Data-Driven Techniques to Model Complex Systems [1]

Val K. Bykovsky
NRC Senior Associate
Virtek, Inc.
val@vtek.com

The further backward you look, the further forward you can see Winston Churchill

1.1. Introduction

The observation of the *dependencies* between the data and the conditions of the observation always was and is a *primary* source of knowledge about complex objects and their dynamics.

We are interested in *direct* program-driven analysis of data and their dependencies with the goal to build a model directly in computer and thus to predict the dynamics of the object based on the measured data. We see the *generalization* of data dependencies as a critical step in building data-driven models. This approach mimics a traditional model building process - from data to a model - but the program-driven model building has a number of benefits. The direct analysis of data dependencies is a key point of building any data-driven predictive model.

There are two main sources of data dependencies: (1) *direct* physical experiments and (2) *indirect, computer or in-silico* experiments. The direct experiments immediately generate data dependencies of interest. The computer experiments is a way of using computer as an electronic (fast) experimental setup to get the "measurements data". Such a setup mimics the real experiments when those are difficult or impossible or very costly. Thus, the experiments get simulated in computer and the output data ("measurements") get collected. That is an alternative to using a computer as a fast calculating device. "Good computing requires a firm foundation in the principles of natural behavior", wrote Metropolis [H83]. The experimentation approach was proposed by S. Ulam and N. Metropolis [N87, B00] at Los Alamos when designing weapon systems with direct experiments basically impossible. Of course, as in any experiment – physical or computer one – the degree of its correspondence to reality is a challenging issue to be analyzed.

Originally, their idea was to develop the "experimental approach", where the computer-driven experiments and their data would be an integral part of the process aimed to understand a real complex phenomena. The idea of proposed Monte Carlo (MC) method [N87] was to generate the "experiments" in a computer. This included a generation of *random* initial experimental configurations (events) and their evolution. Such "experiments" represented in certain degree the properties and dynamics of the phenomenon of interest.

However, it was not just events generation and data collection. N. Metropolis wrote in his historical account of development of the Monte Carlo method [N87]: "It is interesting to look back over two-score years and note the emergence, rather early on, of experimental mathematics, a natural consequence of the electronic computer. The role of the Monte Carlo method in reinforcing such mathematics seems self-evident." Stressing the hands-on aspect of experimental mathematics, he wrote then: "When shared-time operations became realistic, experimental

[1] Presented at the International Conference on Complex Systems, June 25-30, 2006, Boston, MA

mathematics came to age. At long last, mathematics achieved a certain parity – the twofold aspect of experiment and theory – that all other sciences enjoy."

The idea of bridging a gap between a data source and the model based on the data was also introduced by Metropolis [A86, B00], and he demonstrated it in late 50s and early 60s by two-way coupling a computer to the Navy cyclotron.

The same idea of dynamic integration of modeling and using the results to steer the new measurements was recently resurrected in the NSF-sponsored program DDDAS [NSF]. Such a dynamic data-driven application simulation (DDDAS) is "a paradigm whereby application (or simulations) and related measurements become an integrated feedback-driven control system". . The need for such dynamic applications is obvious in the context of fast-changing real world and keeping in mind that the existing data analysis paradigm was developed hundreds years ago for totally different world. We extend the DDDAS paradigm to include dynamic data-driven *model building*, *testing* and *update* with new data measured *on-demand*.

Reflectance models [M03] is another successful example of data-driven modeling. It shows capabilities of data-driven modeling in reconstruction of dynamics of realistic 3D objects involved in complex processes such as rusting.

Our approach additionally includes a *model building* component based on *generalization* of dependencies in the data. That part was not explicitly presented in the MC method. It can be added to the electronic experimentation process in order to *generalize* data accumulated in the experiments and obtain the data-driven models of the phenomenon with various degree of details. Importance of *data correlation* analysis was also stressed by Metropolis [H83]: "Theory, described in its most homely terms, is the cataloging of correlations, and mathematics is the language whereby these correlations can be expressed with precision."

In addition to experiments with *random* generation of events and following their histories, we introduce also the *directly measured* data. In this case, *statistical sampling* [B87] is not required, and the experiments can be built in computer on *individual* basis and in a dynamical, *measurement-on-demand mode*. We also *extend* the event-generation and event-handling capabilities. Originally, the *random numbers* were the foundation for the generation dynamics. We introduce the *feedback-driven capabilities* to *control* the events generation and evolution process, a step that makes the event generation/evolution more realistic and easy-to-modify to match the application criteria. The controlled search in the configuration space is a critical component of that approach.

The last but not least addition is the *persistent data framework* structured by data obtained in the experimentation process, an *infrastructure* that provides the functionality and the tools to support all the mentioned functional components. One of those is a *data container*, a template for realistic data capable to handle dimensionalities and any other *properties* of *real-world data*. The data containers also may have a built-in logic to support a flexible *in-situ* data validation and analysis. A package Traits [T06] to support that advanced functionality is an impressive example of such advanced data-handling capabilities.

The recent efforts to make databases more dynamic and integrated with the logic of programming languages were described in [G04]. As a whole, the proposed approach conceptually mimics the traditional model building process, however *starts with the data* and *data dependencies*, and builds an online data-driven model (mapping) by *programmatic* generalization of those data dependencies. That introduces the power and flexibility of programmatic technology (vs. traditional model-specific *analytical* efforts). The data source becomes a part of the model building process and is available for online model testing and updating. In the traditional model building process, when the model is built, its connection with the data source gets lost, and the model (symbolic equation) lives its own life occasionally interacting with the external data such as parameters and initial/boundary conditions.

Being programmatic, the proposed approach is *complexity-neutral* and so can be used for analysis of dynamics of complex real-world objects.

