Capítulo 3 - MÓDULO DE ADMINISTRAÇÃO – ANÁLISES

A plataforma TerraMA² permite o desenvolvimento de análises integrando os dados dinâmicos ambientais com dados estáticos vetoriais ou matriciais. Uma análise envolve a escolha do tipo, dados de entrada, saídas e um programa ("script") para definir como será feita a integração dos dados. Os modelos de análise são escritos na linguagem Python e um conjunto de operadores espaciais criados especialmente para a plataforma.

Todos os tipos de análise produzem novos dados dinâmicos que podem ser reutilizados em outras análises (Figura 3.1). As análises são executadas automaticamente sempre que um dado novo foi coletado, manualmente, por reprocessamento de dados históricos ou a intervalos de tempos pré-definidos pelo usuário. Resumidamente, temos que:

- Nome de análise é único para cada projeto do TerraMA²;
- Um usuário pode criar várias análises combinando os diversos dados;
- Uma análise deve pertencer a apenas um tipo. Ícones são associados quando a análise for baseada em objetos monitorados ( ), baseada em matrizes ( ) ou baseada em PCD ( ).
- Uma análise utiliza dado dinâmico e estático para produzir novos dados dinâmicos;
- Para ver o resultado de uma análise, uma visualização deve ser definida;
- Um alerta poderá ser criado a partir do resultado de uma análise.

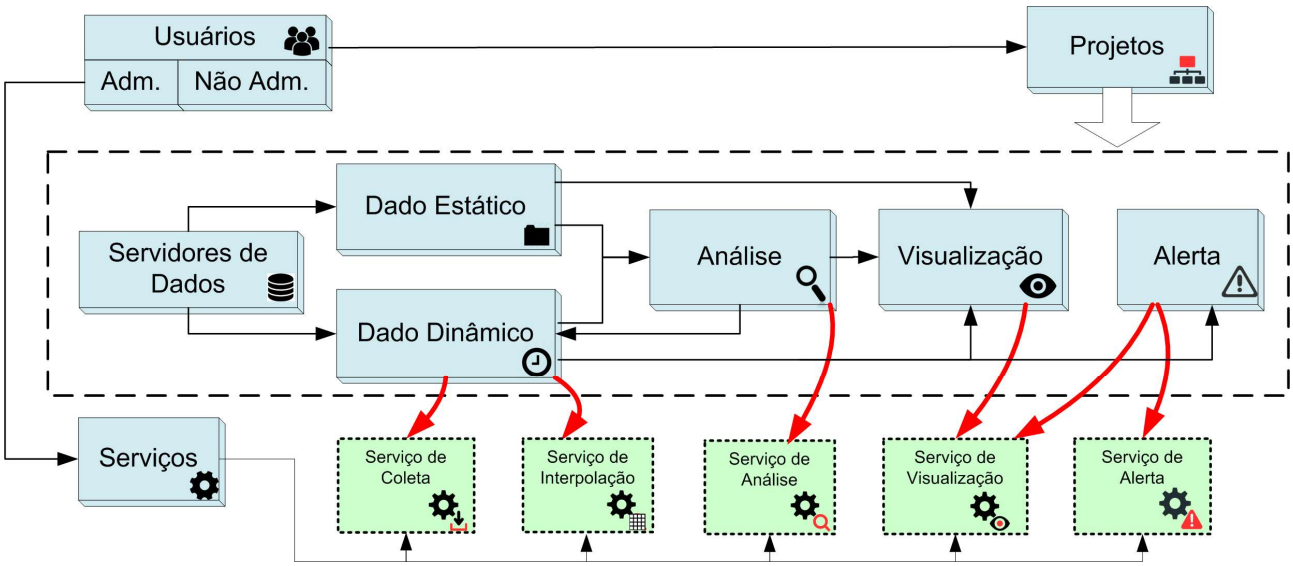


Figura 3.1 – Esquema de um projeto e relações com os serviços. Note que uma análise recebe dados dinâmicos e estáticos produzindo novos dados dinâmicos. O serviço de análise recebe e executa as tarefas.

3.1 – TIPOS DE ANÁLISES

Na TerraMA² são três tipos de análises suportadas. A seguir uma breve descrição dos tipos:

- **Análises baseadas em Objetos Monitorados** : É o principal tipo de análise utilizada. Neste tipo de análise, um dado estático vetorial (identificado como **objeto monitorado**), com representação geométrica de pontos, linhas ou polígonos, é sobreposto a um ou mais dados dinâmicos (grades, PCD ou ocorrências) para produzirem novos dados dinâmicos na forma de uma tabela temporal associada ao mapa de objetos. Essa nova tabela armazena a data/hora de execução da análise e os atributos resultantes dos cálculos da análise. Note que o relacionamento entre a tabela do objeto a tabela resultante da análise é de 1 para muitos (1-n). O esquema apresentado na Figura 3.2 mostra que a análise requer como entrada: um mapa vetorial previamente disponível como dado estático, dados dinâmicos cadastrados e um modelo de análise escrito em Python.

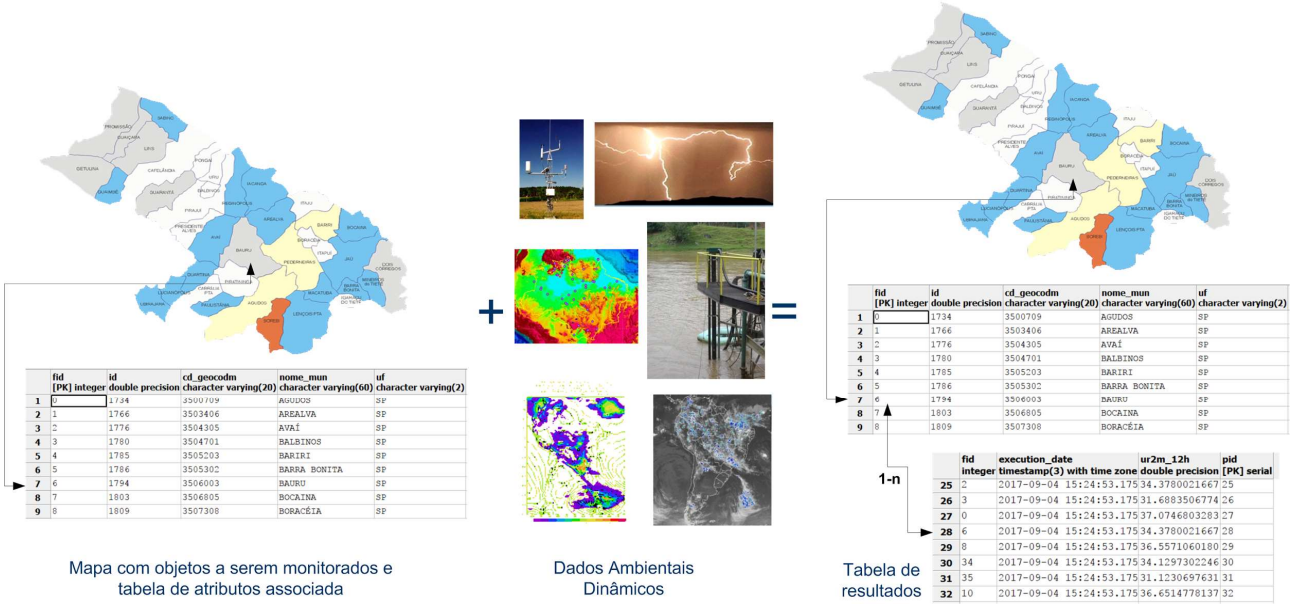


Figura 3.2 – Esquema de uma análise com base em objeto monitorado.

Nota: Além dos recursos da linguagem Python uma lista de operadores zonais (item 3.3.3) podem ser utilizados sobre os dados dinâmicos nos modelos de análise do usuário. Atributos do dado estático também podem ser utilizados nos cálculos. Veja exemplo típico no item 3.3.1. Veja sobre Python no anexo A2, na documentação disponível em português em https://wiki.python.org.br ou no site oficial em https://www.python.org/ .

- **Análises baseadas em grades** : São análises que tem por objetivo a criação de novas grades dinâmicas com base na aplicação de um modelo matemático sobre dados estáticos e dinâmicos, ambos matriciais. O esquema apresentado na Figura 3.3 mostra que a análise requer como entrada; dados estáticos matriciais (não obrigatório), dados dinâmicos matriciais cadastrados (pelo menos um) e um modelo de análise escrito em Python. Como saída gera-se um novo dado dinâmico matricial. Nota-se que o dado matricial resultante pode servir como entrada para a análise com objetos monitorados.

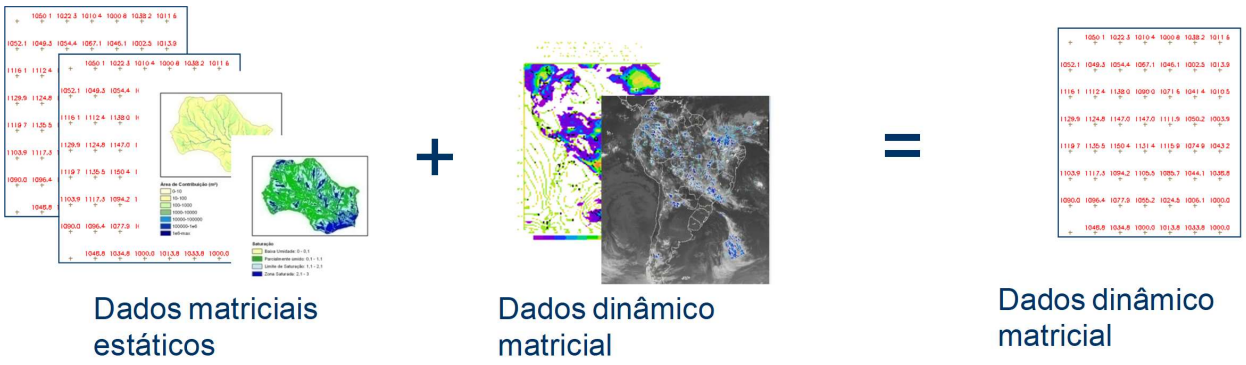


Figura 3.3 – Esquema de uma análise com base em grades.

Nota: Pelo menos um dos dados deve ser dinâmico de modo que a saída será criada na frequência desse dado ou a intervalos fixos definido pelo usuário. Ajustes deverão ser definidos para grades com diferentes resoluções espaciais e diferentes tamanhos. Um conjunto de operadores sobre grades de observação e previsão estão disponíveis (item 3.4.3)

- **Análises baseadas em PCD** : Neste tipo de análise, uma fonte de dados do tipo PCD fornece um conjunto de pontos a serem analisados. Para cada um destes pontos será aplicada individualmente uma regra de análise fornecida pelo usuário para definir um novo valor de atributo. O esquema apresentado na Figura 3.4 mostra que a análise requer como entrada; uma série de dados dinâmicos do tipo PCD e um modelo de análise escrito em Python. Como saída gera-se uma tabela com os resultados da análise.

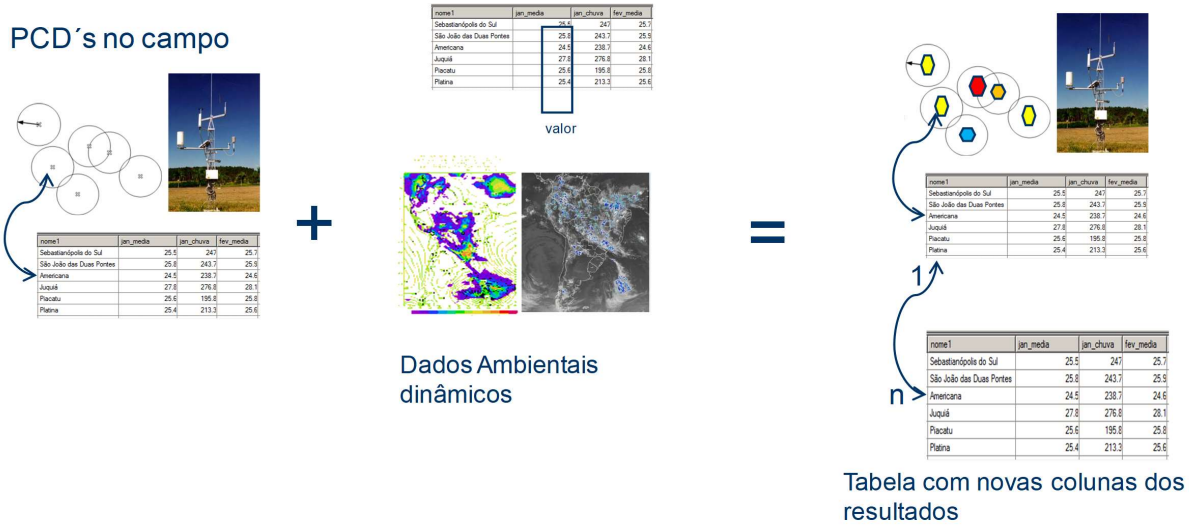


Figura 3.4 – Esquema de uma análise com base em PCD.

Nota: Além dos cálculos realizados sobre os atributos da PCD (item 3.5.3), novos atributos podem ser incluídos como resultado do cruzamento com outros dados dinâmicos, sejam matriciais, PCDs ou ocorrências. Neste caso, as PCD´s podem ser consideradas como objetos monitorados para fazer uso dos operadores zonais disponíveis nesse tipo de análise.

3.2 – MANIPULANDO ANÁLISES

No menu "🔍 **Análises**" o usuário pode criar e gerenciar os modelos de análise a serem aplicados aos dados para posterior geração de alertas. Descrevemos a seguir algumas operações nesse menu que são comuns a qualquer dos tipos de análise descritas acima.

