Introduction

RSMs and Computer Experiments

Robert B. Gramacy (rbg@vt.edu : http://bobby.gramacy.com)
Department of Statistics, Virginia Tech

Plan

- "Classical" RSMs, but only as a jumping-off point.
- The interplay between mathematical models, numerical approximation, simulation, computer experiments, and (field) data.
- Gaussian process (GP) spatial models, emphasizing
  - surrogate computer modeling,
  - sequential design, Bayesian optimization,
  - calibration,
  - variable selection and sensitivity analysis, and more.
- Uncertainty quantification, where statistics ought to monopolize but sometimes doesn't.
- Machine learning methods:
  - "big-n" GP solutions (sparsity), non-stationary GP modeling, the frontier …

"Classical" RSM overview

RSM

Response surface methodology (RSM) is a collection of statistical and mathematical techniques for developing, improving, and optimizing processes.

Applications historically come from industry and manufacturing, focused on

- design, development, and formulation of new products,
- and the improvement of existing products,

but also from (national) laboratory research, and with obvious military application.

The over-arching theme is a study of how

- **input variables** controlling a product or process potentially influence a
- **response** measuring performance or quality characteristics.

Terminology

Consider the relationship between the

- response variable yield (y) in a chemical process
- and the two process variables reaction time (ξ_1) and reaction temperature (ξ_2)

```
yield <- function(xi1, xi2)
  {
    xi1 <- 3*xi1 - 15
    xi2 <- xi2/50 - 13
    xi1 <- cos(0.5)*xi1 - sin(0.5)*xi2
    xi2 <- sin(0.5)*xi1 + cos(0.5)*xi2
    y <- exp(-xi1^2/80 - 0.5*(xi2 + 0.03*xi1^2 - 40*0.03)^2)
    return(100*y)
  }
```

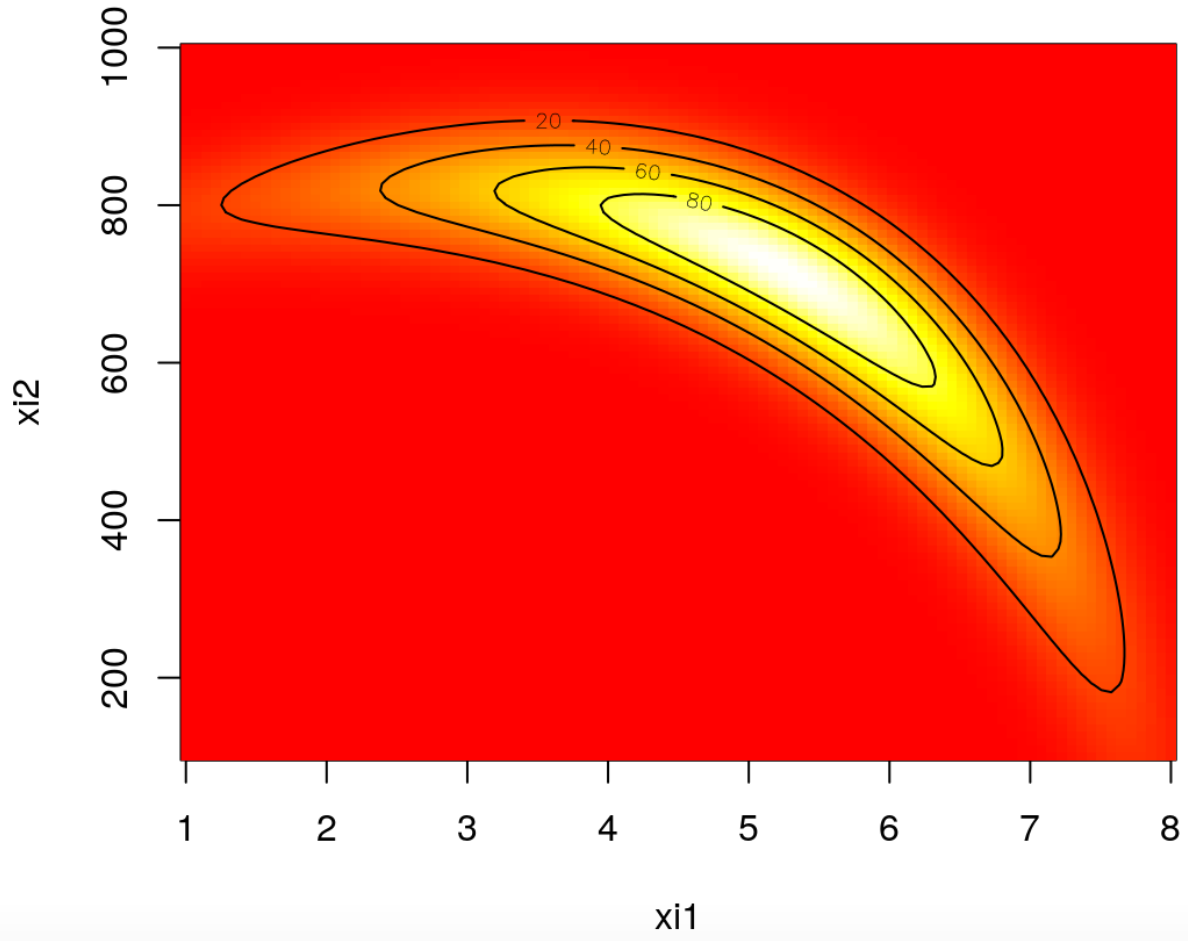
- This toy example is really a variation on the infamous "banana function".

Here, the yield response is plotted as a surface above the time/temperature plane.

```
xi1 <- seq(1, 8, length=100); xi2 <- seq(100, 1000, length=100)
g <- expand.grid(xi1, xi2); y <- yield(g[,1], g[,2])
persp(xi1, xi2, matrix(y, ncol=length(xi2)), theta=45, phi=45, lwd=0.5,
    xlab="xi1 : time", ylab="xi2 : temperature", zlab="yield", expand=0.4)
```

By inspection, the yield response is optimized near $(\xi_1, \xi_2) = (5 \text{ hr}, 750^\circ\text{C})$

```
image(xi1, xi2, matrix(y, ncol=length(xi2)), col=heat.colors(128))
contour(xi1, xi2, matrix(y, ncol=length(xi2)), nlevels=4, add=TRUE)
```



Easier said than done

Unfortunately, in practice, the true response surface is unknown.

It is too expensive to evaluate `yield` over a dense grid, because

- re-configuring the inputs may involve restarting an intricate (manufacturing) process,
- or might require an upgrade of equipment,
- or be otherwise inconvenient.

Measuring `yield` may be a noisy/inexact process.

That's where stats comes in

RSMs consist of the experimental strategies for

- statistically modeling the relationship between the response (yield) and process variables;
- paired with optimization/sequential design methods for finding the levels or values of the process variables that produce desirable values of the responses
    - (e.g., that maximize yield or explain variation).

The setup:

- Fit an **empirical model** (usually first or second-order linear) to observed data from the process or system *nearby the current regime* using a carefully designed experiment.
- Gradients and Hessians of the predictive equations yield the method of **steepest ascent** and **ridge analysis**.

Laborious process

It is a very "careful" enterprise.

- It requires statistical and design expertise, making automation difficult.
- And is at best informal about how to leverage domain specific (physical) knowledge.

These days folks rarely study industrial or physical processes solely via "on-the-bench"/field experiments.

- They are getting more more out of their statistical models, designs and optimizations, by coupling with **mathematical models** of the system(s) they are studying.

Often mathematical models are all there is.

- It can be too expensive or even unethical to gather data on certain phenomena.

Mathematical models

Simple equations seldom make for adequate descriptions of real-world systems.

- Physicists figured that out fifty years ago; industrial engineers followed suit.
- Biologists, social scientists, climate scientists, are coming on board.

Systems of equations are required, perhaps solved over meshes

- e.g., a so-called **finite-element** analysis.

Or you might have a big agent/individual-based model that governs how

- predictor and prey (randomly) interact with each other and their habitat;
- an epidemic spreads through a population, person by person;
- citizens make choices about health care and insurance.

Cheaper experimentation

Mathematical/computer models allow a cheaper means of

- exploring a system before embarking on a field experiment, or before the next one;
- screening variables and assessing main effects/sensitivity;
- optimizing (maximizing yield, say) or otherwise searching the input space in an automated way (with a wrapper around the simulation code).

And they can be studied in isolation, or coupled with field data experiments:

- tuning or calibrating the computer model, and/or
- leveraging the computer model to predict what would be observed in the field.

An aircraft wing weight example

The following equation has been used to help understand the weight of an unpainted light aircraft wing as a function of design and operational parameters.

$$W = 0.0365 S_{\mathrm{w}}^{0.758} W_{\mathrm{fw}}^{0.0035} \left(\frac{A}{\cos^2 \Lambda} \right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100 R_{\mathrm{tc}}}{\cos \Lambda} \right)^{-0.3} (N_{\mathrm{z}} W_{\mathrm{dg}})^{0.49}$$

- It is not really a computer simulation, but it will stand in for one in this example.
- It was derived by "calibrating" known physical relationships to curves obtained from existing aircraft data.

The next slide shows

- reasonable ranges for these **natural variables** (ξ),
- and a baseline setting coming from a Sessna C172 Skyhawk aircraft.

Symbol	Parameter	Baseline	Minimum	Maximum
S_w	Wing area (ft^2)	174	150	200
W_{fw}	Weight of fuel in wing (lb)	252	220	300
A	Aspect ratios	7.52	6	10
Λ	Quarter-chord sweep (deg)	0	-10	10
q	Dynamic pressure at cruise (lb/ft^2)	34	16	45
λ	Taper ratio	0.672	0.5	1
R_{tc}	Aerofoil thickness to chord ratio	0.12	0.08	0.18
N_z	Ultimate load factor	3.8	2.5	6
W_{dg}	Flight design gross weight (lb)	2000	1700	2500

Computer code

In **coded variables** ($x \in [0,1]^9$), with baseline as the default.

```
wingwt <- function(Sw=0.48, Wfw=0.28, A=0.38, L=0.5, q=0.62, l=0.344, Rtc=0.4,
    Nz=0.37, Wdg=0.38)
  {
    ## put coded inputs back on natural scale
    Sw <- Sw*(200 - 150) + 150
    Wfw <- Wfw*(300 - 220) + 220
    A <- A*(10 - 6) + 6
    L <- (L*(10 - (-10)) - 10) * pi/180
    q <- q*(45 - 16) + 16
    l <- l*(1 - 0.5) + 0.5
    Rtc <- Rtc*(0.18 - 0.08) + 0.08
    Nz <- Nz*(6-2.5) + 2.5
    Wdg <- Wdg*(2500 - 1700) + 1700

    ## calculation on natural scale
    W <- 0.036*Sw^0.758 * Wfw^0.0035 * (A/cos(L)^2)^0.6 * q^0.006
    W <- W * l^0.04 * (100*Rtc/cos(L))^(-0.3) * (Nz*Wdg)^(0.49)
    return(W)
  }
```

Sensitivity analysis

Now, if computing is cheap we can explore which variables matter and which work together.