1.2. Data, Measurements, and Data Dependencies

1.2.1. What is Data?

Generally, data is a combination (a pair) of data and its *context*, that is, the initial, boundary and all the other *necessary* conditions that make the measurement and the data generated *unique*. Thus, the data is *always related* to its context sometimes called *metadata*. Each field and a record may have *metadata* attached which *describe* the nature of the fields and the record - dimensionalities, comments, annotations, commands to check/validate the fields and the record, etc. The metadata is a critical part of the data exploration and model building that allows one to establish a *primary* dependence between the *context* of the measurements and the variables of the problem (application). With the metadata attached, a field position in a record may not be its identification any more; and so data then can be located, accessed and interpreted based on its application-specific attributes, a flexibility which may drastically help to explore data.

With metadata, access to data can be based on its application-specific *attributes* and less on the field *names*. The access to data becomes much more flexible - in particular, it can be *associative*, so that *all* the data with the same tag can be easily accessed, collected, moved, processed, and placed in a database based on its description and/or properties. In its turn, processing the data may lead to *updating* its attributes. Thus, data can be analyzed and handled as well as combined into new datasets based on their description and properties, the opportunities which are not available in the traditional tabular datasets with the common *positional* addressing the fields.

In its turn, the physical experiments need to be designed to supply metadata for the data to be measured thus providing proper dependencies `p=(mData,Data)`, where a data pair `p` is the *unit* and primary source of *dependencies* to be analyzed and generalized to build a predictive model based on data. The experiments to be designed should be *configurable* and have the extra sensors – *metasensors*, that is, sensors for metadata. This allows one: (1) to measure data for various *configurable* conditions and (2) to attach the metadata to the measured data *automatically* using meta-sensors readings . Thus, a set of data pairs `p[1],p[2],..,p[n]` can be generated for a *representative* range of conditions.

The *programmatic* approach to data exploration also *enhances* the *concept* of data. Say, a record gets elevated to the level of a *data structure* with the built-in *logic* designed to validate and test the data. That makes a record a *live* or *active* record and opens a new dimension in data exploration making it a highly flexible process easy to adapt to various application requirements. Live or active record thus becomes a *data container*. That enhancement also makes easier a generalization of data dependencies and building a data-driven model.

1.2.2. State Vector Dynamics and Data Dependencies

In simple cases, data is a set of observations `rec = (f[1],f[2],..,f[N])`, where `f[i]` is a `i`-th field, usually the number to be interpreted in the *context of its position* in the record.

A data record taken as a measurement of a system state at a specific time is actually a system *state vector*, and its change in time describes the system dynamics which used to be described by the system “motion equations”. So, analysis of data record or state vector dynamics and revealing hidden patterns and regularities in data can be *likened* to the analysis of the system dynamics in the conventional sense - by building the motion equations and their integration in space and in time.

In the conventional model-building process, one analyzes a representative set of data - in fact, a set of *data dependencies*. Each data dependence is a relationship between the conditions `x` of the experiment and the data measured `y`; this relationship `p=(x,y)` is a *unit* of data dependency which also can be represented in usual form `y=f(x)`. The function `f` is unknown, and should be determined. The relationship can also be a differential one establishing an *incremental* dependence such as a balance between input and output variables for a volume unit. In other words, in the traditional model-building process, one originates from elementary (usually differential) data dependencies, and *generalizes* those into a symbolic (analytical) model by brain storming. Then by another brain

storming the model/equation is converted into an algorithm, and then into a program to be executed to get an estimate for *new* input data. Importantly, the estimation requires an *initial/boundary conditions* to *select* a specific solution of interest. A differential relationship itself is not enough to represent the overall spatial behavior which needs to be *specified* by proper initial and/or boundary conditions.

well as the records *are* the
to find the *common features*.
nto a symbolic equation is a
iation: generalization of data
the proposed approach is to
s in the data dependencies
imations - predictions. With
it is *complexity-neutral*.

, is given in **Fig. 1**. The first
the second is the function
oduct `out(t,x(t))` of two
=dot(t,out).

pes and the trajectories. This
spatial superimposition. The
d trajectory.