ADICIONANDO UMA NOVA ANÁLISE:

Para adicionar uma nova análise no projeto ativo selecione "🔍 **Análises**" no menu de opções e na área de trabalho correspondente clique no botão " **+** " para adicionar. Após editar todos os campos necessários e o modelo de análise utilize o botão **"Salvar"** ou **"Salvar e Executar"** para salvar e executar a análise. Não é permitido nomes de análises em duplicidade. Botão **"Cancelar"** volta à tela anterior sem salvar a análise. A Figura 3.5 mostra a área de trabalho do menu "🔍 **Análises"**. Note que cada tipo de análise tem um ícone associado. Nesta área é possível selecionar uma análise para fazer alguma edição, criar uma nova ou ainda remover uma análise. Veja a seguir as opções de manipulação.

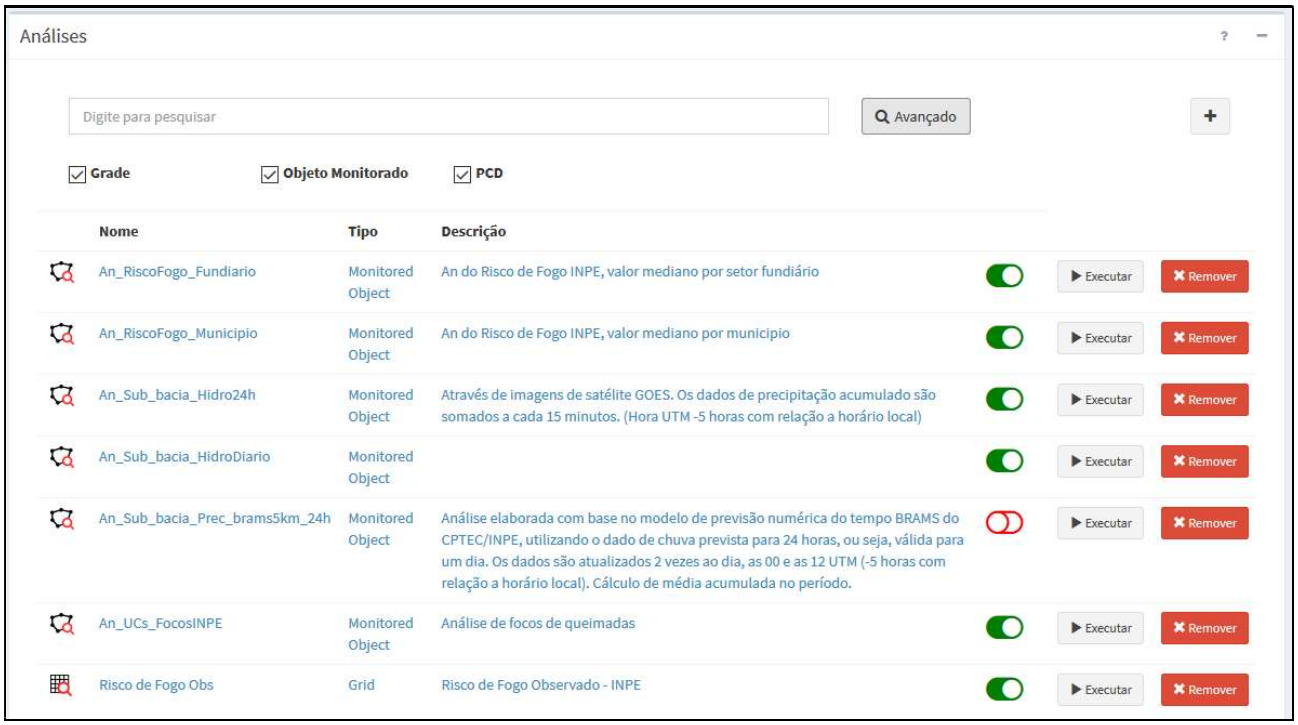


Figura 3.5 - Módulo de Administração: Lista de análises disponíveis na área de trabalho.

Os campos a serem preenchidos dependem do tipo de análise, isto é, se análise baseada em objetos monitorados, se análises baseadas em dados matriciais ou análise baseada nas PCD. Forneça o nome da análise, uma descrição, os dados de entrada, o agendamento para disparar a execução da análise e o programa em Python a ser executado. A descrição detalhada de cada tipo de análise será apresentada mais à frente neste capítulo.

CONSULTANDO E ALTERANDO ANÁLISE:

Para consultar e alterar as configurações de uma análise clique em "**Análises**" no menu de opções e na área de trabalho clique sobre o nome ou tipo de um item disponível. Após editar os campos desejados utilize o botão **"Salvar"** para salvar as alterações. Botão **"Cancelar"** volta à tela anterior sem salvar alterações.

ATIVANDO OU DESATIVANDO UMA ANÁLISE:

Para ativar ou desativar uma análise clique em "**Análises**" no menu de opções e na área de trabalho clique sobre o botão (ativado) ou (desativado) do item correspondente. Análises deixam de ser realizadas na posição desativado. Ao selecionar uma análise qualquer, na aba "Dado Geral" o botão Ativo tem o mesmo efeito dos botões acima apresentados na lista de análises.

FILTRANDO ITENS NA LISTA DE ANÁLISES:

Para filtrar itens na lista de análises clique em "**Análises**" no menu de opções. Na área de trabalho no campo texto "Digite para pesquisar" digite o texto desejado. Note que todas as colunas disponíveis são utilizadas no filtro. Utilize o botão "Avançado" para apresentar os botões referentes aos tipos de análises e escolher os que deverão fazer parte da lista. Por padrão todos os tipos estarão selecionados. O filtro digitado e os botões de escolha do tipo de análise são combinados para apresentar os itens da lista.

IMPORTANDO UMA ANÁLISE:

Para importar a configuração de uma nova análise clique em "**Projetos**" no menu de opções. Na parte superior da área de trabalho clique em " " (botão **Importar**). Na janela apresentada, localize o diretório onde a análise está disponível. Escolha o arquivo "*.terrama2", clique em abrir e selecione o projeto para qual deseja importar os dados.

EXPORTANDO UMA ANÁLISE:

Para exportar a configuração de uma análise utilize a opção do menu "**Projetos**". Na frente do nome do projeto clique em " " (botão **Exportar**). Na janela apresentada desmarque o botão a frente do nome do projeto e abra os itens em "Análises" com o botão " **+** ". Marque ou desmarque o item desejado. Clique no botão **"Exportar"** para confirmar exportação dos itens marcados.

REMOVENDO UMA ANÁLISE:

Para remover uma análise de um projeto ativo clique no menu "**Análises**" para apresentar a lista de análises na área de trabalho. Na frente do nome da análise clique em "**X Remover**". Confirme a remoção na mensagem com "OK".

Atenção: Ao remover uma análise, as visualizações e alertas associados também serão removidos. Neste caso, somente os metadados da análise serão removidos e não há opção de recuperar uma remoção. Já os dados gerados pela análise como grades e tabelas associadas a um objeto monitorado não serão removidos.

3.3 - ANÁLISES BASEADAS EM OBJETOS MONITORADOS

Em geral, análises baseadas em objeto monitorado são realizadas pela sobreposição de dados estáticos vetoriais com um dos três tipos de dados dinâmicos, isto é, grades retangulares, PCD ou ocorrências, que foram coletados, gerados ou apenas disponíveis na base de dados. Resultados de outras análises também podem ser utilizados.

A sobreposição dos objetos monitorados com os dados dinâmicos é realizada com uso dos operadores geográficos (veja item 3.3.3 abaixo) criados exclusivamente para uso na plataforma, juntamente com os recursos que a linguagem de programação Python proporciona.

Os resultados desses operadores são armazenados em variáveis locais do Python, que podem ser armazenadas em uma nova tabela de atributos que estará associada a tabela do objeto monitorado. Nessa nova tabela são armazenados ainda a data/hora toda vez que é realizada a análise.

3.3.1 – EXEMPLO TÍPICO

A seguir apresentamos um exemplo típico da utilização de análise baseada em objeto monitorado que compara um atributo do objeto com o valor máximo de um dado matricial de chuva (Figura 3.6). Assim, no exemplo abaixo considere o seguinte cenário:

1. O objeto monitorado selecionado contém um atributo denominado "**limiar_critico**" que contém um limite de chuva acumulada nas últimas 24 horas a partir do qual deve ser emitido um alerta para a área.

2. A série de dados "**hidro_diario**" contém uma grade com o total de chuva acumulada nas últimas 24 horas.

3. Se a chuva acumulada estiver abaixo do limiar definido para cada objeto, o nível de alerta deve ser 0. Se estiver até 20% acima do limiar, o nível será 1. Chuvas acima de 20% do limiar, o nível será 2.

4. O resultado da análise será armazenado em uma nova tabela de nome "**tab_an_max**" que estará associada a tabela do objeto "**tab_obj**".

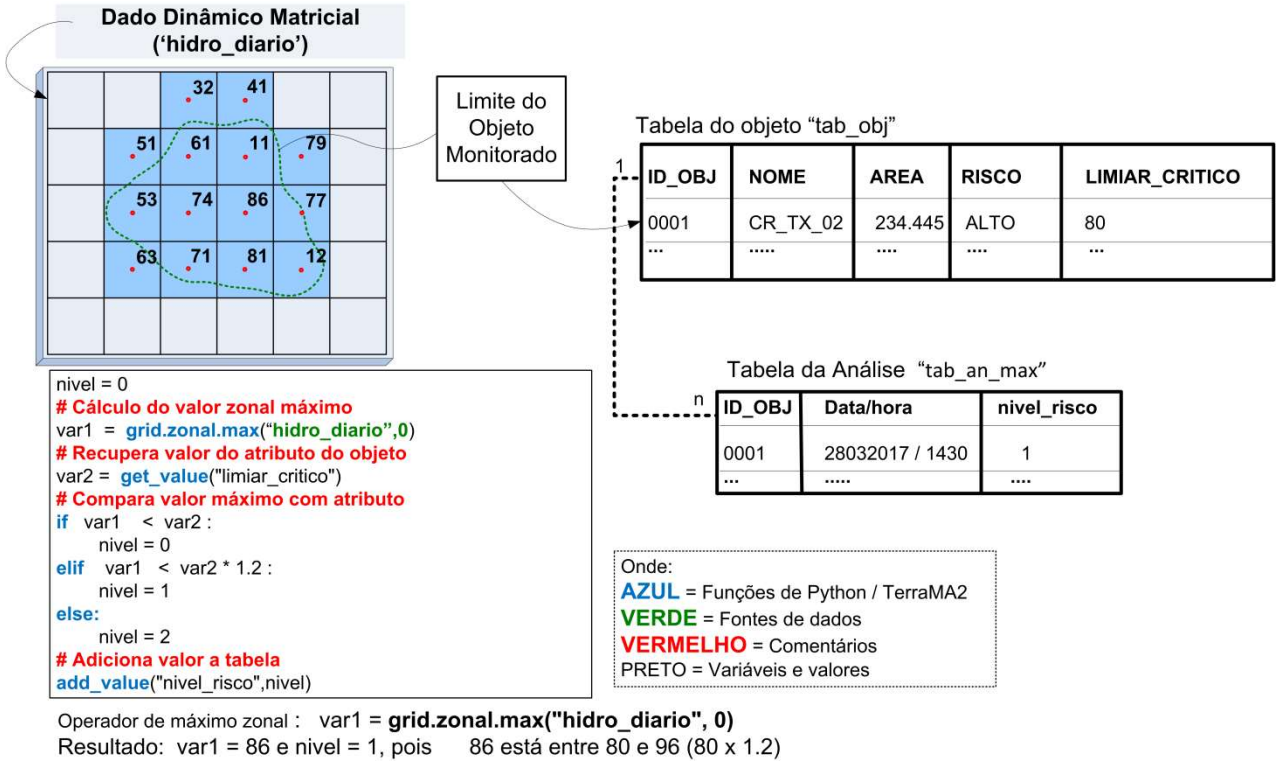


Figura 3.6 – Exemplo de análise com objeto monitorado.

```
# Cálculo do valor zonal máximo
var1 = grid.zonal.max("hidro_diario", 0)
```

Essa linha define uma nova variável denominada de "**var1**" que será inicializada com o valor do operador zonal "**maximo**" aplicado à grade "**hidro_diario**". Na prática, estamos buscando qual o valor de chuva acumulada na área em análise a partir da grade. Mais adiante será feita uma discussão maior sobre operadores zonais. Observe que "**hidro_diario**" é o nome do dado dinâmico que contém as informações de chuva e deve estar cadastrado no projeto.

```
# Recupera valor do atributo do objeto
var2 = get_value("limiar_critico")
```

Essa linha define uma nova variável denominada de "**var2**" que será inicializada com o valor do atributo "**limiar_critico**" de cada objeto monitorado.

```
# Compara valor máximo com atributo
if var1 < var2 :
```

Na linha acima estamos comparando o valor da chuva acumulada obtida da grade com o limiar definido em um objeto em particular, que tem valor 80. Lembrando que a fonte de objeto monitorado pode ter vários objetos (polígonos) com limiares diferentes.

```
    nivel =0
elif var1 < var2 * 1.2:
    nivel = 1
else :
    nivel = 2
```

Nas linhas a seguir estamos estabelecendo os níveis de alerta determinados pela comparação como um limiar pré-definido. Se o valor de chuva acumulada for menor do que o valor do limiar de chuva para esta área, vamos retornar o nível = 0. Seguindo as definições do exemplo, a seguir retornamos nível = 1 se a chuva acumulada

superar o limite em menos de 20%, isto é o valor de 96 (= 80 x 1.2). Caso contrário o valor de nível = 2.