Lets make a 2d grid for exploring pairs of inputs.

```
x <- seq(0,1,length=100)
g <- expand.grid(x,x)
```

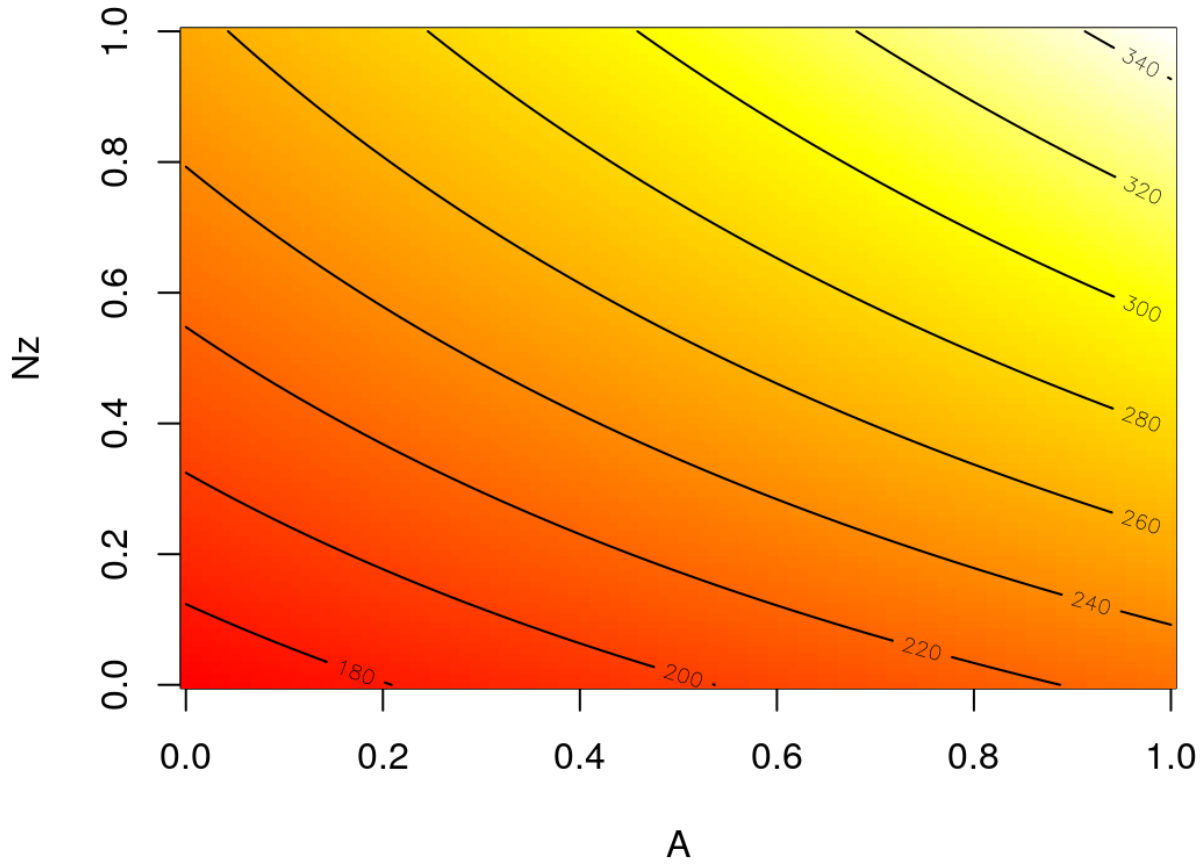
Now we can use the grid to, say, vary N_z and A, with the others fixed at their baseline values.

```
W.A.Nz <- wingwt(A=g[,1], Nz=g[,2])
```

(Some auxiliary code for plotting images with smooth colors.)

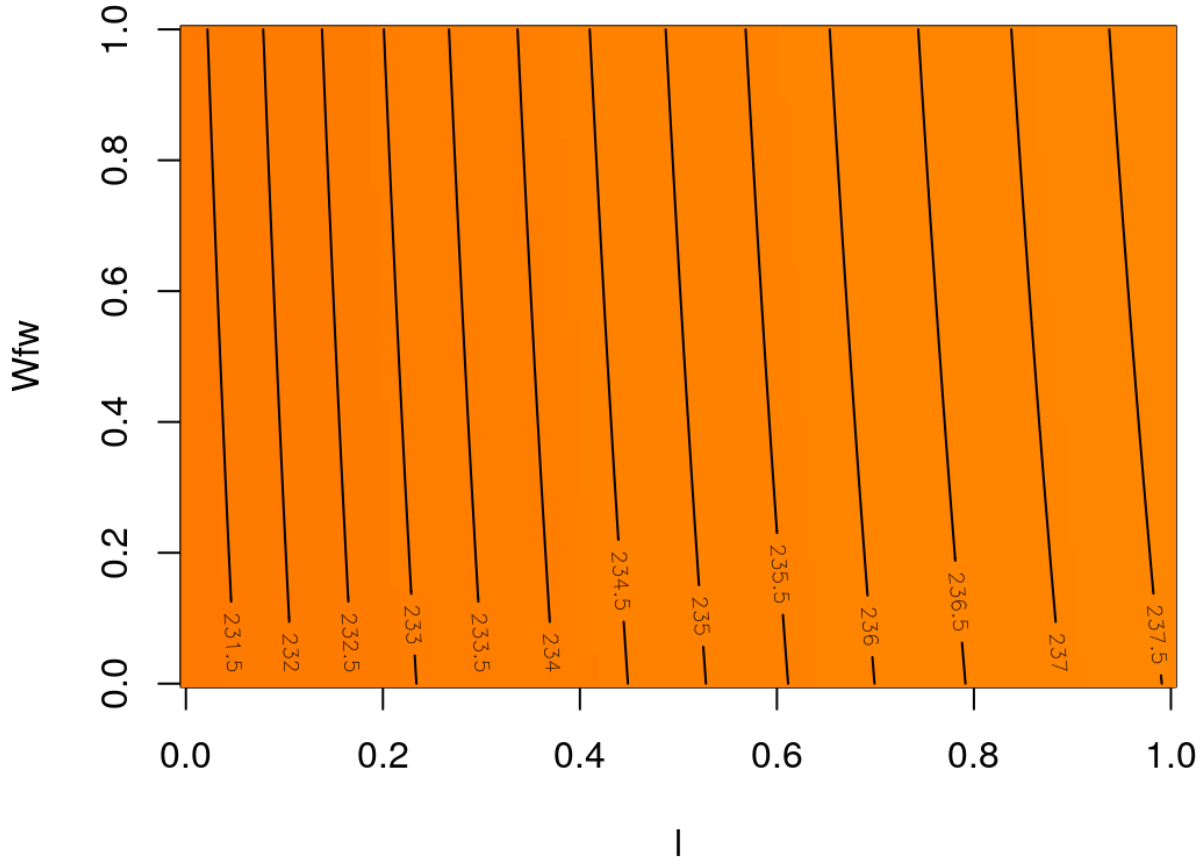
```
cs <- heat.colors(128)
bs <- seq(min(W.A.Nz), max(W.A.Nz), length=129)
```

```
image(x,x, matrix(W.A.Nz, ncol=length(x)), col=cs,breaks=bs,xlab="A",ylab="Nz")
contour(x,x, matrix(W.A.Nz, ncol=length(x)), add=TRUE)
```



- Indicates a heavy wing for high aspect ratios (A) and large g-forces (large N_z).

```
W.l.Wfw <- wingwt(l=g[,1], Wfw=g[,2])
image(x,x, matrix(W.l.Wfw,ncol=length(x)), col=cs,breaks=bs,xlab="l",ylab="Wfw")
contour(x,x, matrix(W.l.Wfw,ncol=length(x)), add=TRUE)
```



- no interaction and very small effect compared to A and N_z

Sensible but expensive

Well that's all fine and good. We've learned about two pairs of inputs (out of 36 pairs)

- and for each pair we evaluated `wingwt` 10,000 times.
- So to do all pairs would require 360K evaluations — not a reasonable number with a real computer simulation that takes any non-trivial amount of time to evaluate.
  - Even at just 1s per evaluation we're talking > 100 hours.
  - Many computer experiments take minutes/hours/days to execute a single run.
- And even then, we'd only really know about pairs.
- How about main effects, or three-way interactions?

We need a different strategy.

Computer model emulation

How about (meta-) modeling the computer model?

The setting is as follows.

- The computer model $f(x) : \Re^p \to \Re$ is expensive to evaluate.
- So we evaluate it at a "small", well-chosen design of locations $X_n = \{x_1, \ldots, x_n\}$, obtaining n pairs (x_i, y_i), where $y_i \sim f(x_i)$ for $i = 1, \ldots, n$.
  - If f is deterministic then $y_i = f(x_i)$.
- The n data pairs $D_n = (X_n, Y_n)$ are used to train a statistical (regression) model, producing an **emulator** $\hat{f}_n$.
- A good emulator does about what f would do.

Surrogate model

More precisely, a **good emulator** can be used in any way f could have been used, qualified with appropriate **uncertainty quantification**.

- Provides a predictive distribution $\hat{f}_n(x)$
  - whose mean can be used as a **surrogate** for $f(x)$ at new x-locations
  - and whose variance provides uncertainty estimates — intervals for $f(x)$ that have good coverage properties;
- Possibly interpolating when the computer model f is deterministic.

Perhaps most importantly, fitting $\hat{f}_n$ and making predictions $\hat{f}_n(x)$ should be much faster than working directly with f.

Space-filling design

Choosing the design X_n is crucial to good performance.

It might be tempting to work on a grid.

- But that won't work in our 9-dimensional exercise.
- Even just having a modest ten grid elements per dimension would balloon into $10^9 \equiv$ 1-billion runs of the computer code!

So-called **space-filling** designs were created to mimic the spread of grids, while sacrificing their regularity in order to dramatically reduce their size.

Latin hypercubes

One easy such space-filling design is called a **Latin hypercube sample** or LHS.

- It is better than a (uniform) random sample (say via `runif` in R) because it is less clumpy, guaranteeing uniformity in marginal samples.
- But it is not as spread out as a so-called **maxmin** design
  - which maximizes the minimum distance between design elements x_i.

Lets generate a 9d LHS …

```
library(lhs)
n <- 1000
X <- data.frame(randomLHS(n, 9))
names(X) <- names(formals(wingwt))
```

… then evaluate `wingwt` at those locations.

```
Y <- wingwt(X[,1], X[,2], X[,3], X[,4], X[,5], X[,6], X[,7], X[,8], X[,9])
```

Gaussian process emulation

Ok now, what do we do with that?