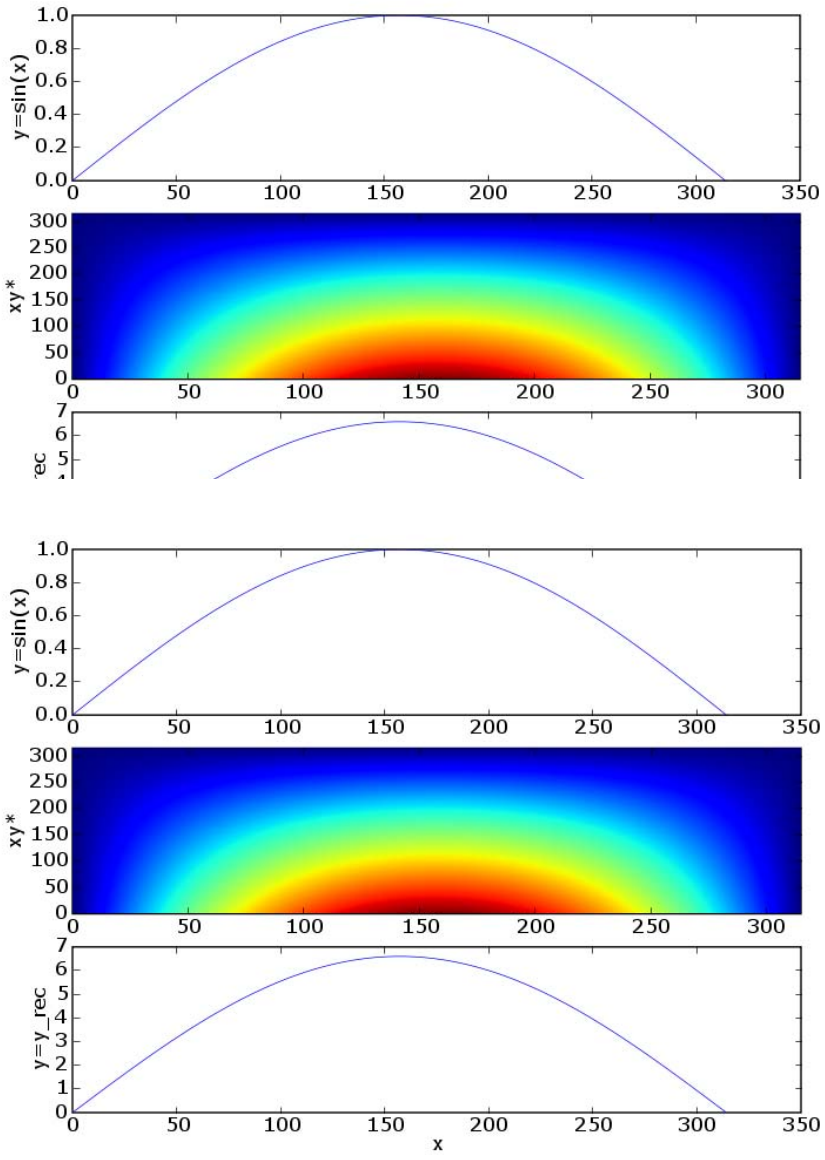

Fig. 1. Building a trajectory reconstruction model. **(a)** Dependence `p=(t,x)` between `t` and the function `x(t)` was represented as the 2D pattern **(b)**, an outer product `out(t,x(t))`, a mapping `x=M(t)` which allows for reconstruction **(c)** of trajectory `x` for a new slope `t`

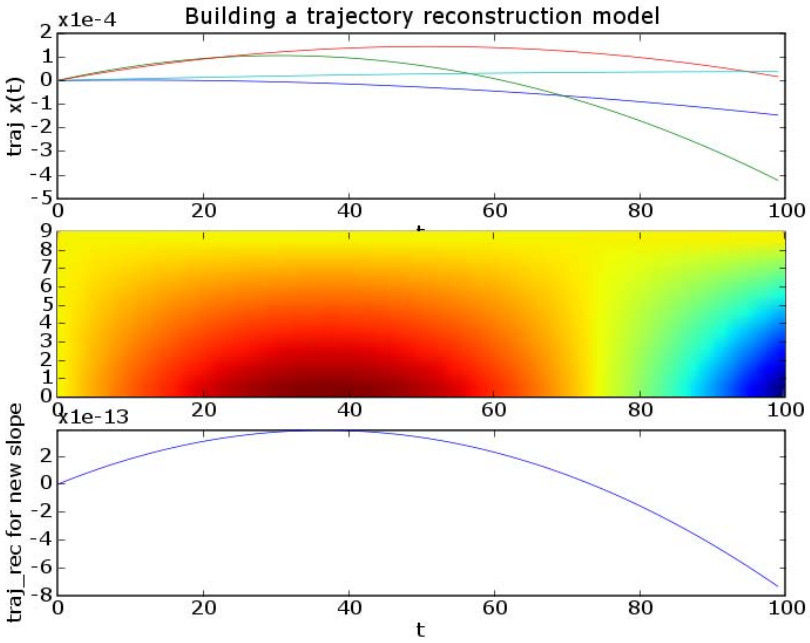

Fig. 2. Building a trajectory reconstruction model. **(a)** Dependence `p=(t,x)` between `t` and the trajectory `x(t)` has been represented as the 2D pattern, an outer product `out(t,x(t))`. **(b)** The dependencies `p[i], i=1,2,3,4` for 4 trajectories were computed as 2D patterns and generalized by spatial overlapping; this created a mapping `x=M(t`. **(c)** Reconstruction of *new* trajectory `x` for a *new* slope `t`.

1.2.3. A Dynamic Data Source: A Gas-of-Particles Approach

In that case data is generated by a data source that represents a state-vector dynamics of a system, for example, of the gas of atoms - hard balls. Of course, more realistic gas models can be handled as the approach is *programmatic* and therefore complexity-neutral. To monitor data dependencies of interest, the proper *events* need to be chosen, and then captured, accumulated and generalized to see if there are any *common features* in the behavioral dynamics, a foundation for the data-driven model of the gas.