Outro ponto importante a ser observado no exemplo, é a presença de comentários que não tem qualquer significado na execução do programa, apenas para fins de documentação. Use o sinal # para adicionar uma linha de comentário.

```
# Adiciona valor a tabela
add_value("nivel_risco", nivel)
```

A última linha do programa utiliza o comando "**add_value**" da plataforma para adicionar uma nova coluna na tabela "**tab_an_max**" e para armazenar o valor do nível. É obrigatório o uso desse operador pelo menos uma vez.

Para uma maior compreensão da sintaxe da linguagem ou uso de opções avançadas, recomenda-se a leitura do anexo A2, a documentação disponível em português em https://wiki.python.org.br ou site oficial em https://www.python.org/ .

A seguir serão apresentadas em detalhes as opções disponíveis para acesso aos dados da análise, aos atributos do objeto monitorado, operações zonais, zonais históricas ou zonais de previsão.

3.3.2 – EDITANDO ANÁLISE COM OBJETO MONITORADO

A Figura 3.7 mostra a área de trabalho de Análises utilizada para se definir uma análise baseada em objetos monitorados. Descrevemos a seguir cada um dos campos dessa interface.

Figura 3.7 – Módulo de Administração: Análise com base em objeto monitorado.

Registro de Análise – Dado Geral:

- **Nome**: Defina o nome da análise (campo obrigatório). O tamanho máximo do nome é de 100 caracteres. Não é permitido nomes duplicados.
- **Tipo**: Escolha o tipo "**Objeto Monitorado"**. A opção "**Grade**" está descrita no item 3.4. e "**PCD**" no item 3.5. IMPORTANTE: Após salvar a análise o tipo não poderá ser alterado.
- **Descrição**: Campo não obrigatório para descrição da análise. O tamanho máximo do texto é de 250 caracteres.
- **Serviço**: Escolha o serviço de análise que estará associado a análise corrente. Se necessário adicionar novos serviços de análise (local ou remoto) consulte Capítulo 2 – item 2.3.
- **Ativo**: Botão ativo executará a análise de acordo com agendamento (ver abaixo) definido. Se o botão estiver desmarcado a análise não será executada. Uma análise que não esteja ativa poderá ser executada apenas manualmente pelo botão "▶ Executar" disponível na lista de análises da área de trabalho.

Registro de Análise – Armazenar

Utilize os parâmetros desta seção para definir o local de armazenamento dos dados e o agendamento da execução. No caso de uma análise baseada em objeto monitorado, será solicitado o nome de uma tabela de banco de dados.

- **Formato de saída**: Para este tipo de análise apenas a opção "Resultado de análise com Objeto Monitorado" encontra-se disponível. Atenção: após salvar a análise o formato de saída não poderá ser alterado.
- **Nome da tabela**: Digite o nome da tabela a ser criada para armazenar os resultados. Esta tabela terá um relacionamento de "n" para "1" com a tabela do dado estático (ou objeto monitorado). Em outras palavras, toda vez que a análise for executada serão armazenados a data/hora e resultados dos cálculos para cada objeto monitorado.

Atenção: Para nome de tabela de banco de dados NÃO utilize espaços em branco ou caracteres especiais. USE somente caracteres minúsculas e um traço "_" para separar sílabas como "an_grid_mun_max".

NOTA: A tabela resultante da análise armazena de forma contínua criando novos registro a cada execução. Se necessário o administrador do banco de dados poderá remover esta tabela que a mesma será recriada na próxima execução da análise. Recomenda-se desativar a execução da análise antes de remover esta tabela. Se for definido um nome diferente para a tabela, será necessário recriar nova visualização e alerta que utilizam a análise modificada.

Registro de Análise – Armazenar - Agendamento

Nesta seção o usuário deve definir quando será executada a análise.

- **Tipo**: Escolha tipo "**Manual**", "**Agendamento**", "**Reprocessamento de dados históricos**", ou "**Automático**". Se "Manual" a execução da análise só será realizada se o usuário utilizar o botão "► Executar" no item da lista de análises que desejar, ou ainda em "Salvar e executar" da análise aberta. Se "**Agendamento**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua. Se "**Reprocessamento de dados históricos**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua, porém em um período inicial e final no passado. Se "**Automático**" dependerá da chegada de qualquer dos dados dinâmicos que uma análise utilizar.

NOTA: Em todas opções do agendamento a tabela da análise armazena de forma contínua os resultados, exceto em "**Reprocessamento de dados históricos**" que a cada execução da análise os registros serão apagados para que os valores sejam atualizados.

- **Data Inicial** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para início do reprocessamento.
- **Data Final** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para fim do reprocessamento.

- **Unidade de tempo**: Escolha um item entre "Segundos, Minutos, Horas e Semanalmente".

- **Frequência** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Digite um valor de um número inteiro.

- **Tempo Inicial** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Clique no campo para escolher o valor de hora, minuto e segundo que será utilizado como referência para executar a análise.

- **Agendamento** *(somente se Unidade de tempo for Semanalmente)*: escolha uma das opções entre "Domingo, Segunda-feira, Terça-feira, Quarta-feira, Quinta-Feira, Sexta-feira e Sábado"

- **Hora** *(somente se Unidade de tempo for Semanalmente)*: clique no campo para escolher o valor de hora, minuto e segundo que será executada para iniciar a análise.

Registro de Análise – Objeto Monitorado

Nesta seção o usuário deve escolher qual será o objeto monitorado, que corresponde a um dado estático vetorial, com representação geométrica de pontos, linhas ou polígonos, previamente cadastrado (ver Capítulo 2 – item 2.8).

- **Série de Dados**: Escolha um dado estático vetorial previamente cadastrado como dado estático. Atenção: após salvar a análise a série de dados não poderá ser alterada.

- **Atributo Identificador**: Clique no campo e escolha um atributo que será utilizado para identificar os objetos nos relatórios.

Registro de Análise – Dado Adicional

Nesta seção o usuário deve escolher qual ou quais dados estáticos (matriciais somente) ou dinâmicos (PCD, Ocorrência ou Matriz) serão cruzados (ou sobrepostos espacialmente) com as geometrias do objeto monitorado.

- **+ :** Clique no botão para selecionar um dado estático ou dinâmico na janela que será apresentada.

    - **Estático**: Clique para abrir a lista de dados estáticos a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.

    - **Dinâmico**: Clique para abrir a lista de dados dinâmicos a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.

Após a inclusão de um dado na lista, o campo de pseudônimo pode ser alterado. Use o botão "***X*** Remover" a frente do item para excluir um dado da lista (Figura 3.8).

- **Pseudônimo:** Ao escolher um dado a lista automaticamente mostra o mesmo conteúdo do nome para seu pseudônimo. Clique no campo correspondente que deseja alterar. Nas regras de análise serão os pseudônimos que deverão ser utilizados pelos operadores (ver item 3.3.1.3). Lembre-se que a plataforma faz distinção entre maiúsculas e minúsculas.

Figura 3.8 – Módulo de Administração: Análise – Lista de Dados Adicionais

Registro de Análise – Programa

Nesta seção o usuário deve editar o programa de análise. A edição do programa utiliza a linguagem Python, portanto, siga rigorosamente a sintaxe dos comandos definidos para esta linguagem. Além dos comandos e funções de Python você pode utilizar os utilitários e os operadores zonais criados especialmente para a plataforma TerraMA².

Para facilitar a edição do programa, botões na parte inferior da janela possibilitam escolher atalhos de alguns itens específicos. Ao escolher um item entre os botões disponíveis, o conteúdo será incluído na posição em que estiver o cursor. Os atalhos disponíveis são:

- Atalho para os utilitários da plataforma, tais como “buffer” a ser aplicado em geometrias do objeto monitorado, unidades de distância, tempo, “add_value”, “get_value” e “get_analysis_date”.

- Atalho para os atributos do objeto monitorado (ou dado estático vetorial) que foi escolhido para a análise atual. Junto do nome do atributo o utilitário “get_value” será inserido, pois este deve ser utilizado para ser atribuído a uma variável do programa.

- Atalho para os operadores que trabalham com dados dinâmicos de PCD. Contém uma lista de operadores que retornam regras de influência das PCD, agrupados em operadores **zonais**, **zonais históricos** e **zonais históricos por intervalo**.

- Atalho para os operadores que trabalham com dados dinâmicos matriciais. Contém uma lista de operadores que utilizam dados de observação e previsão, agrupados em **zonais**, **zonais históricos** e **zonais de previsão**.

- Atalho para os operadores que trabalham com dados dinâmicos de ocorrências. Contém uma lista de operadores históricos agrupados em **zonais**, **zonais por agregação** e **zonais por intervalo**.

 - Atalho para algumas funções, operadores e comandos de Python. Outros recursos veja o anexo A2.

Após editar o programa, poderá utilizar o botão "Validar" para identificar se há erros de sintaxe nos comando, funções e operadores utilizados. O botão "Salvar e executar" grava as últimas alterações e executa a análise mesmo que esta esteja inativa. Se desejar apenas gravar as alterações clique na seta do botão e escolha "Salvar". Para as análises que estiverem ativas, as próximas execuções seguirão as regras definidas na seção "Agendamento".

Importante: O programa de análise definido pelo usuário é executado individualmente para cada geometria do objeto monitorado. É obrigatório que o programa faça uso pelo menos uma vez do utilitário "add_value" para que seja definido um atributo da tabela de análise. Este utilitário pode ser utilizado dentro de um comando condicional (if) para adicionar valores resultantes da análise somente nos objetos de interesse do usuário (veja exemplo no item 3.3.3.1 - I.5).

3.3.3 – OPERADORES PARA ANÁLISE COM OBJETO MONITORADO

Os operadores espaciais disponíveis para serem utilizados com os objetos monitorados são baseados em operações zonais. **Operadores zonais** são funções que permitem obter de uma grade (ou dado matricial) um único valor que melhor represente todos os pontos da grade que caem sobre uma geometria (ponto, linha ou polígono) ou a uma distância fixa dessa geometria (buffer). A Figura 3.9 ilustra exemplos de operações zonais de um polígono de um objeto monitorado sobre os dados de ocorrência, PCD e matriz.

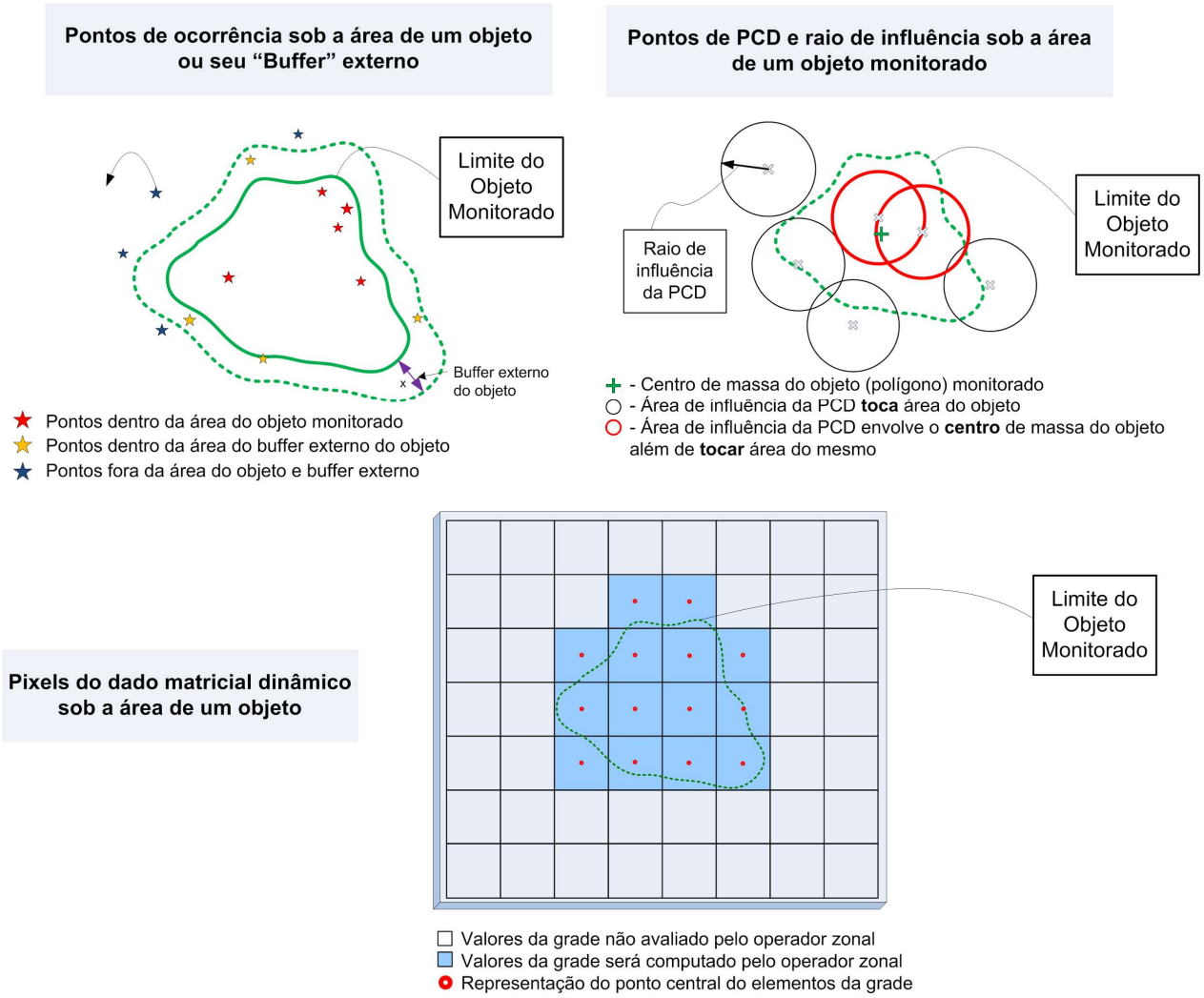


Figura 3.9 – Exemplo dos operadores zonais com dados de ocorrência, PCD e matriz.