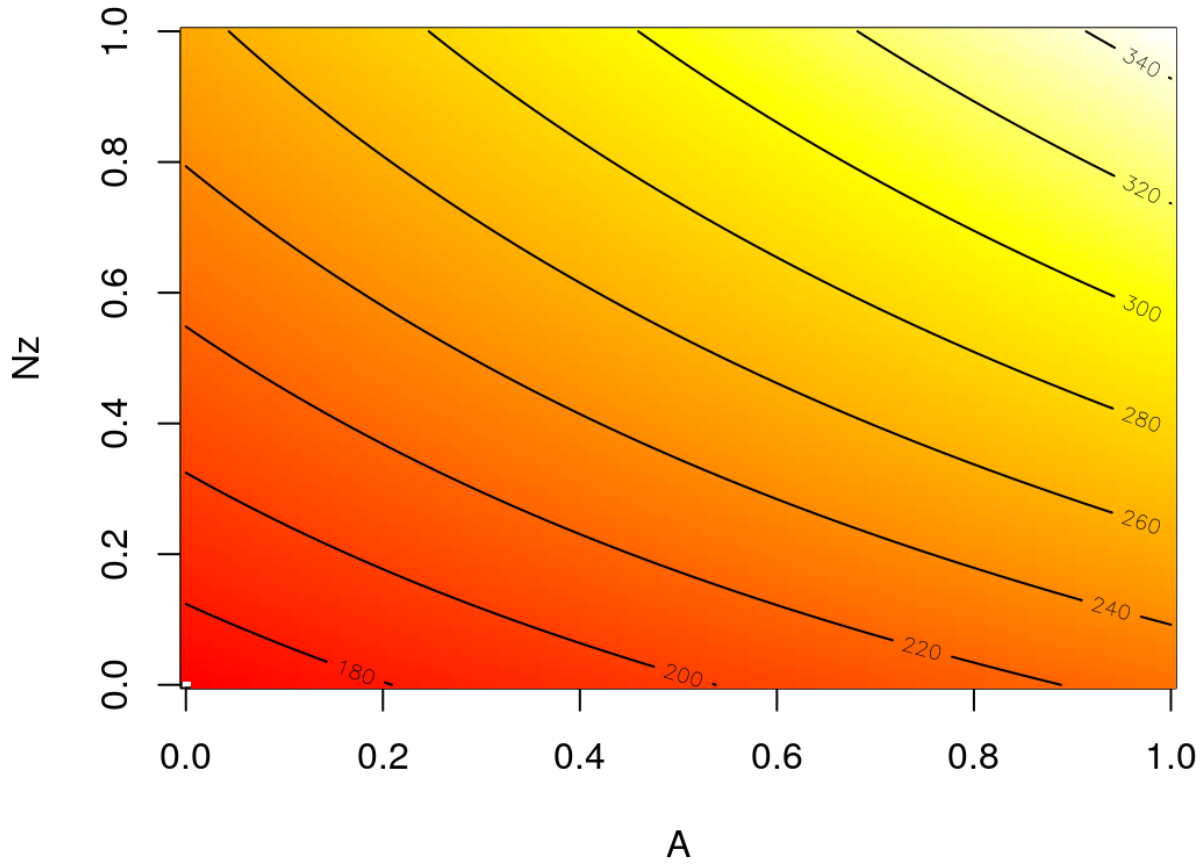
- An emulator could be useful for visualization.
- You could try a linear model, but I think you'll be disappointed.

Gaussian processes (GPs) make good emulators

- but you'll have to suspend disbelief for now.

```
library(laGP)
fit.gp <- newGPsep(X, Y, 2, 1e-6, dK=TRUE)
mle <- mleGPsep(fit.gp)
baseline <- matrix(rep(as.numeric(formals(wingwt)), nrow(g)), ncol=9, byrow=TRUE)
XX <- data.frame(baseline)
names(XX) <- names(X)
XX$A <- g[,1]
XX$Nz <- g[,2]
p <- predGPsep(fit.gp, XX, lite=TRUE)
```

```
image(x, x, matrix(p$mean, ncol=length(x)), col=cs, breaks=bs, xlab="A", ylab="Nz")
contour(x, x, matrix(p$mean, ncol=length(x)), add=TRUE)
```



- Kind of amazing that 1K evaluations in 9d can do the work of 10K in 2d!

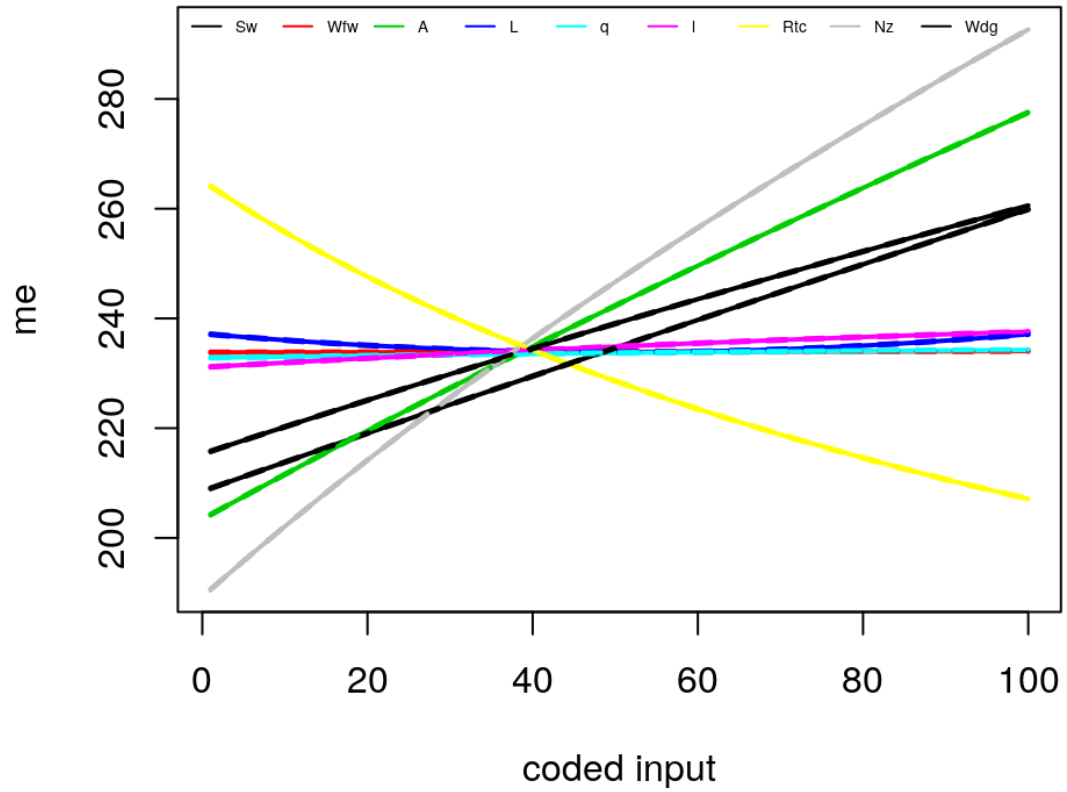
What else?

We can use the emulator, via `predGPsep` in this case, to do whatever `wingwt` could do!

How about main effects?

```
meq1 <- meq2 <- me <- matrix(NA, nrow=length(x), ncol=ncol(X))
for(i in 1:ncol(me)) {
    XX <- data.frame(baseline)[1:length(x),]
    XX[,i] <- x
    p <- predGPsep(fit.gp, XX, lite=TRUE)
    me[,i] <- p$mean
    meq1[,i] <- qt(0.05, p$df)*sqrt(p$s2) + p$mean
    meq2[,i] <- qt(0.95, p$df)*sqrt(p$s2) + p$mean
}
```

```
matplot(me, type="l", lwd=2, lty=1, col=1:9, xlab="coded input")
matlines(meq1, type="l", lwd=2, lty=2, col=1:9)
matlines(meq2, type="l", lwd=2, lty=2, col=1:9)
legend("topleft", names(X), lty=1, col=1:9, horiz=TRUE, bty="n", cex=0.43)
```



- W_{fw}, Λ, q, and λ barely matter!

GP emulation is super powerful

Lots more to come.

- GPs have revolutionized machine learning, geostatistics ("kriging"), and computer simulation experiments.

But they are no panacea.

- They can be **slow** because they involve big matrix decompositions.
- They can over-smooth things,
- and even though they are super flexible they can sometimes be too **rigid**.

The rest of this slide deck sets the stage by introducing four motivating examples where

- there is limited (or no) field data on complicated physical processes,
- and we have computationally expensive computer model simulations.

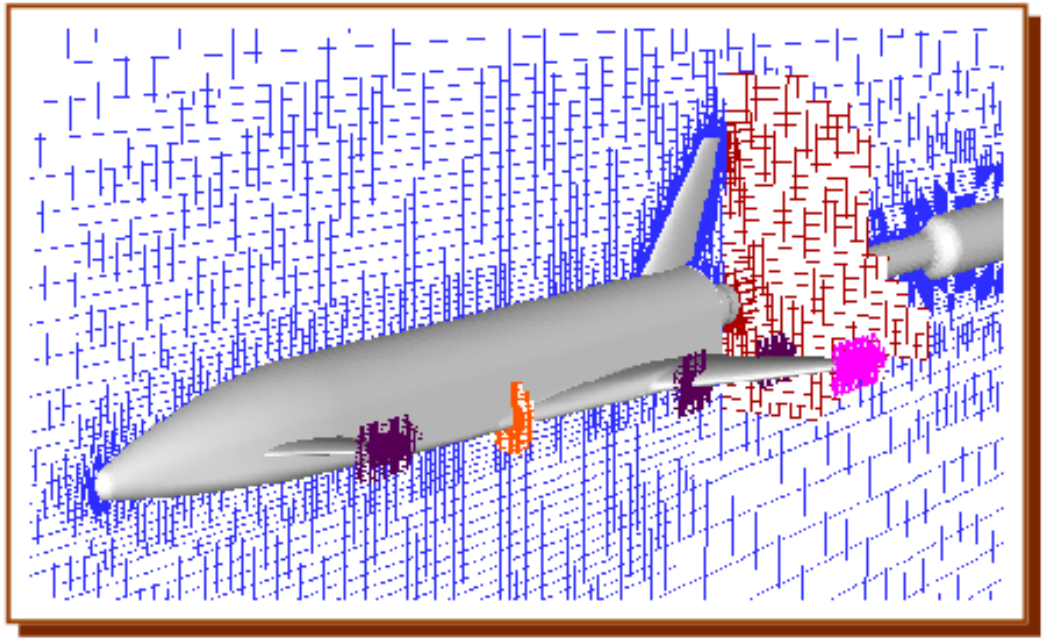
Rocket Booster Dynamics

Langley glide-back booster (LGBB)

NASA proposed a re-usable rocket booster. They developed a CDF solver, **Cart3D**, to simulate dynamics as the rocket re-enters the atmosphere.

- 3 **inputs** describe configuration at re-entry:
- 6 **outputs** delivered in 5+ hours.

3 inputs:
side slip angle
Mach number
angle of attack



6 outputs:
lift
drag
pitching moment
side-force
yawing moment
rolling moment

LGBB data

There are several historical versions of the data.