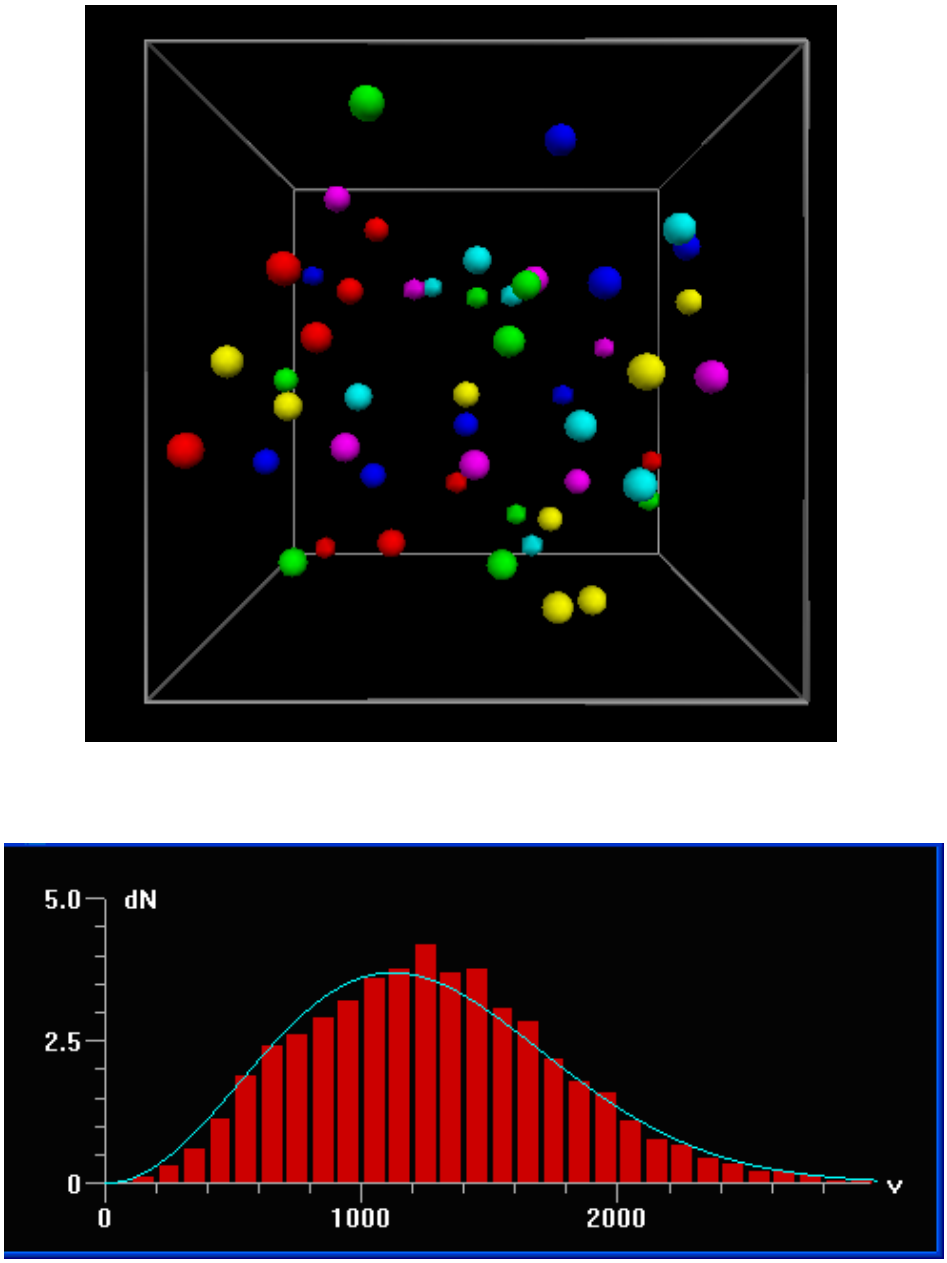

Fig. 3. **(a)** Building a data-driven model for a dynamic data source - the gas of hard-ball atoms at certain temperature. The system and its state at a certain time. **(b)** A velocity distribution in the histogram mode with the bins showing the average number of atoms with specified range of velocities. A continuous curve is the theoretical Maxwell distribution

With the model built this way, the gas dynamic behavior can be estimated using the developed mapping (the model) and the *new* experimental conditions - size, temperature, atom potential, etc.

In the programmatic data exploration mode, data can be *flexibly* cross-correlated to see the *strength* of specific *dependencies* between the fields or records. The strong dependencies can then be used for generalization, that is, for model-building. For example, the in-record correlations between different fields can be easily evaluated by computing the `f[i]*f[j]` products, averaged over many records (over time).

For a pair of records `p` and `q`, the direct or outer *product* can be computed:

```
c(p,q) = |p><q| = A(p[i]*q[j]), i,j=1..n,m,
```

where `A` is an outer product matrix PQ*, and records `p` and `q` may be considered to be vectors with the fields taken as components.

An interesting *aspect* of the data-driven approach to the analysis of complex dynamics is a possible interpretation of a record (a measurement) as a *particle* with its *properties*, and the inter-record dependencies as the *interactions* between the particles. Then the observation dataset can be thought of as a system of interacting particles, and various characteristics such as energy, entropy, one-, two- and many-particle distribution functions can be computed with the records-particles to better understand the system dynamics. This language and particle-based statistical paradigms fill a gap between traditional physics-based model building and programmatic data-driven model building, and can be useful in revealing *programmatically* the hidden patterns and regularities in the massive datasets such as, for example, hyperspectral imagery, micro-array data, etc. Quasi-particles and their interaction is a convenient paradigm in exploring (data/particle) dependencies; indeed, an interaction between particles `p1` and `p2` can be described as `<p1*p2>`, where `<...>` is a proper average; if `<p1*p2> = <p1>*<p2>` , that means that the particles `p1` and `p2` do not interact. Then, the combinations of variables/fields can be sought to *minimize* the interactions using a type of minimization scheme such as

```
min{d(<p1p2>, <p1><p2>)},
```

where `d()` is the appropriate metric and the `min` gets taken over all possible combinations of representative fields.

Such a flexible scheme is the foundation for various *independent* quasi-particle models built directly with data. The methods such as "independent component analysis" can be viewed as a version of data-based quasi-particles. Principal component analysis and projection pursuit algorithm can also be viewed as the versions of the particle-interaction model. Even more detailed 3- and generally N-particle interactions can be introduced and handled programmatically to provide a detailed look into higher-correlation data dependencies.

In its turn, the quasi-particle model was introduced, developed and extensively used in statistical physics and quantum mechanics [Fal]. In fact, thermodynamics was derived as an independent particle approximation to the quantum mechanical description of particle interactions in terms of N-particle density matrix independently by J. von Neumann and L. Landau [N27].