Nos operadores há parâmetros que são obrigatórios e outros opcionais. Aqueles que são opcionais são identificados por colchetes [ ] quando são escolhidos pelos botões de atalho no final da janela de edição do programa de análise. Tais parâmetros estão sempre no final da lista de cada operador e se não forem utilizados serão considerados os valores padrões de cada um. No exemplo abaixo os parâmetros [<band>] e [<buffer>] são opcionais e neste caso se não declarar o [<buffer>] será considerada a própria geometria do objeto a ser monitorado e no caso do [<band>] será considerada a banda 0 (zero) de um dado dinâmico matricial.

grid.zonal.min("<dynamic_data_grid>", [<band>], [<buffer>])

Nas três linhas do operador "grid.zonal.min" abaixo o resultado é o mesmo, pois o [<band>] e [<buffer>] são utilizados como valores padrão ou não são declarados.

Exemplo: b1 = Buffer()
x1 = grid.zonal.min("Chuva", 0, b1)
ou

x1 = grid.zonal.min("Chuva", 0)

ou

```
x1 = grid.zonal.min("Chuva")
```

No exemplo, se o [<buffer>] for definido diferente do padrão, o [<band>] deve ser obrigatoriamente definido mesmo que se faça uso do padrão. Veja no exemplo a seguir que o [<band>] deve ser declarado.

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 50, "cm")
x1 = grid.zonal.min("Chuva", 0, b1)
```

3.3.3.1- Utilitários para operadores sobre objetos monitorados

Antes de apresentar os operadores espaciais propriamente dito, veremos um conjunto de utilitários que podem ser utilizados junto com os operadores.

I.1- Unidade de distância

Os operadores que utilizam unidades de distância devem usar as letras entre aspas duplas ("<unidade>"). As seguintes opções estão disponíveis:

- "cm": centímetros
- "m" : metros
- "km": quilômetros

Exemplo de uso no operador "buffer":

```
buffer1 = Buffer(BufferType.Out_union, 50, "cm")
buffer2 = Buffer(BufferType.Level,  400, "m", 200, "m")
buffer3 = Buffer(BufferType.In_out,  200, "km", 200, "km")
```

I.2- Unidade de tempo

Os operadores que utilizam unidades de tempo devem usar a unidade imediatamente após o valor numérico, ambos entre aspas dupla ("<num><unidade>"). As seguintes opções estão disponíveis:

- s: Second – tempo em segundos a partir da data/hora atual.
- min: Minute – tempo em minutos a partir da data/hora atual.
- h: Hour – tempo em horas a partir da data/hora atual.
- d: Day – tempo em dias a partir da data/hora atual.
- d+: Day (Extended) – tempo em dias a partir da data/hora atual até a zero horas do número de dias informado.
- w: Week – tempo em semanas a partir da data/hora atual.
- w+: Week (Extended) – tempo em semanas a partir da hora atual até a zero horas do número de semanas informado.

A Figura 3.10 mostra a diferença ao utilizar o utilitário de unidade de tempo "d" e "d+" em dois horários diferentes, as 7 e as 11 horas da manhã. Supondo que estes utilitários fossem aplicados dois operadores históricos que olham para o passado a

partir das 7 horas da manhã, note que a quantidade de horas retornada por cada utilitário é diferente. No utilitário "1d" sempre retorna 24 horas de tempo passado. Já o utilitário "1d+" retorna um tempo maior de 31 horas de tempo passado, pois considera o intervalo de tempo da 0 hora do dia anterior até a hora atual.

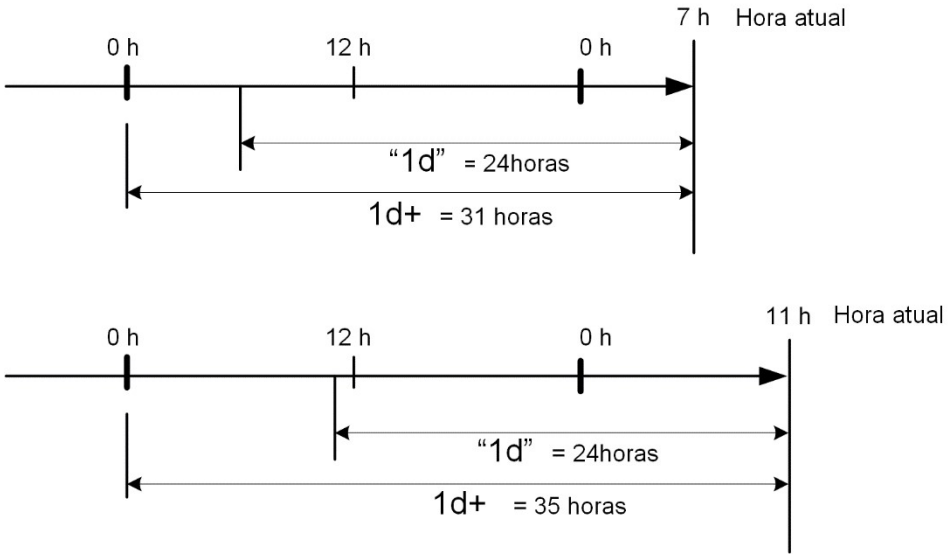


Figura 3.10 – Exemplo do uso de diferentes unidades de tempo "d" e "d+".

Na mesma Figura 3.10 supondo que estes utilitários fossem utilizados em dois operadores históricos a partir das 11 horas da manhã, o intervalo de horas é diferente para ambos utilitários. O utilitário "1d" retorna o mesmo intervalo de 24 horas de tempo passado da situação anterior (referência das 7 horas da manhã). Já o utilitário "1d+" retorna um intervalo de tempo maior de 35 horas de tempo passado, comparado a situação anterior (referência das 7 horas da manhã).

Estes utilitários de tempo podem ser utilizados para operadores históricos (análise do tempo no passado) ou para operadores de previsão (análise do tempo no futuro) a partir da data/hora atual.

Exemplo de uso em alguns operadores que fazem uso da unidade de tempo:

```
x1 = occurrence.zonal.count("ocorrencias", "120s", buf1, "UF = 'AM'")
x2 = occurrence.zonal.interval.max("focos", "30min", "10min", "Intensidade", buf1)
x3 = dcp.zonal.history.min("Serra do Mar", "Pluvio", "48h", ids)
x4 = grid.zonal.forecast.median("ETA15km", "3d+", buffer_mun)
x5 = grid.zonal.history.accum.min("hidro", "1w", 0, buffer_reg)
x5 = grid.zonal.history.mean("hidro", "5w+")
```

I.3- Utilitário de "Buffer"

Todas as análises que trabalham com objetos monitorados, isto é, um mapa vetorial com geometrias de pontos, linhas ou polígonos, pode-se optar por utilizar o utilitário "buffer" para definir áreas com equidistâncias dessas geometrias. A área definida pelo "buffer" será sobreposta aos dados dinâmicos, fazendo uso dos operadores zonais. Os tipos de "buffer" disponíveis são:

- **Buffer( )** : Sem buffer. Será considerada a própria geometria do ponto, linha ou área do polígono (Figura 3.11).

Figura 3.11 – Sem "buffer" no ponto, linha e área do polígono

Exemplo sem buffer:

```
b1 = Buffer( )                    ou
b1 = Buffer(BufferType.None)
```

NOTA: Operadores onde o "buffer" não é um parâmetro obrigatório e sendo este o último parâmetro no operador, basta omitir tal parâmetro para não utilizar "buffer". No exemplo abaixo, as variáveis x1, x2 e x3 produzirão o mesmo resultado com o operador "grid.zonal.mean", pois ambos não utilizam "buffer". Na variável x3 ainda é possível omitir o valor 0 considerado padrão para a primeira banda/camada de um dado matricial.

```
b1 = Buffer( )
x1 = grid.zonal.mean("Chuva", 0, b1)
            ou
x2 = grid.zonal.mean("Chuva", 0)
            ou ainda
x3 = grid.zonal.mean("Chuva")
```

- **BufferType.Out** : Somente a área do buffer externo. Será considerada somente área do "buffer" externa à geometria do ponto, linha e limite do polígono (Figura 3.12).

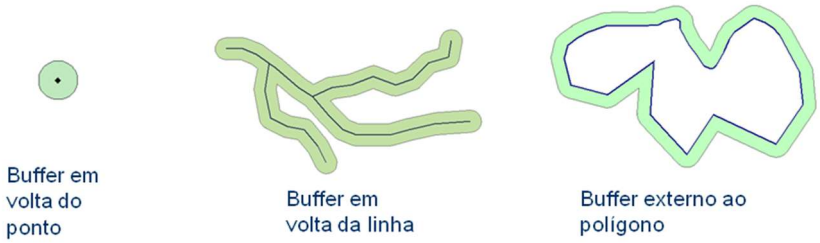


Figura 3.12 – "Buffer" externo ao ponto, linha e limite do polígono

Exemplo do buffer externo:

```
b1 = Buffer(BufferType.Out,  200, "m")
```

- **BufferType.In** : Somente a área do buffer interno. Será considerada somente área do "buffer" interno à geometria do polígono (Figura 3.13). Para geometrias de ponto e linha considera-se ausência de "buffer".

Figura 3.13 – "Buffer" interno ao ponto, linha e limite do polígono

Exemplo do buffer interno:

```
b1 = Buffer(BufferType.In,  200, “m")
```

- **Buffer.Type.In_out** : Área total do “buffer” interno e externo. Será considerada a união das áreas interna e externa do “buffer” para a geometria de ponto, linha e limite do polígono (Figura 3.14). Para geometrias de ponto e linha considera-se apenas o “buffer” externo.

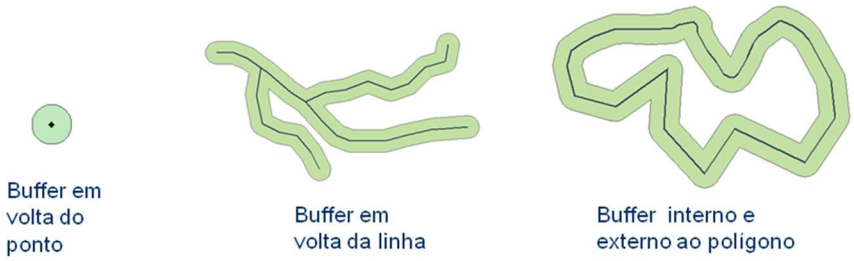


Figura 3.14 – “Buffer” interno e externo ao ponto, linha e limite do polígono

Exemplo do buffer interno externo:

```
b1 = Buffer(BufferType.In_out,  200, "m", 200, "m")
```

- **Buffer.Type.Out_union** : Área do “buffer” externo mais área da geometria. Será considerada a união da área do “buffer” externo mais toda área da geometria quando polígono (Figura 3.15). Para geometrias de ponto e linha considera-se apenas o “buffer” externo.

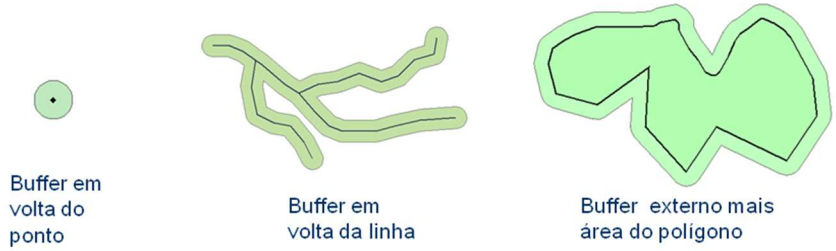


Figura 3.15 – “Buffer” externo somada a área da geometria de ponto, linha e limite do polígono

Exemplo do buffer interno externo:

```
b1 = Buffer(BufferType.Out_union,  200, "m")
```

- **Buffer.Type.In_diff** : Área da geometria menos a área do buffer interno. Será considerada a área da geometria menos a área do buffer interno quando polígono (Figura 3.16). Para geometrias de ponto e linha considera-se ausência de “buffer”.

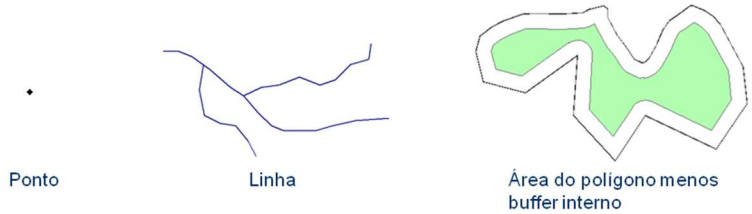


Figura 3.16 – Área da geometria menos “buffer” interno da geometria de ponto, linha e limite do polígono

Exemplo da diferença do buffer interno:

```
b1 = Buffer(BufferType.In_diff,  200, "m")
```

- **Buffer.Type.Level** : Área diferença entre dois buffer externos. Será considerada a diferença entre um buffer maior menos o menor, definindo uma área não adjacente a geometria utilizada (Figura 3.17). O primeiro valor do buffer deve ser obrigatoriamente o mais distante.

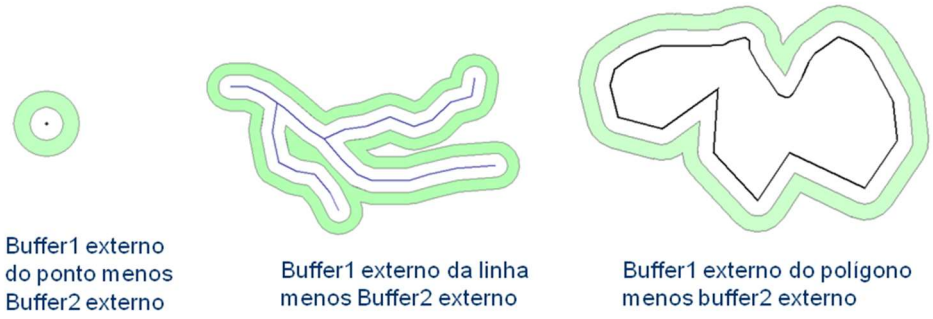


Figura 3.17 – Diferença entre dois "buffer" externos da geometria de ponto, linha e limite do polígono

Exemplo de níveis do buffer externo:

```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
```

I.4- Utilitários "Get Value"

Utilitário para acesso aos atributos de um objeto monitorado. Válido para atributos numéricos ou alfanuméricos.

Sintaxe:

```
get_value("<attribute_name>")
```

Exemplo: `var1 = get_value("vulnerabildade")`

NOTA: Na interface de edição do modelo de análise há um atalho para os atributos do objeto monitorado  . Ao escolher o atributo, o utilitário "get_value" será inserido automaticamente.

I.5- Utilitários "Add Value"

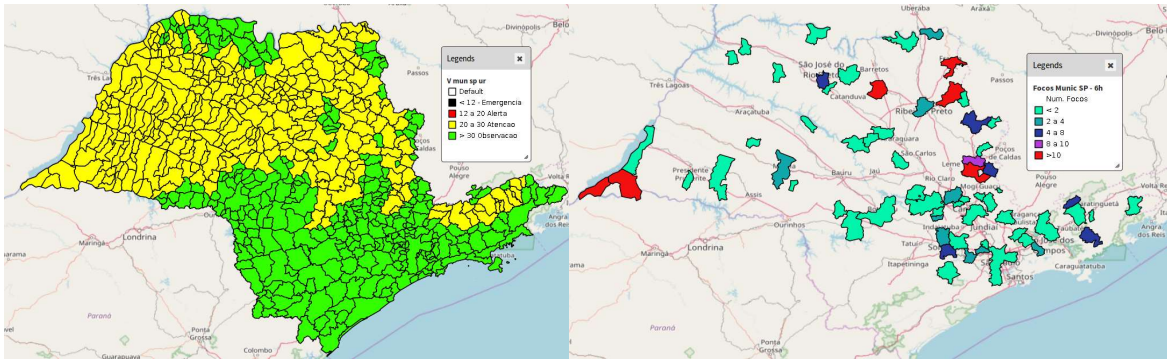
Utilitário para adicionar o valor de uma variável a um atributo na tabela resultante de uma análise baseada em objeto monitorado ou de PCD. Válido para atributos numéricos ou alfanuméricos. É obrigatório pelo menos uma vez o uso desse operador. Se nenhuma condição for atribuída ao utilitário, todos os objetos monitorados receberão um valor mesmo que este seja nulo. Para evitar o armazenamento de valores nulos ou que não tenham significado prático para um alerta, utilize o comando condicional (if) para restringir quais objetos receberão os resultados desse utilitário. A Figura 3.18 mostra como ficará a visualização de um mapa de municípios de São Paulo sem restrição, onde todos municípios são apresentados com uma cor, e com restrição, onde somente parte dos municípios serão **apresentados.**

Sintaxe:

```
add_value("<attribute_name>", <value>)
```

Exemplo:

```
var1 = 10
add_value("vulnerabildade", var1)          # sem restrição
if  var1 > 0:
        add_value("vulnerabildade", var1)    # com restrição
```



(a) (b)

Figura 3.18 – Exemplo da visualização de duas análises, sem (a) e com (b) restrição no uso do utilitário "add_value".

I.6- Utilitários "Get analysis date"

Utilitário que retorna a data/hora (*datetime*) de execução da análise, seja de uma análise em tempo real com valor de data/hora atual ou de um reprocessamento de dado histórico com data/hora no passado:

Sintaxe:

get_analysis_date()

Exemplo: var1 = get_analysis_date()

Nesse exemplo a variável var1 recebe o conteúdo de data/hora com "time zone" no formato "**2018-06-15T23:11:11.876Z**". Para extrair parte do conteúdo de um "*datetime*" na forma numérica referente a data/hora de execução de uma análise, parâmetros como "year, month, day, hour, minute, second e microsecond" podem ser utilizados como nos exemplos a seguir.

Exemplos:

var2 = add_analysis_date().year : retorna o valor do ano entre 1 e ano atual.
var3 = add_analysis_date().month : retorna o valor do mês entre 1 e 12.
var4 = add_analysis_date().day : retorna o valor do dia entre 1 e 31 dependendo no mês e do ano.
var5 = add_analysis_date().hour : retorna o valor da hora entre 0 e 23.
var6 = add_analysis_date().minute : retorna o valor do minuto entre 0 e 59.
var7 = add_analysis_date().second : retorna o valor do segundo entre 0 e 59.
var8 = add_analysis_date().microsecond : retorna o valor da micro-segundo entre 1 e 1000000.

Para extrair parte do conteúdo de um "*datetime*" na forma numérica referente a data/hora do sistema operacional, independente do horário de execução da análise, utilize "datetime" do Python com os mesmos parâmetros como "year, month, day, hour, minute, second e microsecond", como no exemplo a seguir, acrescido do parâmetro "today()" ou "now()".

Exemplos:

var9 = datetime.date.today().year : retorna o valor do ano entre 1 e ano atual.

var10 = datetime.datetime.now().month : retorna o valor do mês entre 1 e 12.

Outra opção que pode ser utilizada á a função “strftime()” que retorna um “string” referente a data/hora do sistema operacional. Os exemplos a seguir mostram as opções de parâmetros para esta função.

Exemplos:

import time	: necessário para usar a função “strftime()”.
var11 = time.strftime("%Y")	: retorna o “string” do ano entre 0001 e ano atual.
var12 = time.strftime("%m")	: retorna o “string” do mês entre 01 e 12.
var13 = time.strftime("%d")	: retorna o “string” do dia entre 01 e 31 dependendo no mês e do ano.
var14 = time.strftime("%H")	: retorna o “string” da hora entre 00 e 23.
var15 = time.strftime("%M")	: retorna o “string” do minuto entre 00 e 59.
var16 = time.strftime("%S")	: retorna o “string” do segundo entre 00 e 59.

I.7- Funções estatísticas para agregação

Para agregação de valores será útil fazer uso das funções de estatística, tais como:

- Statistic.min: valor mínimo de uma lista de valores
- Statistic.max: valor máximo de uma lista de valores
- Statistic.mean: valor médio de uma lista de valores
- Statistic.sum: soma de uma lista de valores
- Statistic.mean: média de uma lista de valores
- Statistic.standard_deviation: desvio padrão de uma lista de valores

3.3.3.2 - Operadores zonais de Ocorrências

Operadores zonais de ocorrências são operadores utilizados para obter estatísticas sobre as fontes de dados do tipo de ocorrências. Consideram a localização dos pontos de ocorrências e seus atributos que interceptam as geometrias (pontos, linhas ou polígonos) dos objetos a serem monitorados ou a área de influência (buffer) dessas geometrias num intervalo de tempo passado.

A Figura 3.19 mostra os pontos de ocorrência em relação a um objeto monitorado na forma de polígono. Ocorrências podem estar dentro do polígono, na área do “buffer” (depende do tipo de utilitário “buffer” utilizado) ou não fazem interseção espacial com o polígono nem o seu “buffer”. Veremos um grupo de operadores onde um segundo “buffer” poderá ser utilizado sobre cada ocorrência para verificar se há pontos de ocorrências sobrepostos e assim considerar como uma única.

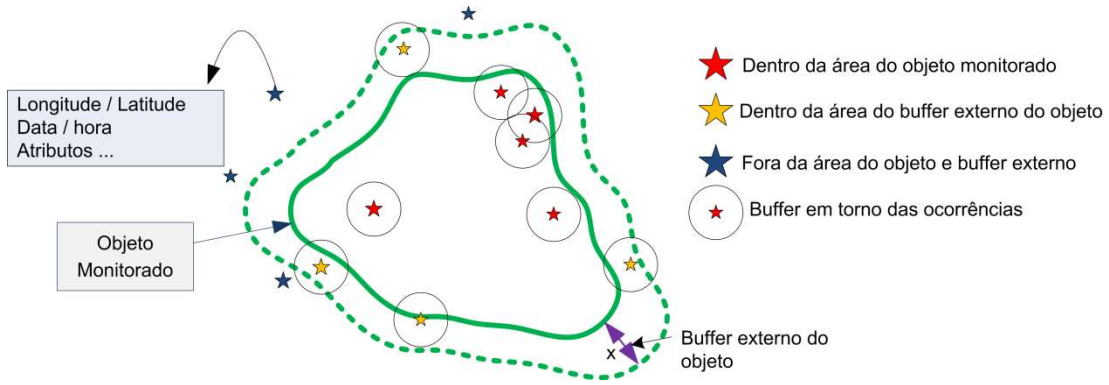


Figura 3.19 – Diferentes situações de ocorrências em relação a um objeto monitorado.

Estes operadores são divididos em três grupos: **Zonal**, **Zonal por intervalos** e **Zonal por agregação**. A descrição de cada tipo a seguir.

II.1- Zonal

Grupo de operadores que consideram as ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

occurrence.zonal.<operator>("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

onde:

- **operator**: count, min, max, mean, sum, median, standard_deviation, variance
- **dynamic_data_occurrence**: String com o nome da série de dados de ocorrências;
- **attribute**: String com o nome do atributo da ocorrência que deve ser utilizado para recuperar algum valor associado a ocorrência. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador “count”;
- **time**: String com o intervalo de tempo, a partir da hora atual, para filtrar as ocorrências. Este intervalo será aberto (< x) no valor informado e fechado (=) na hora atual. Ver utilitário unidades de tempo;
- **buffer**: [Opcional] “Buffer” para ser aplicado ao objeto monitorado. Parâmetro obrigatório somente se o parâmetro seguinte (restriction) for utilizado. Se não declarado será considerada a própria geometria do objeto (no caso do um mapa de polígonos, a própria área do polígono é utilizada). Ver utilitário “buffer” acima;
- **restriction**: [Opcional] String com a restrição SQL a ser aplicada sobre atributos do dado dinâmico de ocorrência. Não utilizar se não houver restrição. Se for utilizada a restrição, obrigatoriamente o parâmetro de buffer acima deve ser declarado, mesmo que o buffer seja nulo.

Segue a descrição de cada operador.

Zonal: **Contagem**

Retorna a quantidade de ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.count("<dynamic_data_occurrence>", "<time>", [<buffer>], ["<restriction>"])

Exemplo:

```
# exemplo com buffer e com restrição
buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
x = occurrence.zonal.count("ocorrencias", "1d", buf1, "UF = 'AM'")

# exemplo sem buffer e sem restrição
x = occurrence.zonal.count("ocorrencias", "1d+")

# exemplo sem buffer e com restrição
buf1 = Buffer()
x = occurrence.zonal.count("ocorrencias", "1d", buf1, "UF = 'AM'")
# ou
x = occurrence.zonal.count("ocorrencias", "1d", Buffer(), "UF = 'AM'")
```

Zonal: **Mínimo**

Retorna o menor valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.min("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo:

```
buf1 = Buffer()
x = occurrence.zonal.min("foco", "intensidade","1d",  buf1, "UF = 'AM'")
```

Zonal: **Máximo**

Retorna o maior valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.max("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo:

```
buf1 = Buffer()
x = occurrence.zonal.max("ocorrencias", "Intensidade", "1d", buf1, "UF = 'AM'")
```

Zonal: **Média**

Retorna a média dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.mean("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer()
x = occurrence.zonal.mean("ocorrencias", "intensidade", "1d", buf1, "UF = 'AM'")

Zonal: **Mediana**

Retorna a mediana dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.median("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer()
x = occurrence.zonal.median("ocorrencias", "intensidade", "1d", buf1, "UF = 'AM'")

Zonal: **Soma**

Retorna a soma dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.sum("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer()
x = occurrence.zonal.sum("ocorrencias", "intensidade", "1d", buf1, "UF = 'AM'")

Zonal: **Desvio Padrão**

Retorna o desvio padrão dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.standard_deviation("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer(BufferType.Level, 400, “m“, 200, “m”)
x = occurrence.zonal.standard_deviation("ocorrencias", "intensidade", "1d", buf1)

Zonal: **Variância**

Retorna a variância dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.variance("<dynamic_data_occurrence>", "<attribute>", "<time>", [<buffer>], ["<restriction>"])

Exemplo: x = occurrence.zonal.variance("ocorrencias", "1d", "Intensidade")

II.2- Zonal por intervalos

Grupo de operadores que consideram as ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre dois valores de tempo informado no passado.

SINTAXE GERAL:

occurrence.zonal.interval.<operator>("<dynamic_data_occurrence>", "<attribute>", "<time_begin>", "<time_end>", [<buffer>], ["<restriction>"])

onde:

- **operator** : count, min, max, mean, median, sum, standard_deviation, variance
- **dynamic_data_occurrence** : String com o nome da série de dados de ocorrências;
- **attribute**: String com o nome do atributo da ocorrência que deve ser utilizado para recuperar os valores, o atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador "count";
- **time_begin**: String inicial (mais antigo) do intervalo de tempo para filtrar as ocorrências. Este valor será aberto (< tempo mais antigo) no tempo informado;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar as ocorrências. Este valor será fechado (<= tempo mais recente) no tempo informado;
- **buffer**: [Opcional] "Buffer" para ser aplicado ao objeto monitorado. Parâmetro obrigatório somente se o parâmetro seguinte (restriction) for utilizado. Se não declarado será considerada a própria geometria do objeto (no caso do um mapa de polígonos, a própria área do polígono é utilizada). Ver utilitário "buffer" acima;
- **restriction**: [Opcional] String com a restrição SQL a ser aplicada sobre atributos do dado dinâmico de ocorrência. Não utilizar se não houver restrição. Se for utilizada a restrição, obrigatoriamente o parâmetro de buffer acima deve ser declarado, mesmo que o buffer seja nulo.

Segue a descrição de cada operador.

Zonal por intervalo: **Contagem**

Retorna a quantidade de ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

occurrence.zonal.interval.count("<dynamic_data_occurrence>", "<time_begin>", "<time_end>", [<buffer>], ["<restriction>"])

Exemplo:

```
# exemplo com buffer e com restrição
buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
x = occurrence.zonal.interval.count("ocorrencias", "2d", "1d", buf1, "UF = 'AM'")

# exemplo sem buffer e sem restrição
x = occurrence.zonal.interval.count("ocorrencias", "2d", "1d")

# exemplo sem buffer e com restrição
buf1 = Buffer()
x = occurrence.zonal.interval.count("ocorrencias", "2d", "1d", buf1, "UF = 'AM'")
# ou
```

```
x = occurrence.zonal.interval.count("ocorrencias", "2d", "1d", Buffer(), "UF = 'AM'")
```

Zonal por intervalo: **Mínimo**

Retorna o menor valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
occurrence.zonal.interval.min("<dynamic_data_occurrence>", "<attribute>", "<time_begin>",
                              "<time_end>", [<buffer>], ["<restriction>"])
```

Exemplo:
```
buf1 = Buffer()
x = occurrence.zonal.interval.min("raios", "Intensidade", "4h", "1h", buf1, "UF = 'AM'")
```

Zonal por intervalo: **Máximo**

Retorna o maior valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
occurrence.zonal.interval.max("<dynamic_data_occurrence>", "<attribute>", "<time_begin>",
                              "<time_end>", [<buffer>], ["<restriction>"])
```

Exemplo:
```
buf1 = Buffer()
x = occurrence.zonal.interval.max("raios", "Intensidade", "4h", "1h", buf1, "UF = 'AM'")
```

Zonal por intervalo: **Média**

Retorna a média dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
occurrence.zonal.interval.mean("<dynamic_data_occurrence>", "<attribute>",
                               "<time_begin>",  "<time_end>", [<buffer>], ["<restriction>"])
```

Exemplo:
```
buf1 = Buffer()
x = occurrence.zonal.interval.mean("raios", "Intensidade", "4h", "1h", buf1, "UF = 'AM'")
```

Zonal por intervalo: **Mediana**

Retorna a mediana dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
occurrence.zonal.interval.median("<dynamic_data_occurrence>", "<attribute>",
                                 "<time_begin>",  "<time_end>", [<buffer>], ["<restriction>"])
```

Exemplo:
```
buf1 = Buffer()
x = occurrence.zonal.interval.median("raios", "Intensidade", "4h", "1h", buf1, "UF = 'AM'")
```

Zonal por intervalo: **Soma**

Retorna a soma dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.interval.sum(“<dynamic_data_occurrence>”,“<time_begin>”, “<time_end>”, “<attribute>”, [<buffer>], “<restriction>”)

Exemplo: buf1 = Buffer()
x = occurrence.zonal.interval.sum("raios", "Intensidade", "4h", "1h", buf1, "UF = 'AM'")

Zonal por intervalo: **Desvio Padrão**

Retorna o desvio padrão dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.interval.standard_deviation("<dynamic_data_occurrence>", "<attribute>", "<time_begin>", "<time_end>", [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
x = occurrence.zonal.interval.standard_deviation("raios", "Intensidade", "4h", "1h", buf1)

Zonal por intervalo: **Variância**

Retorna a variância dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

occurrence.zonal.interval.variance("<dynamic_data_occurrence>", "<attribute>", "<time_begin>", "<time_end>", [<buffer>], ["<restriction>"])

Exemplo: x = occurrence.zonal.interval.variance("raios", "Intensidade", "360s", "48s")

II.3- Zonal por agregação

Grupo de operadores que consideram as ocorrências agrupadas em torno do mesmo ponto e que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Para verificar as ocorrências agrupadas um outro “buffer” nas ocorrências é definido.

SINTAXE GERAL:

occurrence.zonal.aggregation.<operator>("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

onde:

- **operator**: count, min, max, mean, sum, median, standard_deviation, variance
- dynamic_data_occurrence : String com o nome da série de dados de ocorrências;

- **attribute**: String com o nome do atributo da ocorrência que deve ser utilizado para recuperar os valores, o atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador "count";
- **time**: String com o intervalo de tempo, a partir da hora atual, para filtrar as ocorrências. Ver utilitário Unidades de tempo;
- **aggregationStatistic**: Tipo de operador estatístico a ser utilizado para selecionar o valor do atributo para as ocorrências agregadas. Não usar para operador "count";
- **aggregationBuffer**: Buffer para agregação dos pontos de ocorrência. Ver utilitário Buffer;
- **buffer**: [Opcional] "Buffer" para ser aplicado ao objeto monitorado. Parâmetro obrigatório somente se o parâmetro seguinte (restriction) for utilizado. Se não declarado será considerada a própria geometria do objeto (no caso do um mapa de polígonos, a própria área do polígono é utilizada). Ver utilitário "buffer" acima;
- **restriction**: [Opcional] String com a restrição SQL a ser aplicada sobre atributos do dado dinâmico de ocorrência. Não utilizar se não houver restrição. Se for utilizada a restrição, obrigatoriamente o parâmetro de buffer acima deve ser declarado, mesmo que o buffer seja nulo.

Segue a descrição de cada operador.

Zonal por agregação: **Contagem**

Retorna a quantidade de ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto serão contadas uma única vez.

Sintaxe:

```
occurrence.zonal.aggregation.count("<dynamic_data_occurrence>", "<time>",
        aggregationBuffer, [<buffer>], ["<restriction>"])
```

Exemplo:

```
# exemplo com buffer na ocorrência e no objeto, e com restrição
buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
bufaggreg = Buffer(BufferType.Out,  200, "m")
x = occurrence.zonal.aggregation.count("foco", "1d", bufaggreg, buf1, "UF = 'AM'")

# exemplo sem buffer no objeto, mas com buffer na ocorrência, e sem restrição
bufaggreg = Buffer(BufferType.Out,  200, "m")
x = occurrence.zonal.aggregation.count("foco", "1d", bufaggreg)

# exemplo sem buffer e com restrição
buf1 = Buffer()
x = occurrence.zonal.aggregation.count("foco", "1d", bufaggreg, buf1, "UF = 'AM'")
# ou
x = occurrence.zonal.aggregation.count("foco", "1d", bufaggreg, Buffer(), "UF = 'AM'")
```

Zonal por agregação: **Mínimo**

Retorna o menor valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.min("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo: bufaggreg = Buffer(BufferType.Out, 200, "m")
x = occurrence.zonal.aggregation.min("raios", "Intensidade", "4h", Statistic.min, bufaggreg)

Zonal por agregação: **Máximo**

Retorna o maior valor do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.max("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo: bufaggreg = Buffer(BufferType.Out, 200, "m")
x = occurrence.zonal.aggregation.max("raios", "Intensidade", "4h", Statistic.max, bufaggreg)

Zonal por agregação: **Média**

Retorna a média dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.mean("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo: buf1 = Buffer()
bufaggreg = Buffer(BufferType.Out, 200, "m")
x = occurrence.zonal.aggregation.mean("raios", "Intensidade", "1h", Statistic.max, bufaggreg, buf1, "UF = 'AM'")

Zonal por agregação: **Mediana**

Retorna a mediana dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.median("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"]))

Exemplo: bufaggreg = Buffer(BufferType.Out, 200, "m")
x = occurrence.zonal.aggregation.median("raios", "Intensidade", "1h", Statistic.max, bufaggreg)

Zonal por agregação: **Soma**

Retorna a soma dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.sum("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo:
```
buf1 = Buffer()
bufaggreg = Buffer(BufferType.Out,  200, "m")
x = occurrence.zonal.aggregation.sum("raios", "Intensidade", "1h", Statistic.max,
bufaggreg, buf1, "UF = 'AM'")
```

Zonal por agregação: **Desvio Padrão**

Retorna o desvio padrão dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.standard_deviation("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo:
```
buf1 = Buffer(BufferType.Level,  400, "m", 200, "m")
bufaggreg = Buffer(BufferType.Out,  200, "m")
x = occurrence.zonal.aggregation.standard_deviation("raios", "Intensidade", "1h",
Statistic.max, bufaggreg, buf1)
```

Zonal por agregação: **Variância**

Retorna a variância dos valores do atributo das ocorrências que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Ocorrências agrupadas em torno do mesmo ponto terão um único valor obtido por análise estatística (utilitário estatístico).

Sintaxe:

occurrence.zonal.aggregation.variance("<dynamic_data_occurrence>", "<attribute>", "<time>",aggregationStatistic, aggregationBuffer, [<buffer>], ["<restriction>"])

Exemplo:
```
bufaggreg = Buffer(BufferType.Out,  200, "m")
x = occurrence.zonal.aggregation.variance("raios", "Intensidade", "360sec", Statistic.max,
bufaggreg)
```

3.3.3.3 - Operadores zonais de PCDs

Operadores zonais de PCD (Plataforma de Coleta de Dados) são utilizados para obter estatísticas sobre as fontes de dados do tipo de PCD. Consideram a localização fixa dos pontos e obedecem uma regra de influência ou uma lista informando quais serão os pontos de PCD que serão considerados para cada objeto a ser monitorado ou a área do "buffer" desses. Cada PCD pode ter seu próprio conjunto de atributos associados.

A Figura 3.20 mostra a área de influência de alguns pontos de PCD em relação a área de um objeto monitorado e em relação ao buffer externo (apenas anel externo sem considerar a área de objeto) desse mesmo objeto.

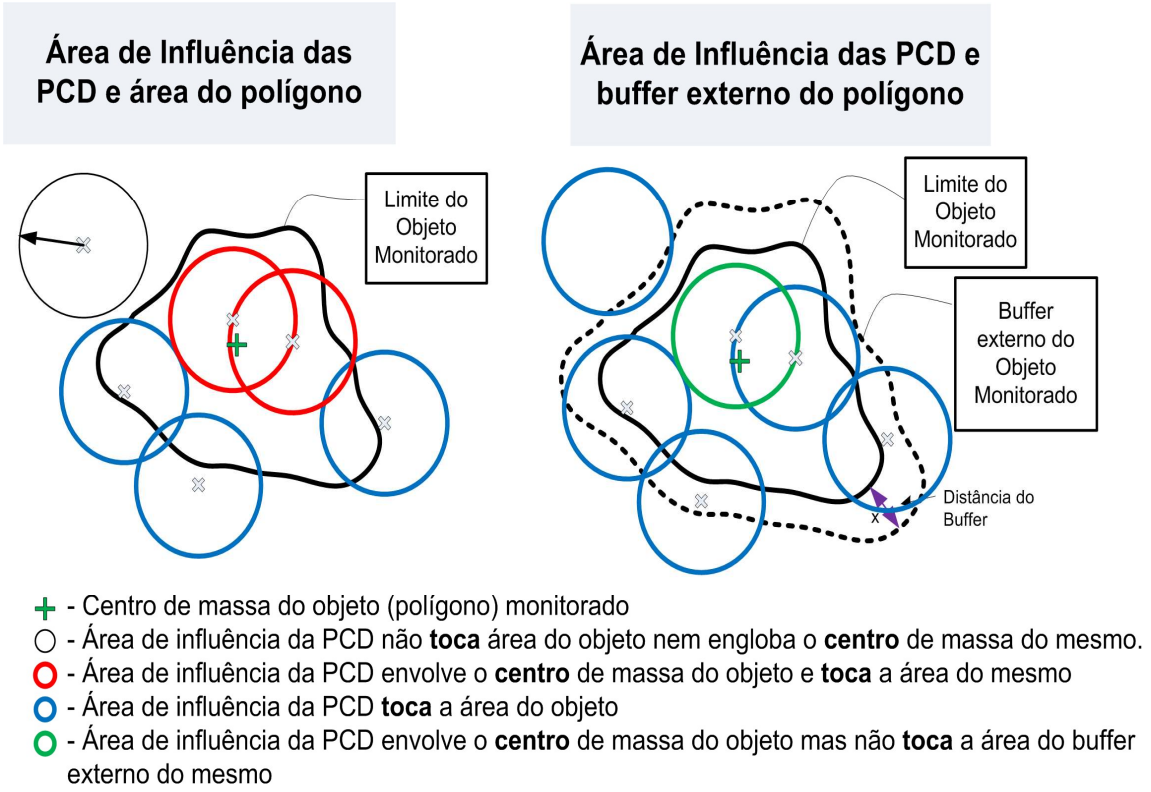


Figura 3.20 – Diferenças de abordagens da relação entre a área de influência de PCD´s e um objeto monitorado ou um "buffer" externo deste objeto.

NOTA: Os operadores e utilitários desse item utilizam dos dados das PCD´s de forma discreta, pois as medidas realizadas são reais nos pontos de localização de cada PCD. Neste caso, um conjunto de regras de influência podem ser aplicadas a cada localização. Outra maneira de utilizar os dados de PCD, de forma contínua, é a partir de um dado matricial criado pela interpolação de uma variável escolhida (ver item 2.4.2). O dado dinâmico matricial resultante da interpolação pode ser utilizado igualmente como qualquer outro dado matricial. Veja a lista de operadores zonais de grades no item 3.3.3.4.

Antes de usar os operadores devemos conhecer os utilitários de PCD que permitem definir as regras de influência, descrito a seguir.

III.1- Regra de Influência

Há dois utilitários para definir quais PCD´s serão consideradas para cada objeto monitorado, um baseado em regras e outro nos atributos do próprio objeto. O utilitário baseado em regras depende do tipo escolhido pelo usuário, isto é, se **toca** a área do objeto ou seu "buffer", se envolve o **centro** de massa da área do objeto ou seu "buffer" ou uma **região** específica para cada PCD. Para os tipos toca e centro, um valor de raio deverá ser informado na análise. A Figura 3.21 mostra os três tipos de regras de influência.

SINTAXE GERAL:

dcp.zonal.influence.by_attribute("<dynamic_data_dcp>", <list_attribute>)

dcp.zonal.influence.by_rule("<dynamic_data_dcp>", [<buffer>])

onde:

- **dynamic_data_dcp** : String com o nome da série de dados de PCD;
- **list_attribute**: Parâmetro contendo a lista de atributos do objeto monitorado contendo ID´s das PCD´s que o influenciam. Ex. [att1, att2, att3] ;
- **buffer** : [Opcional] "Buffer" para ser aplicado ao objeto monitorado. Ver utilitário "buffer" acima.

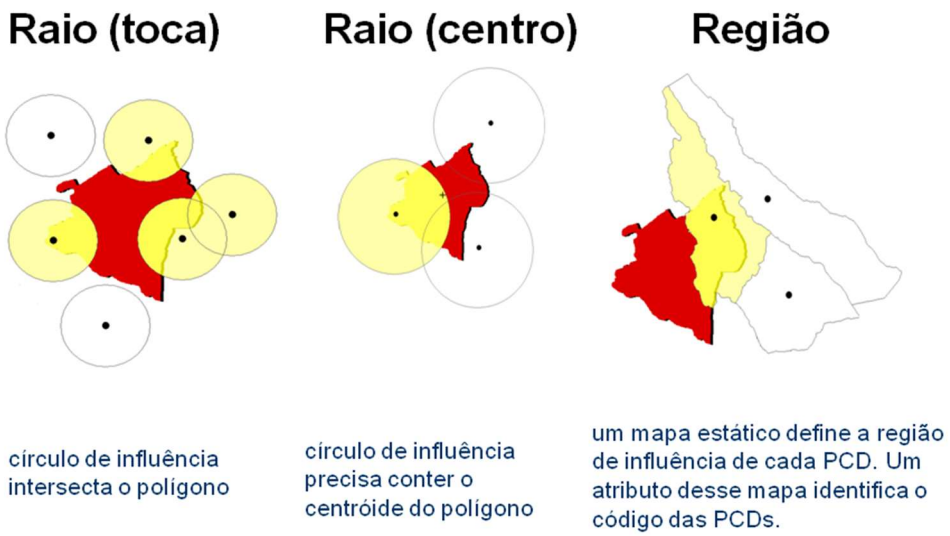
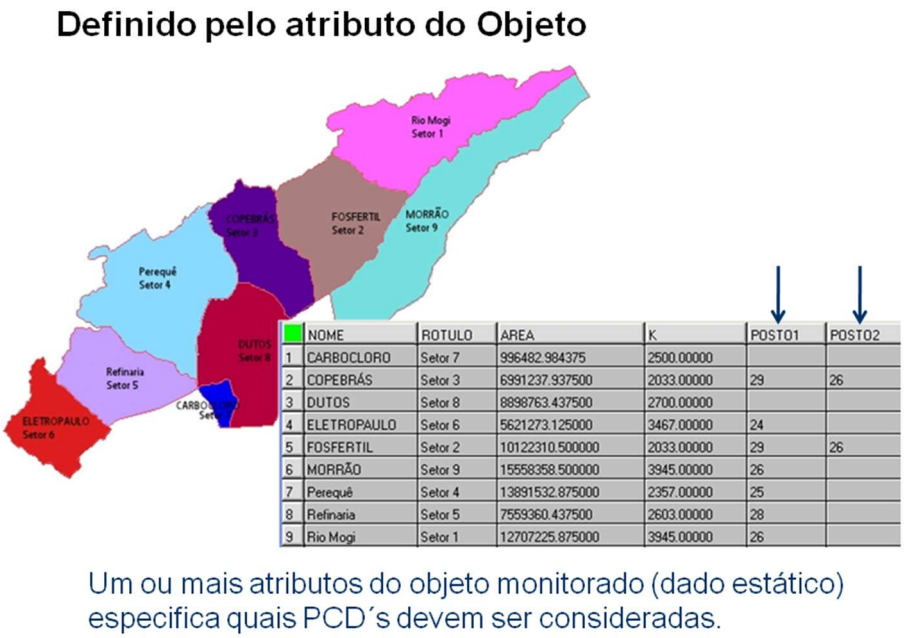


Figura 3.21 – Diferentes regras de influência das PCDs

A Figura 3.22 mostra um mapa contendo polígonos de algumas bacias hidrográficas e a tabela de atributos associada. Duas colunas dessa tabela especificam os códigos das PCDs que devem ser utilizados por cada bacia.



	NOME	ROTULO	AREA	K	POSTO1	POSTO2
1	CARBOCLORO	Setor 7	996482.984375	2500.00000		
2	COPEBRÁS	Setor 3	6991237.937500	2033.00000	29	26
3	DUTOS	Setor 8	8898763.437500	2700.00000		
4	ELETROPAULO	Setor 6	5621273.125000	3467.00000	24	
5	FOSFERTIL	Setor 2	10122310.500000	2033.00000	29	26
6	MORRÃO	Setor 9	15558358.500000	3945.00000	26	
7	Perequê	Setor 4	13891532.875000	2357.00000	25	
8	Refinaria	Setor 5	7559360.437500	2603.00000	28	
9	Rio Mogi	Setor 1	12707225.875000	3945.00000	26	

Figura 3.22 – Mapa e a tabela de atributos com código das PCDs a serem utilizados.

NOTA: Em ambos os utilitários o resultado será uma lista contendo as PCDs que serão utilizados por cada objeto monitorado.

A seguir são apresentados três tipos de operadores: **Zonal**, **Zonal histórico** e **Zonal histórico por intervalo**. A descrição de cada tipo a seguir.

III.2- Zonal

Grupo de operadores que consideram as PCD´s que influenciam o objeto monitorado e utilizam somente a última medida obtidas por cada PCD.

SINTAXE GERAL:

dcp.zonal.<operator>("<dynamic_data_dcp>", <buffer>, "<attribute>", <list_dcp>)

onde:

- **operator**: Count, Min, Max, Mean, Median, Sum, Standard_deviation, Variance;
- **dynamic_data_dcp**: String com o nome da série de dados de PCD;
- **buffer**: Buffer para ser aplicado ao objeto monitorado. Parâmetro obrigatório somente para operador “count”. Ver utilitário Buffer;
- **attribute**: String com o nome do atributo da PCD que deve ser utilizado para recuperar valores estatísticos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador “count”;
- l**ist_dcp**: Lista contendo a identificação das PCD´s que influenciam o objeto monitorado. Ver utilitário “Regra de Influência”. Não usar se operador zonal for “count”.

Segue a descrição de cada operador.

Zonal: **Contagem**

Retorna o número de PCD´s que influenciam o objeto monitorado ou sua área de influência. Depende do valor de raio informado para definir a área ao redor de cada PCD para os tipos Centro ou **Toca,** ou áreas se tipo for **Região.**

Sintaxe:

dcp.zonal.count("<dynamic_data_dcp>", <buffer>)

Exemplo:
```
buf1 = Buffer(BufferType.Out_union, 2, "km")
x = dcp.zonal.count("estacoes", buf1)
```

Zonal: **Mínimo**

Retorna o menor valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.min("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.min("Serra do Mar", "Pluvio", ids)
```