- The first, oldest, version of the data involves
  - a less reliable code implementing the solver
  - evaluated on **hand-designed input grids**.

```
lgbb1 <- read.table("lgbb/lgbb_original.txt", header=TRUE)
names(lgbb1)
```

```
## [1] "mach"  "alpha" "beta"  "lift"  "drag"  "pitch" "side"  "yaw"   "roll"
```

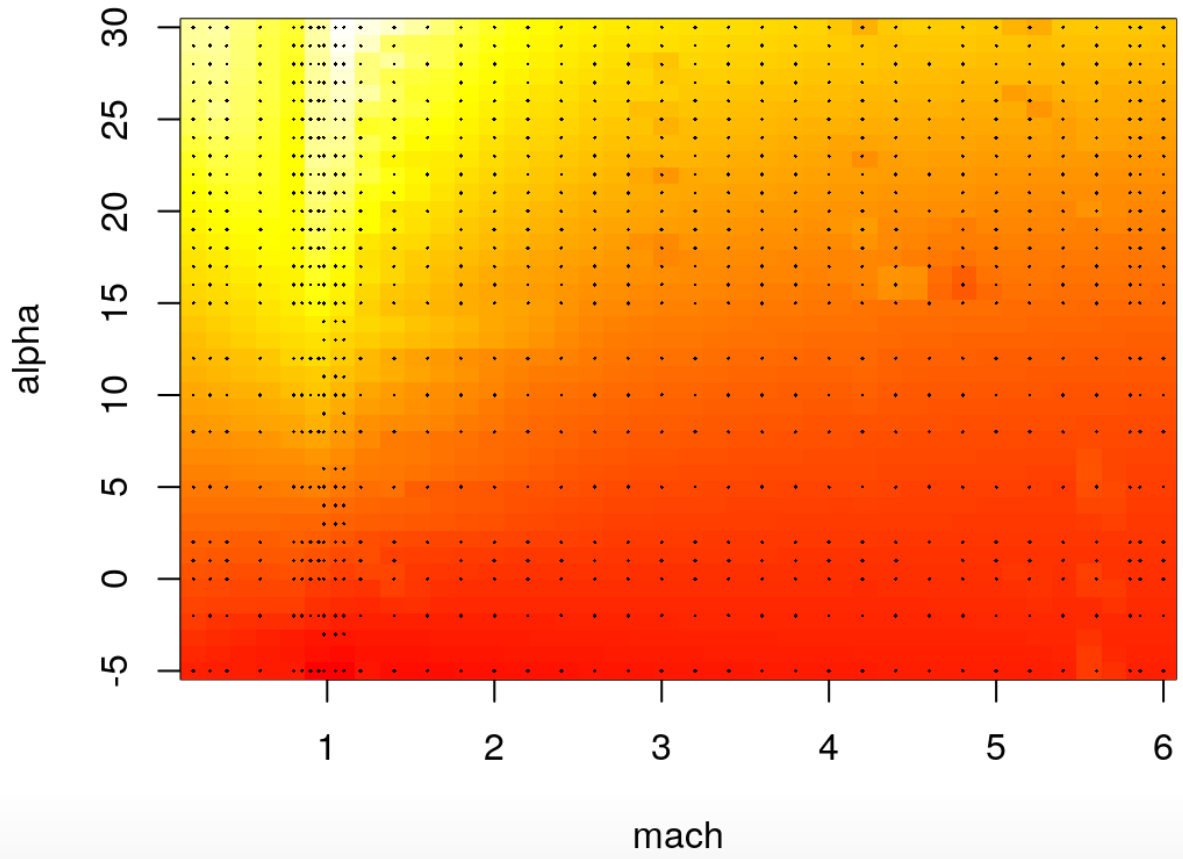
```
nrow(lgbb1)
```

```
## [1] 3167
```

- The grids double up effort in interesting regions, e.g., near the sound barrier.

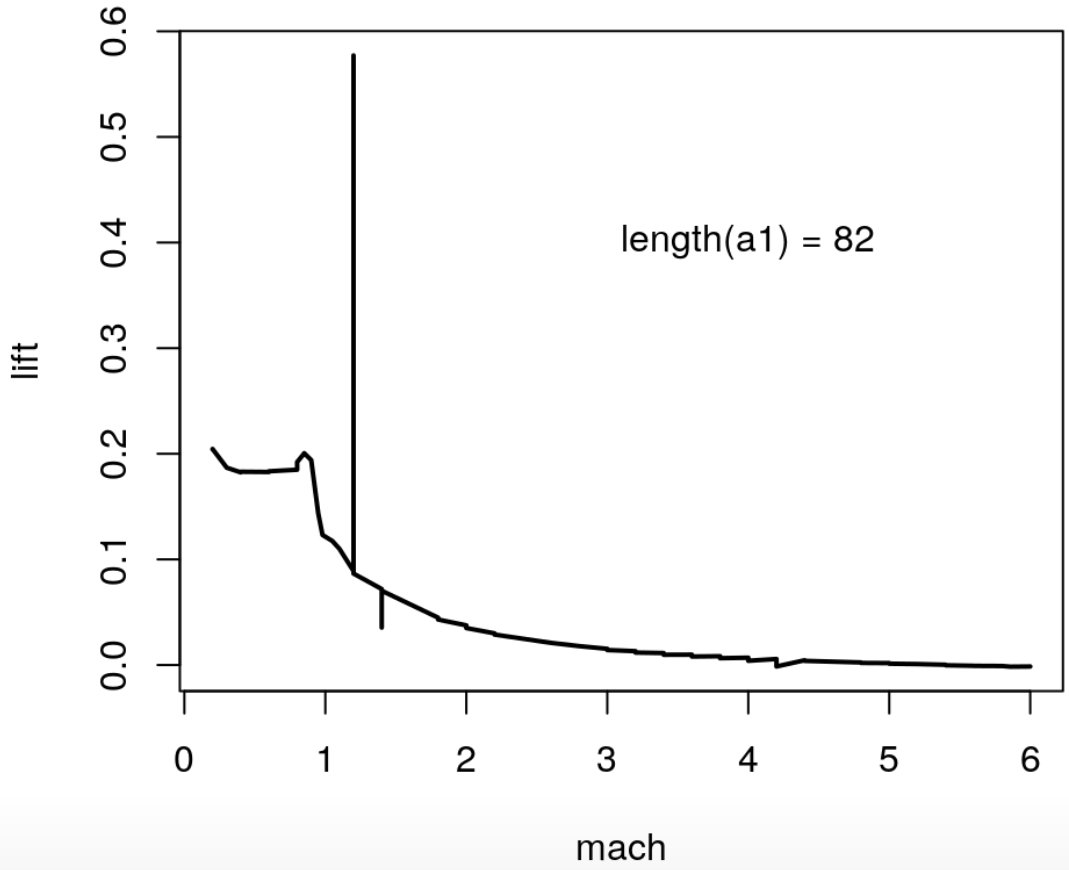
Lift response indicates some numerical instabilities.

```
library(akima)
g <- interp(lgbb1$mach, lgbb1$alpha, lgbb1$lift, dupl="mean")
image(g, col=heat.colors(128), xlab="mach", ylab="alpha")
points(lgbb1$mach, lgbb1$alpha, cex=0.25, pch=18)
```



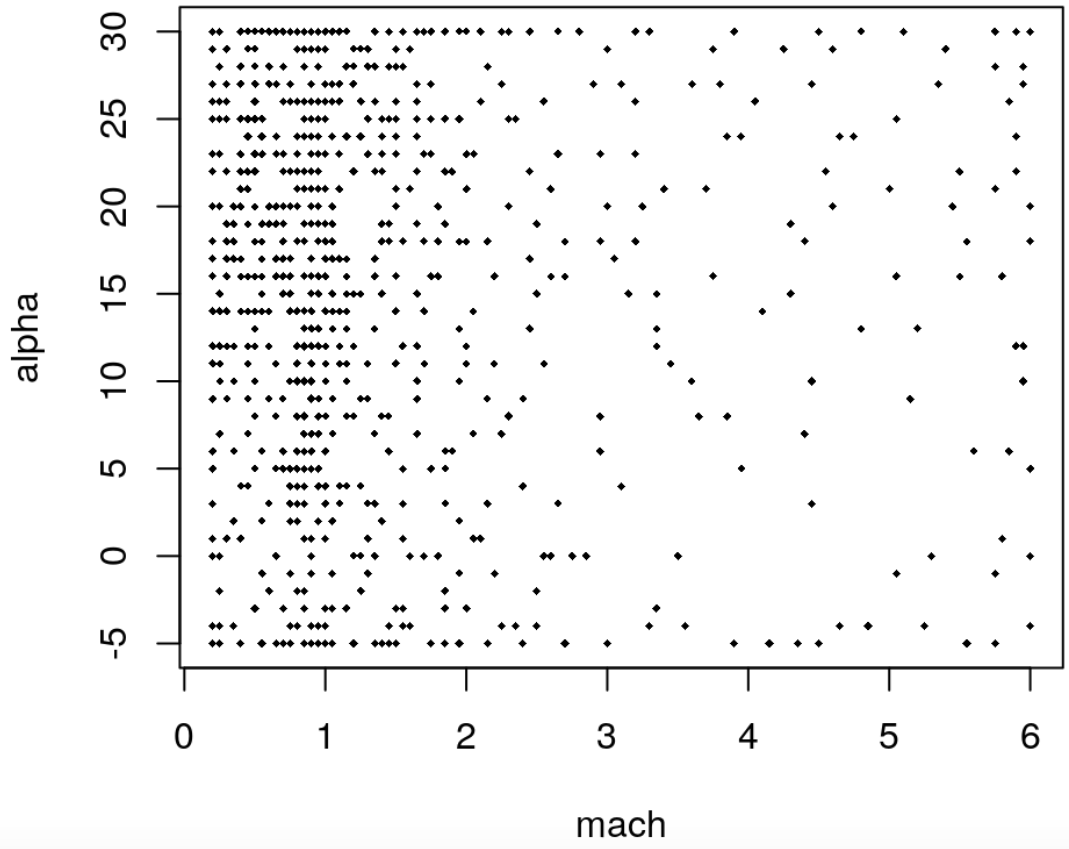
Grids have drawbacks. The data has 3167 rows, but there are only 37 and 33 unique mach and alpha values, respectively.

```
a1 <- which(lgbb1$alpha == 1); a1 <- a1[order(lgbb1$mach[a1])]
plot(lgbb1$mach[a1], lgbb1$lift[a1], type="l", xlab="mach", ylab="lift", lwd=2)
text(4, 0.4, paste("length(a1) =", length(a1)))
```