1.2.4 Data Dependencies and Models

So, data dependencies is a critical part of any model building process, and a data pair `p =(mData,data)` is a *unit* of description and analysis of data dependencies. Classical approach to *analytical* model building is to *generalize* the data dependencies in the observation data into an *incremental balance* equation for the variables of interest. For example, the observations over data dependencies `p=(slope,trajectory)` and their generalization as an incremental balance equation for forces have led I. Newton to the differential equations of mechanics.

The similarities between *analytical* model building and the *programmatic* building of a model are obvious - both perform a generalization of data *dependencies* in the dataset, although differently. The traditional model, while built from experimental data, then becomes a construction which exists as symbols (say, on paper) and is totally *detached* from the original data. It is an *offline* model, and for *testing* it needs an *algorithm* to express its internal logic in a computational language, and then a *program* to implement the algorithm computationally.

The data-based model is an *online*, dynamic construction, which is tightly integrated to the data it is built from. Its unique feature is that it is available for online *testing and updating*. This is a new and powerful dimension allowing for dynamic building, testing and modification of a model as well as for integration the massive and complex data streams and their data dependencies in the online model building process. When built, such an online data-driven model is a *mapping engine* which allows for mapping a newly measured input data into the estimated output result, thus providing a prediction capability. Interestingly, it also can be viewed as *search engine* driven by the new data as a "search word", which looks for a *context* related to the input search word, thus providing a description and *interpretation* for a newly measured data.

That *predictive* (estimation) aspect makes a drastic difference with so called *data-mining* techniques aimed at revealing and visualizing hidden patterns and regularities in the data sets which then need to be analyzed and interpreted by *human experts*. Meanwhile, the visualized patterns do not say much about the application-specific structure and dynamics of the system under study and even less about its *behavior* under *new* conditions; that is, data-mining - as we know it - does not include the predictive component, focusing instead on clustering, reordering and visualization of data.

The proposed data exploration approach can also be viewed as an analysis of the system *correlation dynamics* in terms of temporal, spatial and spatio-temporal dependencies of data. For example, the temporal dynamics of each field as well as inter-field temporal dynamics can be analyzed using the field `<t,xy|f[i]>` and record `<t,xy|rec>` `(t,xy)`-dependencies, where <...> is averaging in time and/or in space. The difference dynamics can be measured as a *variance*, and is an indication of system trends in time. Similarly, the trends in space can be estimated using the averages over space. The examples of the trend characteristics are correlations in space and in time such as `<f[i]>`, `<f[i]|f[j]>`, `<r[p],r[q]>`, etc. Histograms - such as `f[i](t)` and/or `n(f[1]..f[n]), dn=1`, would also be a good indication of system trends in time and in space.

A *histogram* is a detector that indicates what a part of the systems states has the specified characteristics – in space, in time, or both. Various type of histograms can be setup *programmatically* to detect specific features of a system under study. A histogram is an example of a *data container*, which has a built-in logic to detect the specific patterns of correlation; say, a state when the 10% of particles in a gas are in the N-particle clusters (N=2,3,...). A histogram *temporal dynamics* (accumulation mode) may be indicative of important trends in the system dynamics (say, convergences – see **Fig.3**).

By detecting and *collecting* that type of stats, one may build up a data analysis system designed to *detect* unique structural and dynamic features and match/relate those with the specific meta-data – temperature, pressure, concentrations, interactions parameters, etc. Within the programmatic data exploration, the *detected features* can always be matched with the *original data dependencies* that drive those features. Thus, the detected features get their natural interpretation in terms of the meta-data involved, a result that helps to *interpret* the measured data.

For example, in an ideal gas, a simple histogram might indicate how many particles `dN` have a velocity `v` (see **Fig.3**). The particle interactions (e.g., attraction) and external parameters (such as temperature, pressure, concentration) would contribute into histogram properties and its temporal dynamics showing cluster formations and other inter-particle correlations.

In atmospheric turbulence analysis, a histogram might indicate how many `z`-layers have a turbulence parameter within a specified range, including the turbulence parameters of various spatial and temporal scale. The dynamics of a histogram change is a powerful indicator or detector of the atmosphere dynamics; various events and/or features of atmosphere behavior can be monitored programmatically to detect the specific factors that impact weather, laser beams propagation and to respond to those factors properly.

Importantly, all those data dependencies get analyzed and generalized within a consistent *data exploration paradigm* which includes a flexible *programmatic model-building pipeline*. The pipeline allows for data movements backward and forward and for the change of stages involved in data analysis and model building.

1.3. Programmatic Data Exploration and Building Data-Driven Models

1.3.1. Data Exploration Tools

In analysis and interpretation of data, the inter-field and inter-record *dependencies* are of primary focus since they can be used as units to build a generic *input-output mapping*. Such a mapping is actually a foundation of *any* predictive model. Two aspects of data exploration are of interest: (1) hidden patterns and regularities to be visualized and then evaluated by a human expert (data-mining aspect) and (2) data-driven models to be used for predictions - mapping new input data.

For example, representation of a field as a time series would be a typical *trend* analysis, say, an incremental averaging a field `f[i]` in time. Also, a variance `Df[i]=(f[i]-<f[i]>)/<f[i]>` can be monitored incrementally; its time dependence may signal changes in the field dynamics which may require triggering certain response actions. A filtering of the `f[i](t)` using a pre-specified filter (mask) is a technique to filter out a (random) noise and thus reinforce a useful signal. For example, a lock-in amplifier-like filter might be an example of filtering out a specific signal component .