Zonal: **Máximo**

Retorna o maior valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.max("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.max("Serra do Mar", "Pluvio", ids)
```

Zonal: **Média**

Retorna a média dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.mean("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.mean("Serra do Mar", "Chuva", ids)
```

Zonal: **Mediana**

Retorna a mediana dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.median("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.median("Serra do Mar", "temperatura", ids)
```

Zonal: **Soma**

Retorna a soma dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.sum("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.zonal.sum("Serra do Mar", "temperatura", ids)
```

Zonal: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.standard_deviation("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.standard_deviation("Serra do Mar", "temperatura", ids)
```

Zonal: **Variância**

Retorna a variância dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado.

Sintaxe:

dcp.zonal.variance("<dynamic_data_dcp>", "<attribute>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.variance("Serra do Mar", "temperatura", ids)
```

III.3- Zonal histórico

Grupo de operadores que consideram as PCD´s que influenciam o objeto monitorado e utilizam as últimas medidas obtidas por cada PCD, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

dcp.zonal.history.<operator>("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

onde:

- **operator** : Min, Max, Mean, Sum, Median, Standard_deviation, Variance;
- **dynamic_data_dcp** : String com o nome da série de dados de PCD;
- **attribute** : String com o nome do atributo da PCD que deve ser utilizado para recuperar valores estatísticos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador “count”;
- **time** : String com o intervalo de tempo, a partir da hora atual, para filtrar os valores das PCD´s. Este intervalo será aberto (< x) no valor informado e fechado (=) na hora atual. Ver utilitário unidades de tempo;

- **list_dcp** : lista contendo a identificação das PCD´s que influenciam o objeto monitorado. Ver utilitário "Regra de Influência". Não usar se operador zonal for "count".

Segue a descrição de cada operador.

Zonal histórico: **Mínimo**

Retorna o menor valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.min("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.min("Serra do Mar", "Pluvio", "1d", ids)
```

Zonal histórico: **Máximo**

Retorna o maior valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.max("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.max("Serra do Mar", "Pluvio", "30h", ids)
```

Zonal histórico: **Média**

Retorna a média dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.mean("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.mean("Serra do Mar", "Chuva", "3d", ids)
```

Zonal histórico: **Mediana**

Retorna a mediana dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.median("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.median("Serra do Mar", "temperatura", "360min", ids)
```

Zonal histórico: **Soma**

Retorna a soma dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.sum("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.zonal.history.sum("Serra do Mar", "temperatura", "5h", ids)
```

Zonal histórico: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.standard_deviation("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.standard_deviation("Serra do Mar", "temperatura", "1w", ids)
```

Zonal histórico: **Variância**

Retorna a variância dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.zonal.history.variance("<dynamic_data_dcp>", "<attribute>", "<time>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.variance("Serra do Mar", "temperatura", "24h", ids)
```

III.4- Zonal histórico por intervalo

Grupo de operadores que consideram as PCD´s que influenciam o objeto monitorado e utilizam as últimas medidas obtidas por cada PCD, no intervalo de tempo definido entre dois valores de tempo informado no passado.

SINTAXE GERAL:

dcp.zonal.history.interval.<operator>("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

onde:

- **operator** : min, max, mean, sum, median, standard_deviation, variance;
- **dynamic_data_dcp**: String com o nome da série de dados de PCD;

- **attribute**: String com o nome do atributo da PCD que deve ser utilizado para recuperar valores estatísticos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador “count”;
- **time_begin**: String inicial (mais antigo) do intervalo de tempo para filtrar os dados. Este valor será aberto (< tempo mais antigo) no tempo informado;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar os dados. Este valor será fechado (<= tempo mais recente) no tempo informado;
- **list_dcp**: lista contendo a identificação das PCD´s que influenciam o objeto monitorado. Ver utilitário “Regra de Influência”. Não usar se operador zonal for “count”.

Segue a descrição de cada operador.

Zonal histórico por intervalo: **Mínimo**

Retorna o menor valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.min(“<dynamic_data_dcp>”, “<attribute>”, “<time_begin>”, “<time_end>”, <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.min("Serra do Mar", "Pluvio", "2d", '1d', ids)
```

Zonal histórico por intervalo: **Máximo**

Retorna o maior valor de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.max(“<dynamic_data_dcp>”, “<attribute>”, “<time_begin>”, “<time_end>”, <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.max("Serra do Mar", "Pluvio", "24h", "12h", ids)
```

Zonal histórico por intervalo: **Média**

Retorna a média dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.mean("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.mean("Serra do Mar", "Chuva", "24d", "12d", ids)
```

Zonal histórico por intervalo: **Mediana**

Retorna a mediana dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.median("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.median("Serra do Mar", "temperatura", "2d", "1d", ids)

Zonal histórico por intervalo: **Soma**

Retorna a soma dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.sum("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

Exemplo: ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.zonal.history.interval.sum("Serra do Mar", "temperatura", "2d", "1d", ids)

Zonal histórico por intervalo: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.standard_deviation("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.standard_deviation("Serra do Mar", "temperatura", "2d", "1d", ids)

Zonal histórico por intervalo: **Variância**

Retorna a variância dos valores de um atributo comum das PCD´s que influenciam cada geometria de um objeto monitorado, no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.zonal.history.interval.variance("<dynamic_data_dcp>", "<attribute>", "<time_begin>", "<time_end>", <list_dcp>)

Exemplo: b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.zonal.history.interval.variance("Serra do Mar", "temperatura", "2d", "1d", ids)

3.3.3.4 - Operadores zonais de Grades

Operadores zonais de grades são operadores utilizados para obter estatísticas sobre as fontes de dados matriciais. Consideram os valores dos pontos da grade que interceptam todas as geometrias (pontos, linhas ou polígonos) de um mapa (objetos monitorados) ou a área de influência (buffer) dessas geometrias no tempo atual, passado ou futuro.

Tais operadores utilizam dados dinâmicos matriciais que foram coletados de fontes locais ou remotas, seja dados de observação ou previsão, ou ainda de que foram produzidos por interpolação de PCDs ou uma análise baseada em grades.

A Figura 3.23a ilustra um polígono sobre um dado matricial e quais pontos (ou "*pixels*") da grade serão considerados pelo operador zonal. Note que uma pequena interseção entre a área do "pixel" e a área do objeto fará com que o valor naquele ponto da grade seja considerado, não sendo necessário incluir o centro do mesmo. Para o mesmo polígono, considerando a área do buffer externo (Figura 3.23b), os pontos da grade que serão considerados são outros.