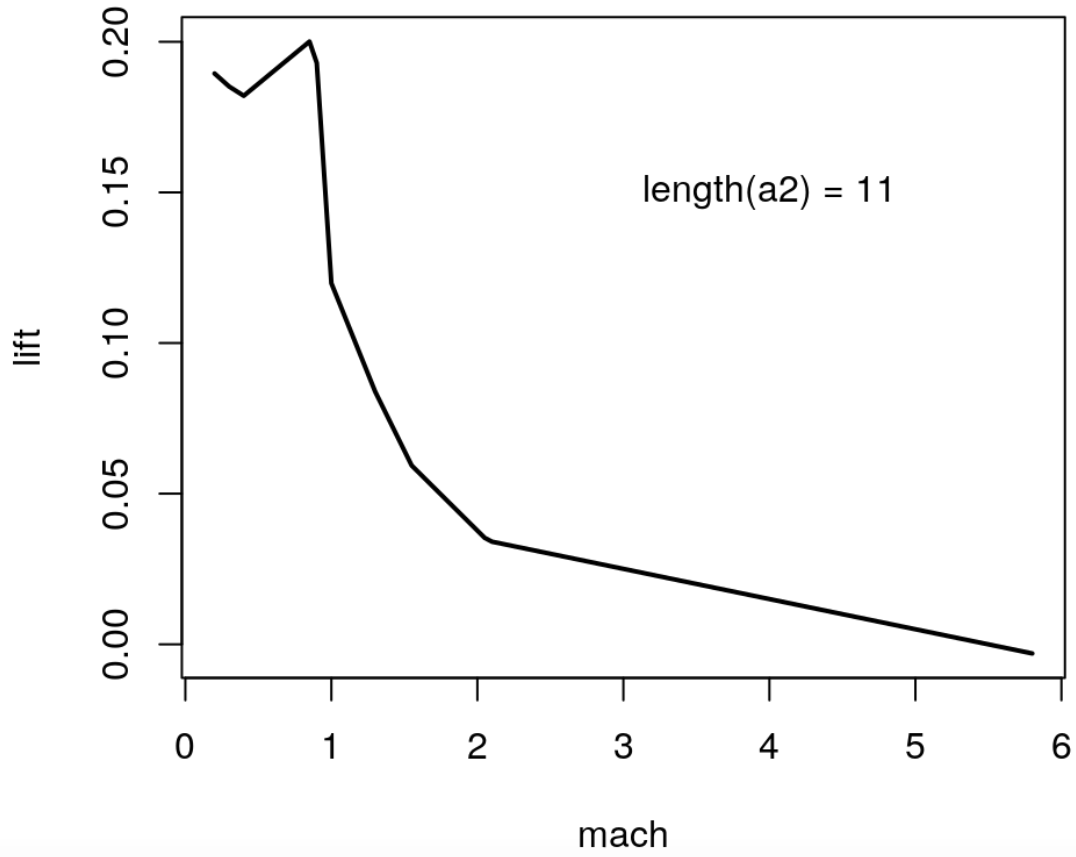
Cart3D.v2 was more stable; and run on an **fully automated adaptive grid** of just 780 points.

```
lgbb2 <- read.table("lgbb/lgbb_as.txt", header=TRUE)
plot(lgbb2$mach, lgbb2$alpha, xlab="mach", ylab="alpha", pch=18, cex=0.5)
```



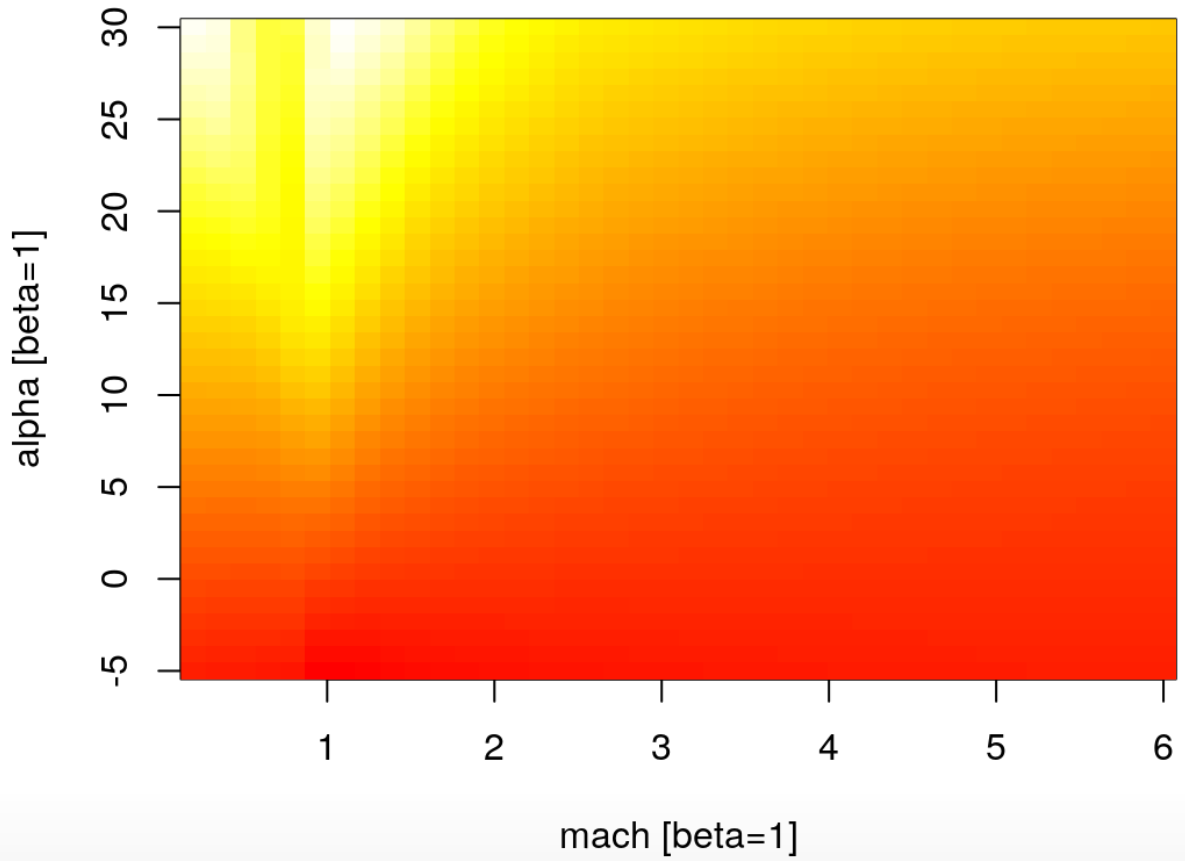
Slices have lower resolution, …

```
a2 <- which(lgbb2$alpha == 1); a2 <- a2[order(lgbb2$mach[a2])]
plot(lgbb2$mach[a2], lgbb2$lift[a2], type="l", xlab="mach", ylab="lift", lwd=2)
text(4, 0.15, paste("length(a2) =", length(a2)))
```

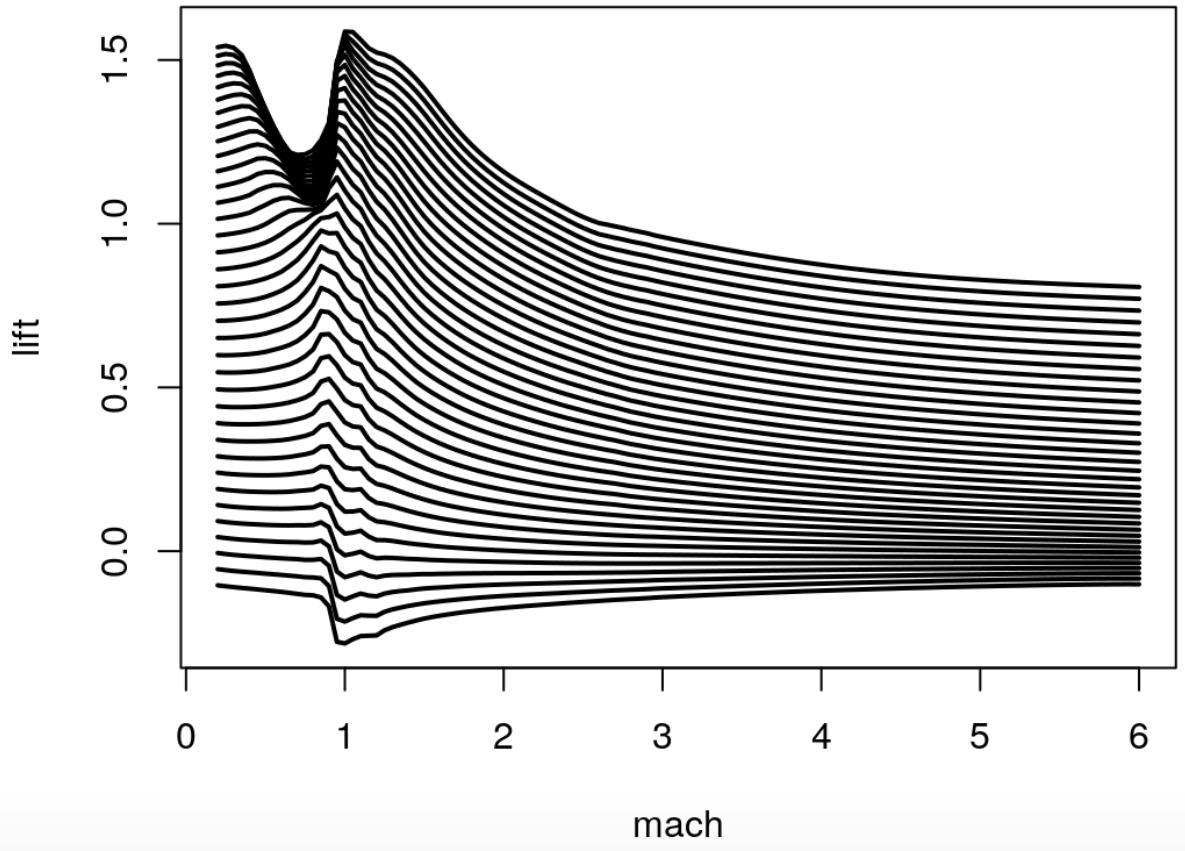


… but GP emulators can fill in the gaps.

```
load("lgbb/lgbb_fill.RData")
lgbb.b1 <- lgbb.fill[lgbb.fill$beta == 1, ]
g <- interp(lgbb.b1$mach, lgbb.b1$alpha, lgbb.b1$lift)
image(g, col=heat.colors(128), xlab="mach [beta=1]", ylab="alpha [beta=1]")
```



```
plot(lgbb.b1$mach, lgbb.b1$lift, type="n", xlab="mach", ylab="lift")
for(ub in unique(lgbb.b1$alpha)) {
  a <- which(lgbb.b1$alpha == ub)
  a <- a[order(lgbb.b1$mach[a])]
  lines(lgbb.b1$mach[a], lgbb.b1$lift[a], type="l", lwd=2)
}
```



Radiative Shock Hydrodynamics

CRASH

Radiative shocks arise from astrophysical phenomena (e.g., super-novae) and other high temperature systems.

- These are shocks where radiation from the shocked matter dominates the energy transport, and results in a complex evolutionary structure.

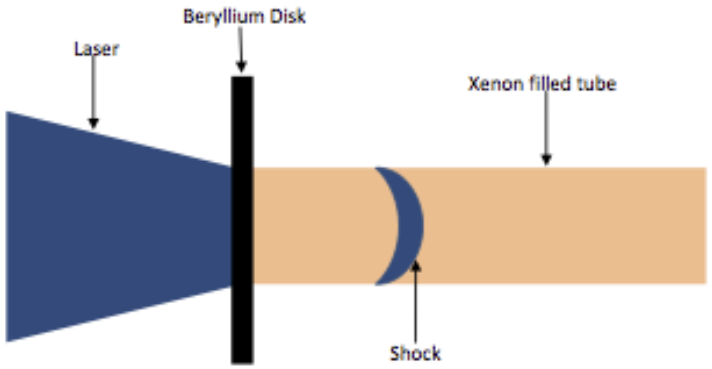
The University of Michigan's **C**enter for **Ra**diative **S**hock **H**ydrodynamics (**CRASH**) is tasked with modeling a particular high-energy laser radiative shock system.