Inter-field and inter-record correlations (within a segment of N records) are of interest when looking for *long-term* and *multi-scale* correlations. Of course, a key point in that type of data analysis (as in traditional model building) is the *difference metrics* for various fields and records. With a data-driven approach, it can be flexible and *configurable* depending on the application requirements and criteria.

Programmatic data exploration introduces a more dynamic vision of data and more opportunities in data exploration. For example, *live* data or *data templating* is a powerful technique which allows data to have a *flexible* format and “live” within a *data container* or *template*. The container is a place for data *and* extra logic added to the data to check *dimensionality*, *format*, and any other applicable *properties* of data, and to correct or signal the discrepancies thus performing data quality control functions. The logic may also include some checks if the data (fields and records) satisfy certain "conservation", "continuity", or other rules, laws or relationships. For example, the data template may include logic to check that specific fields (or their certain combinations) are constant (conserved); such as the sum of kinetic and potential energies `E(x,p)` (or total momentum) should be constant in mechanical systems without friction, or the sum of spectral line intensities should be equal to total radiance/energy at-a-sensor.

Examples of fields, records and even the segments of records (sections or pages) *templates* with the built-in logic are: date-time field, random data field, periodic data field `[(sin(t+a)]`, Gaussian data field `[exp(-(t/a)^2)]`, data field with a built-in check for conservation `F(f[1],..,f[N])=const` , and many-many other data field patterns. In some cases, the groups of records (sections or pages) can be identified and placed in proper application-specific containers to check consistency of records in time or against other parameters, features, or requirements.

Placing data into a *flexible container*, a data handler, allows one to consistently check and handle various properties of the data. It makes data a *programmatic or live* object which has a logic to handle the data properties, convert data into various formats, respond to changes in data, etc. That makes data an active component of the model building process and blurs the border between data and model logic. In its turn, programmatic tools such as *object-relational mappers* allow for flexible access and handling of structured data as elements of data structures vs. their standard handling as fields in a table, which is a very limited data access and handling option.

Data containers can be also helpful in the analysis of *metadata* which may have text, numerical, and other components. An interesting area of metadata analysis is the *dynamic generation* of new metadata (descriptions, annotations) in the process of data exploration and model building. In such a process, a new *explanatory dimension* is added to the programmatic data exploration and model building, a unique opportunity in the analysis of *complex real-world* systems not available within a traditional model-based approach.

1.3.2. Data Generalization

The *direct* data-driven model building is basically the finding of *common features* in the data pairs by *data generalization* methods. Each data pair is a *dependence* between the elements of the pair, a relationship `y=f(x)` or `p=(x,y)`. For complex data, such as vectors or matrices, the outer or direct product cam be used to encode dependencies between components expressed in terms of `f[i]*g[j]` products of elements of vectors `f` and `g`. In the realistic data sets, there are *strong* and *weak* dependencies, and certain combinations of the fields can be built - so called eigenmodes - to represent the strongest dependencies.

Generally, the common features in the set of data pairs can be found by the *generalization* operation, a quite common function in computing, which is opposite to differentiation, finding the differences in the data. The metadata help in finding common features by providing the properties of data to be compared in a consistent way.

In simple cases, the generalization can be done by *spatial superimposition* of patterns representing various data pairs which reinforces common features in the dependency patterns. *Neural* generalization by building a spatially-distributed representation of the data pairs is another popular generalization technique. The common features can be explored using various programmatic techniques depending on data and application requirements. Such a direct technique allows for *flexible* data analysis without limitations typical for *specific* models and/or algorithms. So, the available powerful *programmatic* data access, sorting, linking, association, data search, pattern-finding (matching), etc. methods and tools can be used to generalize data pairs and build a predictive mapping (model). This is a fundamental benefit of the *programmatic* approach which allows for flexible and incremental building a trial model, testing and updating it if necessary.

The *data generalization* can be implemented in various ways. A neural network combines context (metadata) and data into a data pair and *generalizes* such data pairs. It maps the data pairs into a *network* of connected nodes with the data-dependent parameters ("weights") attached to the connections and encoding the data dependencies. Many data pairs get mapped into a single neural data model which is a typical generalization operation. Thus such a neural data model may then serve as a *predictive* tool which relates a *new input data* to its context/metadata. In other words, a new data triggers a search in the neural data structure to find a meta data (context) relevant for the new data. Basically, the data generalization and new-data-estimation logic is typical as well for *any* analytical model. An example of a well-designed and flexible neural network tool with proven efficiency is the CLASSER developed at Boston University which utilizes a set of Artmap neural models (for details, see [C06]).

The programmatic implementation of data *generalization* may use any available NN model, or combination of various models, or implement the generalization in a customized, application-specific way. It is a *stage* in the model-building pipeline (see next sections) which provides the benefits of logical power and flexibility, in particular for massive and complex data typical for spectral and hyper-spectral applications.

1.3.3. The Framework

The proposed data exploration occurs within the dedicated *architecture or framework* designed to support the model building process. That creates an infrastructure for model building and makes it a consistent, controlled and automated activity designed for complex data and complex models (vs. a human brain-storming activity typically limited to relatively small datasets and simple models). In addition, the framework holds input data, data dependencies, partial models, and all the other components related to the model-building activities. As the *structure* of data and metadata is an important issue in the generalization and model building process, the framework is actually a *structured container* for the dataset, dependencies encoded in the data pairs and for the model to be built.

The framework is aware of the data properties, and so is able to *evaluate* the input data stream to determine where to put data depending on its *content*. Thus, the data may *structure* the framework and so the model to be built. Data *incrementally* refine and update the model. The model-building framework can be thought of as a new kind of database - structured and capable to generalize the data pairs - elementary dependencies between the features of the application. It is a *generating* database since it is able to estimate the output for the new input. For example, the

hyper-spectral imagery data structure the framework so that all the spectrally-similar pixels (road, grass, etc) get into the proper *nodes* of the framework each collecting pixels with the same spectral signatures.

