$$

Putting this together gives

$$
\begin{aligned}
f''(x) &\approx \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} \\
&= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}
\end{aligned}
$$

Partial derivatives can be estimated the same way:

$$
\begin{aligned}
\frac{\partial^2}{\partial x^2} f(x,y) &\approx \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2} \\
\frac{\partial^2}{\partial y^2} f(x,y) &\approx \frac{f(x,y+h) - 2f(x,y) + f(x,y-h)}{h^2}
\end{aligned}
$$

5 EXERCISES: The heat equation

The update over time for the heat equation is an obvious candidate for parallel computing:

1. Divide the grid into regions – for example by splitting by the rows.
2. Update the regions.
3. Put the updated regions together and make the necessary modifications on the boundary between the regions.

Exercise:

1. Implement a parallel version of the algorithm for updating the heat equation.