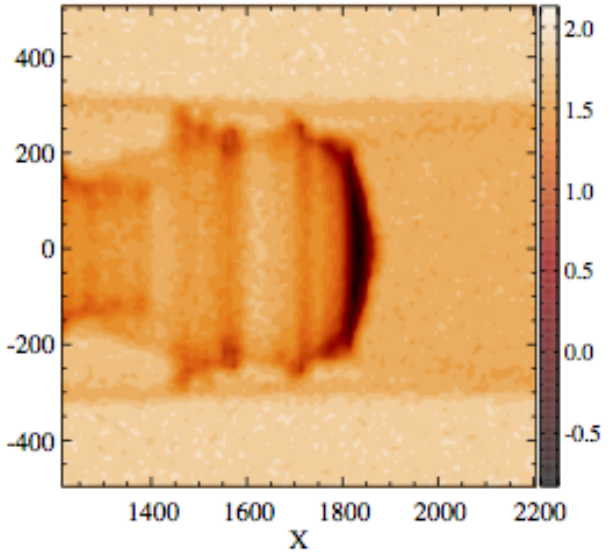
They have

- collected a small amount data from a limited field experiment
- and developed a mathematical model (and computer implementation) that simulates the field apparatus.

Radiative shock experiment

A high-energy laser irradiates a Be disk at the front of a Xe-filled tube, launching a shock.





Experiments involve:

- **9 design variables**: describing energy, disk, tube
- **response**: distance the wave travels in a certain time

Design Parameter	CE1	CE2	Field Design
Be thick (microns)	[18,22]	21	21
Xe fill press (atm)	[1.100,1.2032]	[0.852,1.46]	[1.032,1.311]
Time (nano-secs)	[5,27]	[5.5,27]	6-values in [13, 28]
Tube diam (microns)	575	[575,1150]	{575, 1150}
Taper len (microns)	500	[460,540]	500
Nozzle len (microns)	500	[400,600]	500
Aspect ratio (microns)	1	[1,2]	1
Laser energy (J)	[3600,3990]		[3750.0 3889.6]
Eff laser energy (J)		[2156.4,4060]	

In addition, there are two parameters which pertain only to the computer model

- so-called **calibration** or **tuning** parameters.

Calibration parameter	CE1	CE2	Field Design
Electron flux limiter	[0.04, 0.10]	0.06	
Energy scale-factor	[0.40,1.10]	[0.60,1.00]	

The relationship between design variables and output was explored via

- a **field experiment** with 20 observations
- and **two computer experiments**, 2618 and 2384 runs respectively.

Interest lies in combining the two data sources to learn about radiative shock hydrodynamics.

- This requires calibrating the computer model to the field data.

In the 20 **field data** "runs", only four variables (besides `ShockLocation`) are varied.

```
crash <- read.csv("crash/CRASHExpt_clean.csv")
crash$BeThinkness <- 21  ## Not recorded in field data
print(u <- apply(crash, 2, function(x) { length(unique(x)) }))
```

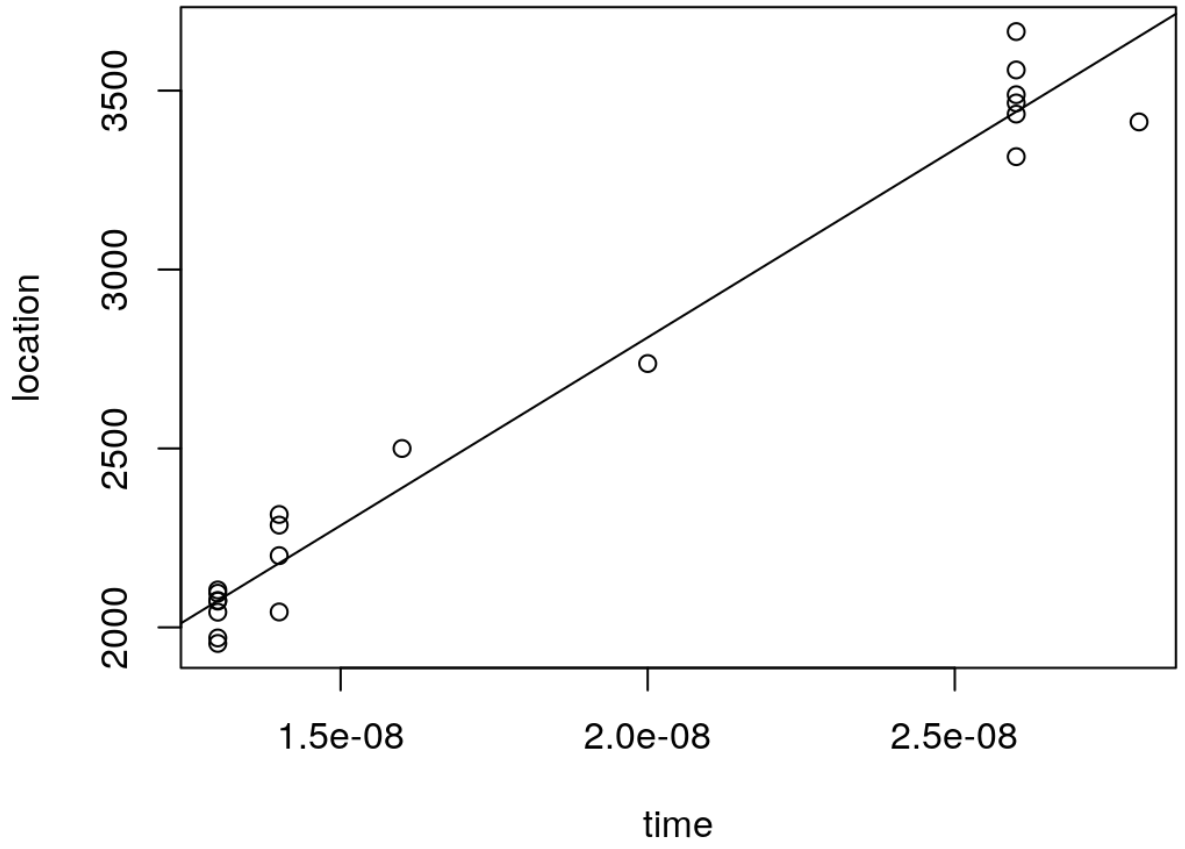
```
##   LaserEnergy   GasPressure   AspectRatio  NozzleLength   TaperLength
##            13            11             1             1             1
##  TubeDiameter          Time ShockLocation   BeThinkness
##             2             6            20             1
```

A linear model indicates that only **time** has a substantial **main effect**.

```
fit <- lm(ShockLocation ~., data=crash[,u > 1])
summary(fit)$coefficients[-1,]
```

```
##                  Estimate   Std. Error     t value     Pr(>|t|)
## LaserEnergy  -3.968075e-01 1.491184e+00 -0.26610235 7.937833e-01
## GasPressure  -1.970699e+02 8.476603e+02 -0.23248692 8.193021e-01
## TubeDiameter -3.423611e-02 4.068208e-01 -0.08415528 9.340459e-01
## Time          1.040318e+11 1.566597e+10  6.64062240 7.866793e-06
```

```
fit.time <- lm(ShockLocation ~ Time, data=crash)
plot(crash$Time, crash$ShockLocation, xlab="time", ylab="location")
abline(fit.time)
```



- Time mops up all of the variability in this data with $R^2 = 0.97$.

Computer model data

Experiment CE1 varied all but four of the parameters.

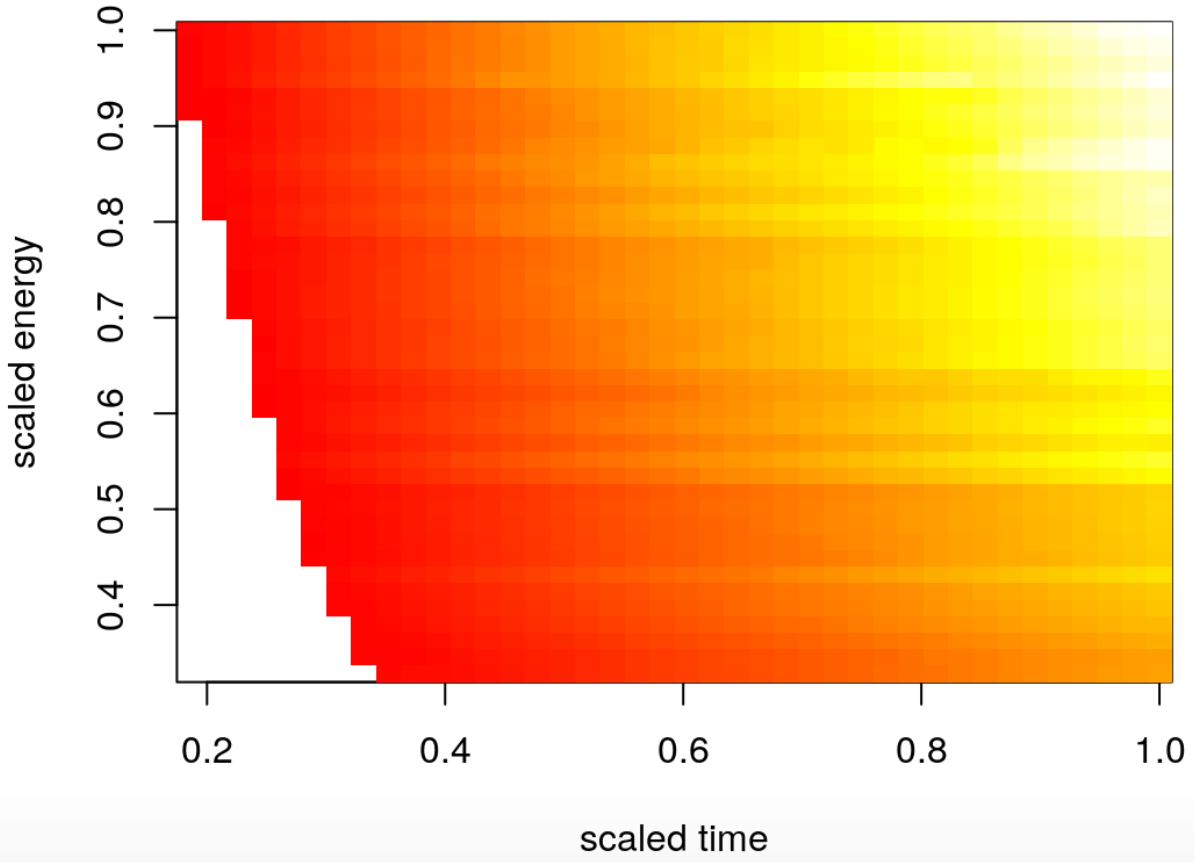
```
ce1 <- read.csv("crash/RS12_SLwithUnnormalizedInputs.csv")
ce1 <- ce1[,-1] ## first col is FileNumber
u.ce1 <- apply(ce1, 2, function(x) { length(unique(x)) })
fit.ce1 <- lm(ShockLocation ~., data=ce1[,u.ce1 > 1])
summary(fit.ce1)$coefficients
```

```
##                          Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept)         -4.601356e+02 1.320949e+02  -3.483372  5.032847e-04
## BeThickness         -7.594590e+01 2.103686e+00 -36.101350 7.283200e-232
## LaserEnergy          3.152568e-01 2.148655e-02  14.672282  6.782470e-47
## GasPressure         -3.829176e+02 8.129088e+01  -4.710461  2.600980e-06
## Time                 1.343696e+11 4.998054e+08 268.843861  0.000000e+00
## ElectronFluxLimiter  4.125576e+02 1.399752e+02   2.947363  3.233373e-03
## EnergyScaleFactor    1.775947e+03 1.214335e+01 146.248499  0.000000e+00
```

- CE1 linear model fit indicates more nuanced relationship (CE2 is similar).