For prototyping the proposed framework we use the hierarchical database [HDF] which has a well-designed interface [T05] for the interpretive language Python [Py] we use for prototyping our data exploration tools. They will be described in a separate paper.

1.4. Data-Driven Modeling and Predictions

Within data-mining approach, the goal is to discover the hidden regularities and patterns in the data/log files or databases using *data-independent* algorithms to process data - to cluster, visualize, etc. The visual patterns are related indirectly to the application structure and dynamics, however the interpretation of the found regularities in terms of specific application data dependencies is a challenge, since there is no *direct* link between the patterns discovered and the possible *dependencies* that may indicate to the *actions* to be taken as a response to specific patterns.

The proposed approach is aimed to *direct* analysis of *dependencies* in the observation data with the goal to build a *data-driven* predictive model. The model is built by *generalization* of the dependencies in the observation data. An analytical model is also built based on data dependencies. The difference is that the analytical model when built gets *detached* from data and is used as a standalone tool. With such a detachment, the link between the data and the model gets lost and the update of the model with new data becomes a challenge. Meanwhile, the online (data-driven) model allows for easy update and thus for more precise predictions. This *two-way* dynamic link model-data source also allows for *update* the model by *directed* measurements “on demand” - the case when certain data is missing and needs to be measured on-demand to build a quality model. Thus, a *reliable* model can be built in an incremental and controlled manner.

The *metadata* is a critical component of data dependencies and thus of the data-driven model building process. This is an *application-specific* descriptor or annotator to the measurement data, actually to the experiment conducted. By proper design of the experiment, the metadata ideally just get "attached" to the measured data forming an elementary dependence. A data pair is though a *symmetrical* entity, so that a data also points to its metadata thus helping at the search (estimation) phase to find the "right" context for new data. That forward and back symmetry also makes the data-driven model building an interesting alternative to very challenging *inverse* problems.

So, each data `d` has its description, a metadata `m`, and the data never gets separated from its description always forming a data pair `(m,d)`. The metadata `m` includes all the appropriate properties – name, dimensionality, conditions, etc. It also may have a built-in validation or whatever logic. It may also have built-in *links* that get clicked programmatically when/if certain conditions are met.

1.5. Examples of Containers for Live Data

The examples in this section illustrate a new vision of data as *live* data - data integrated with the relevant logic, as well as a new vision of databases (see, for example, [G04]) - databases storing live data and even having predictive capabilities. Data is being elevated to the *properly-structured* data containers with built-in logic. Thus, checking and monitoring data properties *in-situ*, at the location of data, becomes possible, and that bridges a gap between data (usually in a database) and the logic to handle the data (usually in a program).

A generalization logic can be added to a container - depending on the application requirements and the subcontainers can be added to properly *structure* the container. A *virtual* or *generic* container, a generalization of many specific containers can be built using an additional *training* logic. Importantly, the necessary features and properties can be added to a container *at run time* depending on current testing results, an option not available for analytical models. A few examples of containers for live data follow (in the order of increasing complexity)

A function container `x=f(t)` or as a data pair `f=(t,x)`, where `t` is the context for `x` . The container can check and monitor function properties; say, for `x=sin(t)`, property of *periodicity* and the period can be found by scanning data and looking for a pattern `abs(x(t)-x(t+a))< eps`, where `a` is the period to be sought.

A container for difference data (a differential equation). A simple example is `x"(t)+x(t)=0`. This difference equation determines a relation

`x"[n] = -x[n]`, or `x"[n] = x[n+1]-2*x[n]+x[n-1] = -x[n]`

If a `rec=(n,x)` of the dataset satisfies the relation `x″[n]=-x[n]` for *any* three neighboring records, the dataset describes a periodic system (vibrator, oscillator, etc). This container can be used to monitor the periodic properties, like the original analytical model

A container for 1D data. The trajectories, spectra or any other curve is an example of the 1D data. The initial condition: `x0` uniquely determines the trajectory `x(t)`, so they form a data pair `dp=(x0,x(t))`. The container stores the mapping, `db=(dp1,dp2,..,dpN)`, so that a data pair `dp1` can be represented as `db[x10]=x1(t)`, the data pair `dp2=(x20,x2(t))` as `db[x20]=x2(t)`, etc. The mapping `db[x]=y` may reconstruct either a slope or a trajectory.

A container for multi-dimensional data. A few types of multi-dimensional containers can be built.

(a) A set of data-pairs `(x,y),` where, for example, `x` is the wing profile and `y` is the air-flow pattern. The container is actually a database `db[x]=y`, so that any element of a data-pair can be obtained based on other element of the data pair. The advanced data container with *generalizing* capability, a database with predictive capabilities can also be built, so that, for example, a new `y=db[x]` for a new wing profile x can be reconstructed.

(b) A set of data-pairs `(mData,Data)`, where `mData=(t,xyz)`, and the `Data=(p1,p2,..pN)` is the measured properties at the `(xyz)` cell at a time. A few type of containers might be helpful to explore such data and build a predictive model: (1) a container for a z-layer data (`z=const, x,y=var`); (2) a container for a z-profile (`x,y=const, z=var`) for each property `f`; (3) a container for all 3D data (`x,y,z=const`); (4) a network container for many segments of 3D data, a properly-structured 3D network of sensors to keep the data distributed in space.