Energy and time work together

```
x <- ce1$Time; y <- ce1$LaserEnergy * ce1$EnergyScaleFactor
g <- interp(x/max(x), y/max(y), ce1$ShockLocation, dupl="mean")
image(g, col=heat.colors(128), xlab="scaled time", ylab="scaled energy")
```



Computer model calibration

However, there is likely predictability left on the table.

- Physical phenomena rarely covary linearly.

We will see how to combine computer model and field data of this sort

- via **computer model calibration**:
  - (non-linearly) emulating the computer model with GPs;
  - estimating the bias between the computer model and the field data;
  - finding the best setting of the calibration parameters relative to that bias;
  - finally, building a predictor that combines (emulated) computer model predictions with bias predictions.

Predicting Satellite Drag

Satellite Orbit prediction

Researchers at Los Alamos National Laboratory (LANL) are tasked with predicting orbits for dozens of research satellites, e.g.:

- **HST** (Hubble space telescope)
- **ISS** (International space station)
- **GRACE** (Gravity Recovery and Climate Experiment)
  - a NASA & German Aerospace Center collaboration
- **CHAMP** (Challenging Minisatellite Payload)
  - German satellite for atmospheric and ionospheric research

Why?

- To plan experiments: what can we see when?
- Adjust course if necessary, for experimental reasons or to **avoid collisions**!

Drag

An important input into their prediction models is **atmospheric drag**.

Drag depends on

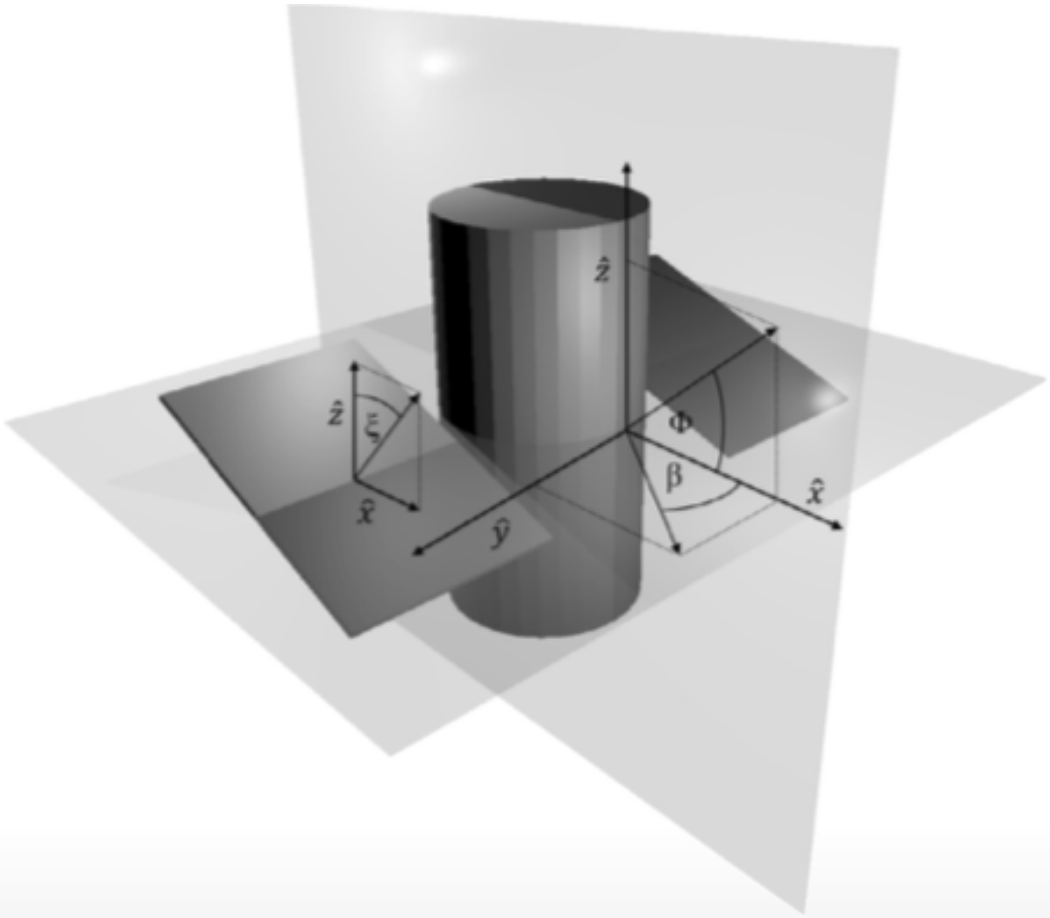
- satellite geometry, orientation, temperature,
- atmospheric chemical composition: concentrations of (O, O_2, N, N_2, He, H);
- which depend on position (latitude and longitude) and altitude.

Numerical simulations

- produce accurate drag coefficient estimates up to uncertainties in atmospheric and gas–surface interaction (GSI) models,
- but are too slow for real-time applications.

Geometry

Geometry is specified in a so-called "mesh file", an ASCII representation of a picture like this, for the Hubble space telescope.



Position and environmental variables

Symbol [ascii]	Parameter [units]	Range
v_{rel} [**Umag**]	velocity [m/s]	[5500, 9500]
T_s [**Ts**]	surface temperature [K]	[100, 500]
T_a [**Ta**]	atmospheric temperature [K]	[200, 2000]
θ [**theta**]	yaw [radians]	$[-\pi, \pi]$
ϕ [**phi**]	pitch [radians]	$[-\pi/2, \pi/2]$
α_n [**alphan**]	normal energy accommodation coefficient [unitless]	[0, 1]
σ_t [**sigmat**]	tangential momentum accommodation coefficient [unitless]	[0, 1]

Emulation goal

Researchers at LANL wanted GP drag emulation

- such that predictions were within 1% of the "true" outputs based on root mean-squared percentage error (RMSPE).

But they realized that they would need $N \gg 4$M runs to accomplish that goal.

- GPs don't scale well to data that big.
- So as proof-of-concept, they limited the range of angles so they could work with a much smaller data set (Metha et al., 2014).

Symbol [ascii]	Ideal Range	Reduced Range	Percentage
θ [yaw]	$[-\pi, \pi]$	[-0.052313, 0.052342]	1.7%
ϕ [pitch]	$[-\pi/2, \pi/2]$	[1.059e-05, 5.232e-02]	1.7%

On the GRACE satellite

Lets look at the GRACE runs (for the He species) that LANL did,

- training on their $N = 1000$-sized design and
- calculating out-of-sample RMSE on a testing set of size 100

```
train <- read.csv("lanl/GRACE/CD_GRACE_1000_He.dat", sep=" ", header=FALSE)
test <- read.csv("lanl/GRACE/CD_GRACE_100_He.dat", sep=" ", header=FALSE)
nms <- c("Umag", "Ts", "Ta", "alphan", "sigmat", "theta", "phi", "drag")
names(train) <- names(test) <- nms
print(r <- apply(rbind(train, test)[,-8], 2, range))
```

```
##          Umag       Ts        Ta       alphan       sigmat        theta
## [1,] 5501.933 100.0163  201.2232 0.0008822413 0.0007614135 1.270032e-05
## [2,] 9497.882 499.8410 1999.9990 0.9999078000 0.9997902000 6.978310e-02
##              phi
## [1,] -0.06978125
## [2,]  0.06971254
```

Convert to coded inputs.

```
X <- train[,1:7]; XX <- test[,1:7]
for(j in 1:ncol(X)) {
    X[,j] <- X[,j] - r[1,j]; XX[,j] <- XX[,j] - r[1,j];
    X[,j] <- X[,j]/(r[2,j]-r[1,j]); XX[,j] <- XX[,j]/(r[2,j]-r[1,j])
}
```

Fit a GP and make predictions

```
library(laGP)
fit.gp <- newGPsep(X, train[,8], 2, 1e-6, dK=TRUE)
mle <- mleGPsep(fit.gp)
p <- predGPsep(fit.gp, XX, lite=TRUE)
rmspe <- sqrt(mean((100*(p$mean - test[,8])/test[,8])^2))
rmspe
```

```
## [1] 0.7401338
```

- Better than 1%.

Big runs

Beating 1% on the whole input space will, for starters, require more runs.

I compiled a new suite of computer model runs for

- HST ($N = 2\text{M}$) for each species, divided equally between panel angles;
- and GRACE ($N = 1\text{M}$) – a smaller design is sufficient, but GRACE is slower,

separately for each chemical species.

- Together, these took about 70K CPU core hours.

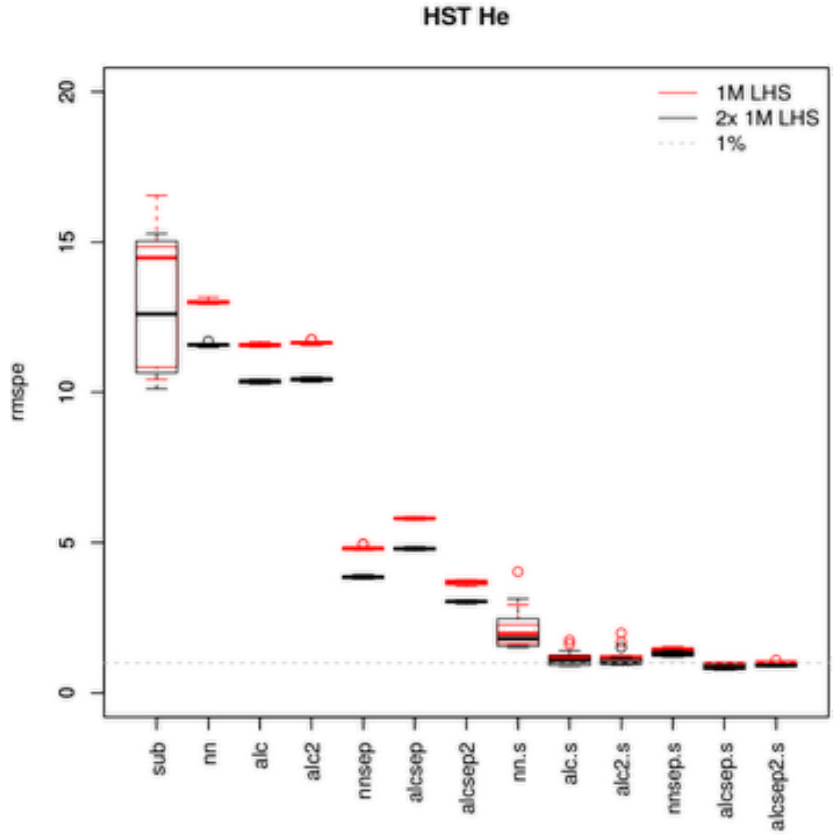
But if GPs struggle with $N \approx 1\text{K}$ how are we going to deal with $N = 2\text{M}$?

- We'll have to cut corners somehow.

Promising results

A soft divide-and-conquer technique called "local approximate GPs" works.

- We'll learn about `laGP` and some other big data GP alternatives.



Groundwater remediation

Dirty water

Worldwide, there are more than 10,000 contaminated land sites (Meer et al., 2008).

- Environmental cleanup at these sites has received increased attention over the past 20-30 years.

Preventing the migration of contaminant plumes is vital to protecting water supplies and preventing disease.

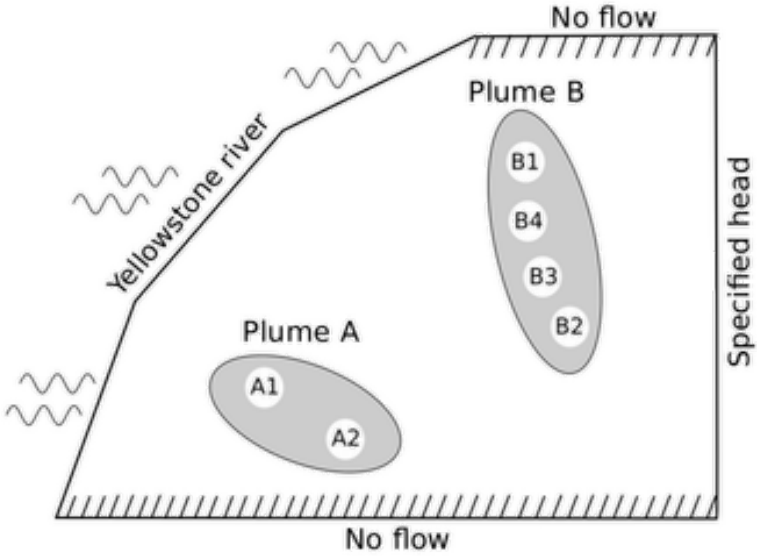
One approach is pump-and-treat remediation, in which wells are strategically placed to

- pump out contaminated water,
- purify it,
- and inject the treated water back into the system to prevent contaminant spread.

A case study

Consider the 580-acre Lockwood Solvent Groundwater Plume Site, an EPA Superfund site located near Billings Montana.

- As a result of industrial practices, the groundwater at this site is contaminated with volatile organic compounds that are hazardous to human health.
- To prevent further expansion of these plumes, six pump and treat wells have been proposed.



Computer model and optimization

The amount of contaminant exiting the boundaries of the system (in particular the river) depends on

- the placement of the wells and their pumping rates.

An **analytic element method** groundwater model was developed

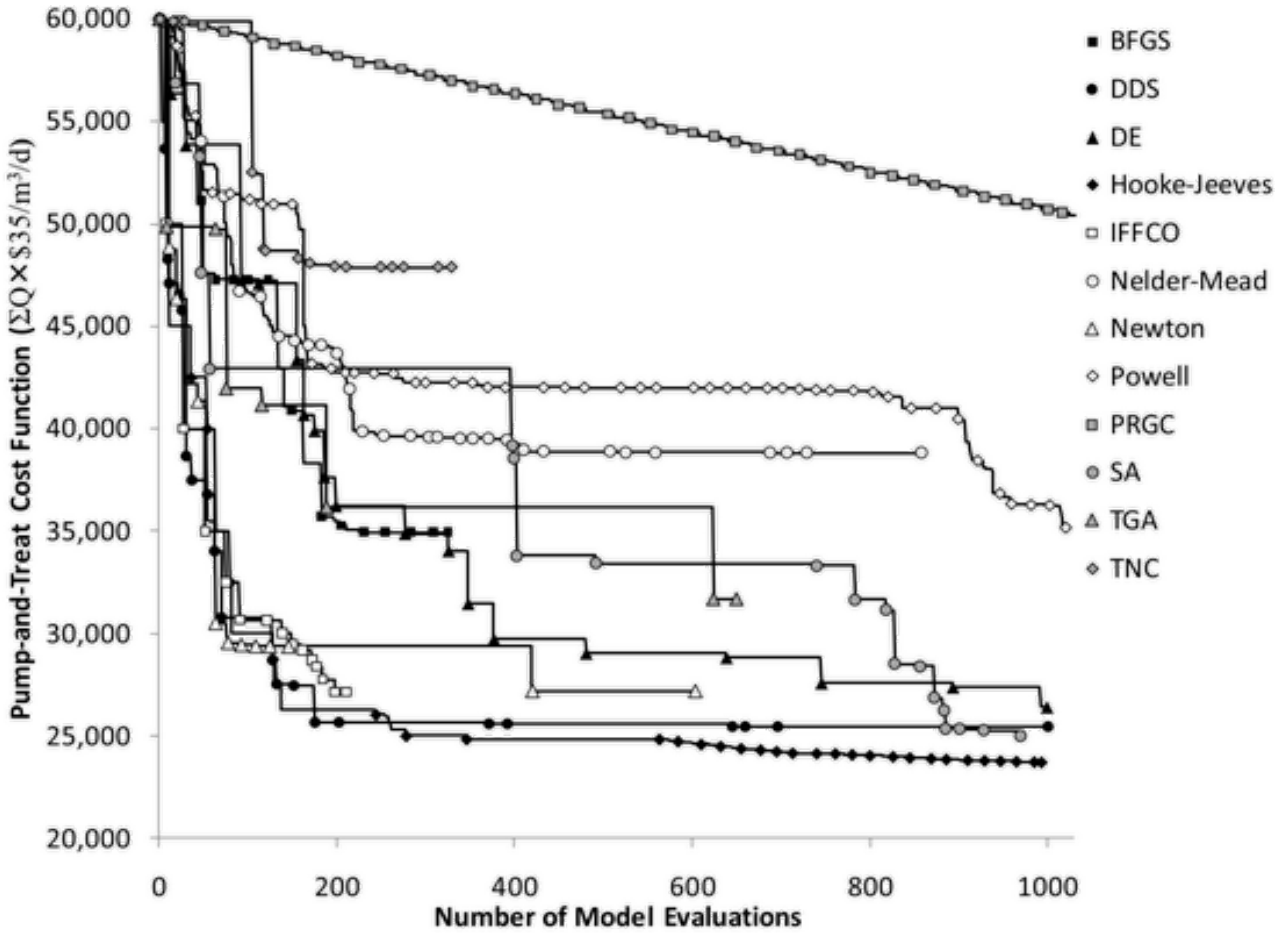
- to simulate the amount of contaminant exiting the (2) boundaries under different pumping regimes (Matott, et al., 2006).

Mayer, et al., (2002) first posed the pump-and-treat setting, generically, as a **constrained "blackbox" optimization** problem.

- Fixing the well locations, let $x_1, \ldots, x_6$ denote pumping rates for six wells, consider

$$\min_x \left\{ f(x) = \sum_{j=1}^{6} x_j : c_1(x) \leq 0,\ c_2(x) \leq 0,\ x \in [0, 2 \cdot 10^4]^6 \right\}.$$

Matott, et al., (2011) compared MATLAB and Python optimizers, treating constraints via the **additive penalty method**, initialized at the known-valid input $x_j^0 = 10^4$.



Objective improving comparator

It is interesting to ask …

- What makes the good methods good?
- Why do the bad methods (in some cases) fail so spectacularly?
- And by the way, how are statistics and RSMs involved?

Consider the following random search method that I call **objective improving candidates**.

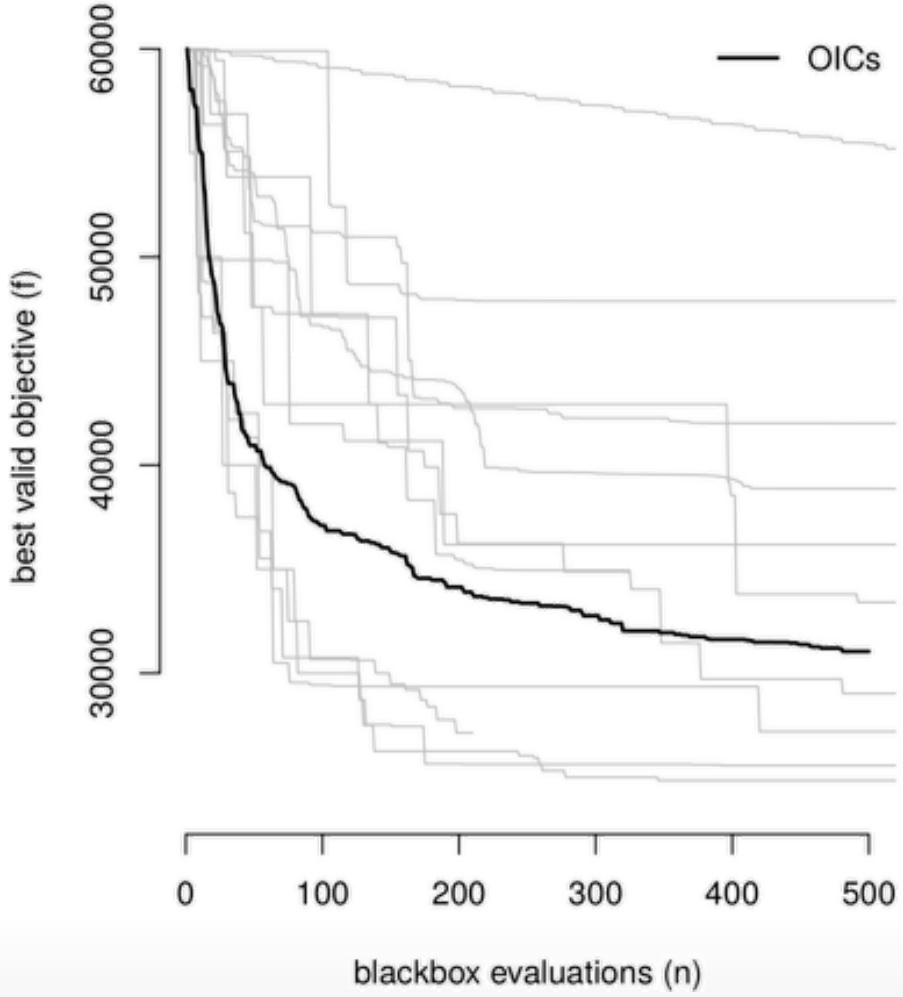
Given the current best valid input x^*, i.e.,

- $c(x^*) \leq 0$, and
- $f(x^*) = \sum_j x_j^* < f(x)$ for all other (tried so far) x such that $c(x) \leq 0$,

draw uniformly from $\{x : f(x) < f(x^*)\}$, for example via rejection.

Here I've extracted the first 500 iterations from Matott, et al., (2011),

- which are in `runlock/pato_results.csv`,
- and added average progress (best valid value) from 30 repeated runs of OICs.



Sequential design

Half of the MATLAB/Python methods are not doing better (on average) than a slightly modified "random search".

- They are getting stuck in a local minima, and failing to explore other opportunities.

Fitting a surrogate model to blackbox evaluations can allow statistical decision criteria to judge trade-offs between reward and uncertainty;

- in this case, balancing exploration and exploitation.

Sequential design is the process of using (surrogate) model fits to drive future data collection, in order to maximize information or reduce variance, say.

- One popular application of this idea to optimization is called **expected improvement (EI)**.
- The machine learning community calls this **Bayesian optimization** owing to the Bayesian interpretation of GP learning.