A container for hyperspectral data. Hyperspectral imagery has a pixel or record as a unit data `rec=(t,xy,R(f))`, where `R(f)` is a reflectance spectrum for the `x,y`-point at time `t,` and `f` is the frequency. The dataset is generated as a *sequence* of pixels measured along the trajectory `(t,xy)`. The *logic* attached to the container might include, for example, a monitoring of a specific line in the R-spectrum assigned to the specific chemical agent of concern. For better precision, a set of spectral lines, a *spectral signature*, can be monitored as well. Also, the application-specific logic can be available to test the presence of the specific line in a current pixel *and in its neighborhood*. That would identify the pixel as belonging to particular class of pixels - a roof pixel, road pixel, etc. Such *application-specific* pixels then can be automatically placed into the proper node of the structured database which mimics the spatial structure of the application.

A container for tabular data. The often-used tabular or spreadsheet format for data is a monolithic table with a measurement as a *row*, the measurement *id* as an access key and many other parameters, *fields*, attached to a row. The container for such data is a simple *mapping* `data[id][f]`, or `data.id.f`, where the `id` is the measurement id, and the `f` is a field (property) for the given measurement. The container allows for a flexible access to *any* component of the data and allows for dynamic reorganization of data based on any property or any mix of those. Also, instead of the monolithic table, the application-specific data unit(s), subcontainer(s), may be introduced to further structure the tabular data. The application-specific data units can be used to look for the *common features* in the data. Another benefit of *live* data container is the possibility to *recalculate* automatically the content triggered by a *change* in data field and/or in data properties.

A pipeline-container for data-driven experiments. Such a container is designed for flexible explorative "experiments" with the data, and defines a kind of *virtual experiment*. The *experiment* container is a flexible data processing *pipeline* with various *configurable* stages such as data evaluation (validation), application-specific data reassembling, data processing stages using various data analysis models, data visualization stages, etc. Such a pipeline is a *container for stages* that can be added, removed, modified, reassembled *programmatically* in the process of data exploration. The pipeline container may have a *control unit* which allows for turning various stages *on* and *off* making each experiment specific and goal-oriented. The database may be integrated as a stage of the pipeline to keep the original, reassembled, processed (simulated) and any other data as well as the built data-driven models for future use in other experiments. The "action" stages can be added to implement the whole *sensors-to-actions* (s2a) pipeline.

Details of building and using containers for specific data types will be given in a separate paper.

1.4. Conclusions

- The proposed approach aims to building *programmatically* an *online* model of a phenomenon using *data dependencies* in the observation data; an analytical model is not necessary.
- Two key stages to build such a model are: (1) building the data pairs each being a *data dependence* `(m,d)`, where `m` is the measurement context or metadata and `d` is the data measured; and (2) *generalization* of the data pairs into an *input-output mapping*, a data-based model.
- A structured database is proposed as a container for the data pairs, and for the online model to be built incrementally by generalization of those data pairs. The estimate for the *new* data is a kind of search (traversing) of the structured database which generates the context related to the new data.
- The methodology and tools, as well as examples of data pairs, tools to build and generalize the data pairs are briefly discussed and demonstrated. Detailed description and the modeling results will be presented in separate papers

Acknowledgements. Author thanks Prof. Simon Berkovich, George Washington University, Dr. Aleks Jakulin of Columbia University, and Prof. Yaneer Bar-Yam, New England Complex Systems Institute, for inspiring discussions on the data-driven model building and the role of data dependencies.

References

[A86] H. Anderson, "Metropolis, Monte Carlo and the MANIAC," *Los Alamos Science* , n.14, p. 69, 1986 (http://lib-www.lanl.gov/la-pubs/00326886.pdf)

[B00] N. L. Balazs, J. C. Browne, J. D. Louck, D. S. Strottman, Nicholas Constantine Metropolis, *Physics Today*, v.53, n.10, p.100, (2000) http://www.aip.org/pt/vol-53/iss-10/p100.html

[B06] V. Bykovsky "Beyond Monte Carlo: Controlled Experimentation with Computer-Generated Prototypes", to be published

[C06] Classer, a neural model package based on artmap neural models, see http://profusion.bu.edu/techlab

[Fal] see, for example, G. Falkovich, "Statistical Physics" http://www.weizmann.ac.il/home/fnfal/statphys.pdf

[G04] J. Gray, “The Revolution in Database Architecture”, Technical Report, Microsoft Research, March 2004, http://research.microsoft.com/~Gray

[HDF] This library was developed by NCSA and successfully used in many National Labs for various *data-intensive* astronomical and space applications, see http://hdf.ncsa.uiuc.edu/

[H83] F. Harlow, N. Metropolis, "Computing and Computers -- Weapons Simulation Leads to the Computer Era," *Los Alamos Science*, n.7, p. 132, 1983 http://lib-www.lanl.gov/la-pubs/00285876.pdf

[M03] W. Matusik, H. Pfister, M. Brand, L. McMillan, "A Data-Driven Reflectance Model", SigGRAPH Conference, 2003
http://people.csail.mit.edu/wojciech/DDRM/ddrm.pdf

[N27] J. von Neumann and E. Wigner, Phys. Z. 30, 467 (1927).

[NCSA] National Supercomputer Applications Center (University of Chicago, Champagne, IL) http://ncsa.uiuc.edu/

[N87] N. Metropolis, “The Beginning of the Monte Carlo Method”, *Los Alamos Science*, Special Issue 1987, pp.125-131 (http://lib-www.lanl.gov/la-pubs/00326866.pdf)

[NSF] for details, see the official NSF site www.nsf.gov/cise/cns/dddas and the public site www.dddas.org

[Py] For details, see the www.python.org

[T05] for detail, see the http://pytables.sf.net, an hierachical database for scientific data (HDF and netCDF format),

[T06] “Type definition and data properties handling package Traits”, Enthought Tools Suite. See http://code.enthought.com/ets/