Os dados matriciais podem representar **medidas** do meio ambiente de um parâmetro, precipitação (chuva) por exemplo, que está sendo observado e neste caso são chamados de dados observacionais. Os operadores que trabalham sobre estes dados lidam com os valores atuais (ou mais recente) e seu **histórico** imediato. A Figura 3.24 mostra um conjunto de dados obtidos em diferentes horários até o mais recente, que são sobrepostos ao limite de um objeto que está sendo monitorado. Estes operadores estão identificados em grupos como "**Zonal Histórico** <sub_tipo>". Normalmente cada dado (grade) está armazenado em um arquivo com data/hora em que foram coletadas as medidas.

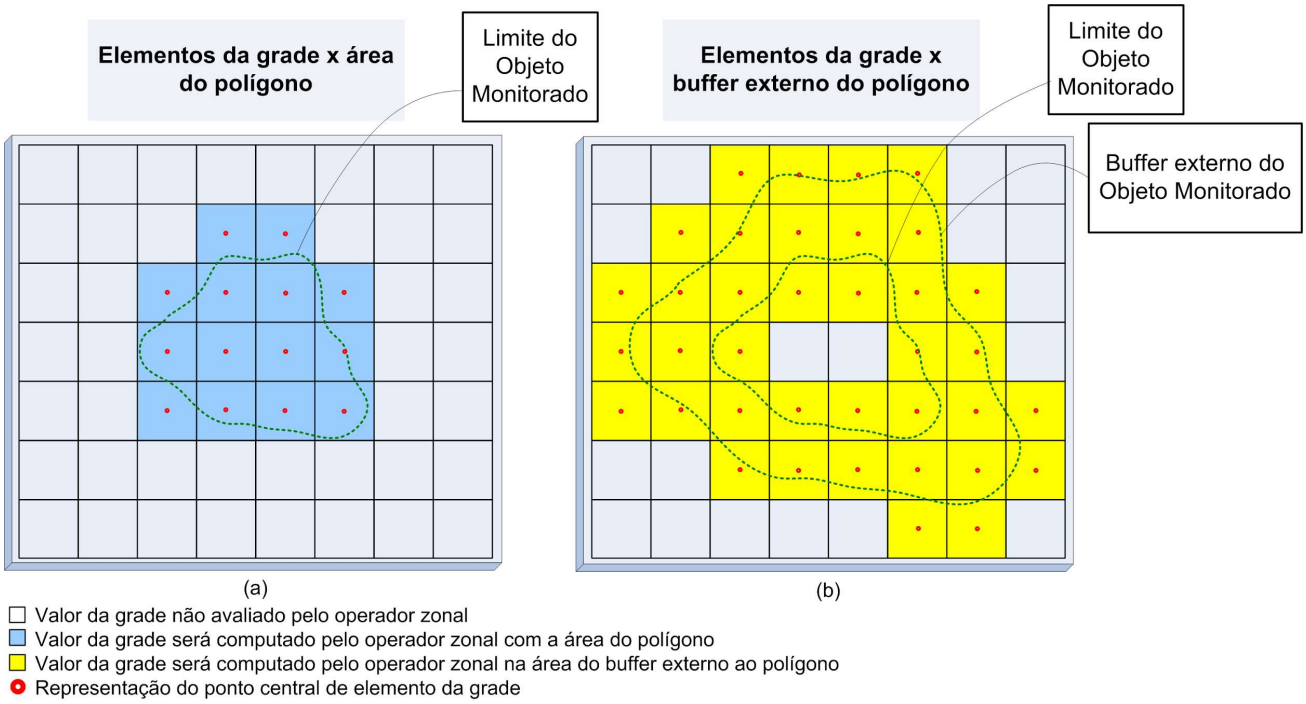


Figura 3.23 – (a) Exemplo dos pontos de uma grade recuperados por um operador zonal que utiliza um polígono (b) Pontos de uma grade recuperados utilizando o buffer externo desse polígono.

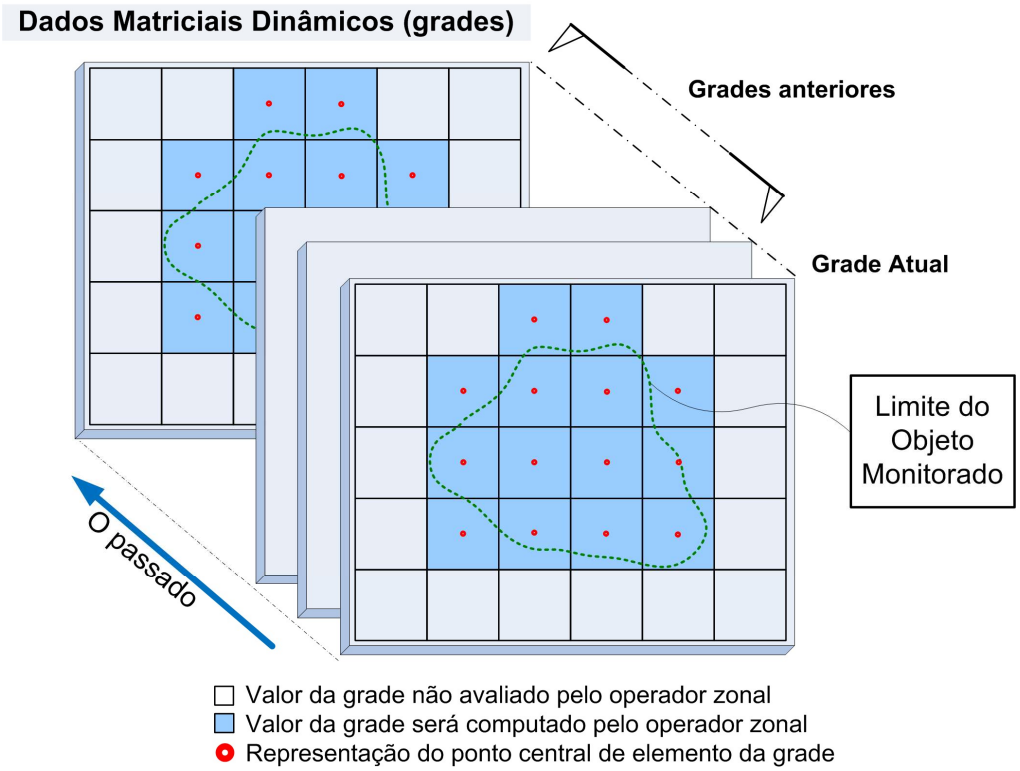


Figura 3.24 – Dados históricos de observação sobrepostos ao objeto sob monitoramento.

De outra maneira, dados matriciais podem representar **estimativas** do meio ambiente de um parâmetro, precipitação (chuva) por exemplo, que está sendo estimada por meio de modelos matemáticos e neste caso são chamados de dados de **previsão**. Os operadores que trabalham sobre estes dados lidam com os valores atuais (ou mais recente) e seu **futuro** imediato. A Figura 3.25 mostra um conjunto de dados estimados para diferentes horários, da camada mais atual até um futuro mais recente, que são sobrepostos ao limite de um objeto que está sendo monitorado. Estes operadores estão identificados em grupos como "**Zonal de Previsão** <sub_tipo>". Normalmente os diferentes horários de previsão (camadas) estão armazenados em um único arquivo com data/hora de referência de quando foi executado o modelo e qual é o intervalo de tempo entre cada camada.

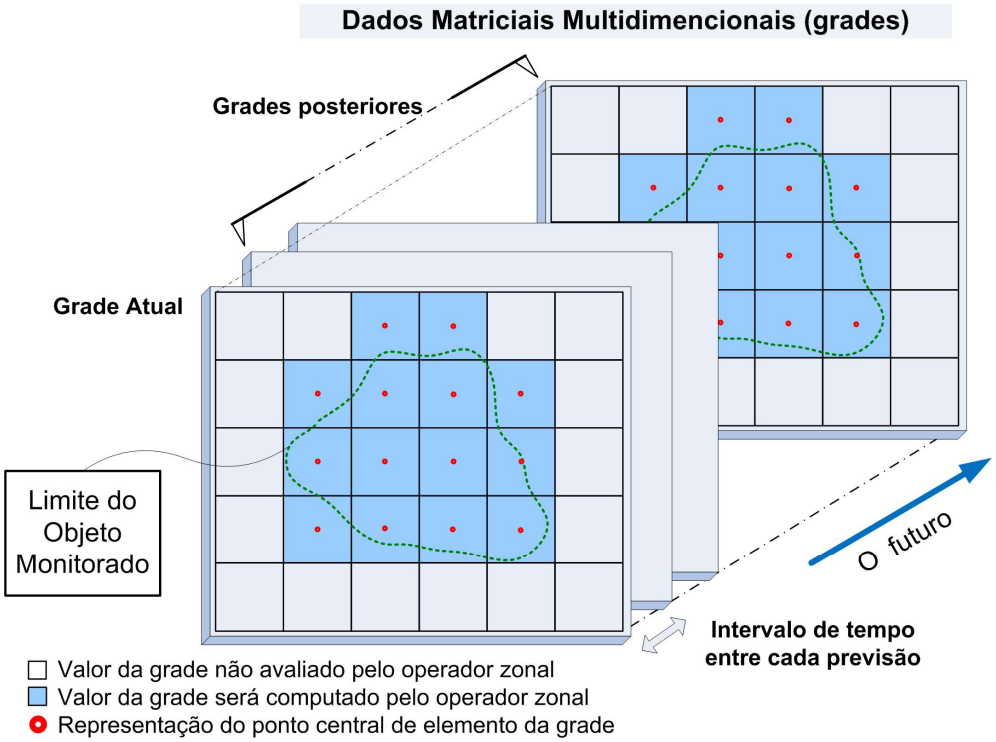


Figura 3.25 – Dados históricos de observação sobrepostos ao objeto sob monitoramento.

Estes operadores zonais de grade são divididos em oito grupos: **Zonal**, **Zonal Histórico, Zonal Histórico de Precipitação, Zonal Histórico Acumulado, Zonal Histórico por Intervalo, Zonal de Previsão, Zonal de Previsão Acumulado** e **Zonal de Previsão por Intervalo**. A descrição de cada tipo a seguir.

IV.1- Zonal

Grupo de operadores que consideram somente o último dado dinâmico matricial (ou mais atual) que intercepta o objeto monitorado ou sua área de influência (buffer).

SINTAXE GERAL:

grid.zonal.<operator>(“<dynamic_data_grid>”, <value>, <begin>, <end>, [<band>], [<buffer>])

onde:

- **operator**: count, count_by_value, count_by_range, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de observação.
- **value**: Valor a ser contado. Utilizado por exemplo para contar classes de um mapa temático matricial (matriz de números inteiros). Utilizado somente para operador “count_by_value”.
- **begin, end**: Valor inicial e final do intervalo utilizado para contar “pixels”. Utilizado por exemplo para contar pixels dentro de um intervalo de um dado matricial (matriz de reais). Utilizado somente para operador “count_by_range”.

- **band**: [Opcional] Banda da grade ser utilizada. Parâmetro obrigatório somente se o parâmetro seguinte (buffer) for utilizado. Se não informado será considerado a primeira banda (0).
- **buffer**: [Opcional] Buffer para ser aplicado ao objeto monitorado. Parâmetro não obrigatório. Ver utilitário Buffer.

Segue a descrição de cada operador.

Zonal: **Contagem**

Retorna a quantidade de "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.count("<dynamic_data_grid>", [<band>], [<buffer>])
```

Exemplo:
```
buf1 = Buffer()
x = grid.zonal.count("hidro", 0, buf1)
```

Zonal: **Contagem por Valor**

Retorna a quantidade de "pixels" de um determinado valor da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.count_by_value("<dynamic_data_grid>", <value>, [<band>], [<buffer>])
```

Exemplo:
```
# conta valores 5 da primeira camada do dado matricial sobre o buffer do objeto
buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
x = grid.zonal.count_by_value("uso_solo", 5, 0, buf1)

# conta valores 3 da segunda camada do dado matricial sem buffer do objeto
x = grid.zonal.count_by_value("uso_solo", 3, 1)

# conta valores 12 da primeira camada do dado matricial sem buffer do objeto
buf1 = Buffer()
x = grid.zonal.count_by_value("uso_solo", 12, 0, buf1)
# ou
x = grid.zonal.count_by_value("uso_solo", 12)
```

Zonal: **Contagem por intervalo**

Retorna a quantidade de "pixels" dentro de um determinado intervalo de valores da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.count_by_range("<dynamic_data_grid>", <begin>, <end>, [<band>], [<buffer>])
```

Exemplo:
```
# conta valores entre 10.5  e 35 da primeira camada do dado matricial sobre o buffer do objeto
buf1 = Buffer(BufferType.Level, 400, "m", 200, "m")
x = grid.zonal.count_by_range("prec", 10.5, 35, 0, buf1)

# conta valores entre 200 e 1200 da primeira camada do dado matricial sem buffer do objeto
buf1 = Buffer()
x = grid.zonal.count_by_value("prec", 200, 1200, 0, buf1)
# ou
x = grid.zonal.count_by_value("prec", 200, 1200)
```

Zonal: **Soma**

Retorna a soma dos valores dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.sum("<dynamic_data_grid>", [<band>], [<buffer>])
```

Exemplo:
```
x = grid.zonal.sum("hidro", 0)
#  ou
x = grid.zonal.sum("hidro")
```

Zonal: **Média**

Retorna a média dos valores dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.mean("<dynamic_data_grid>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
x = grid.zonal.mean("hidro", 0, b1)
```

Zonal: **Mínimo**

Retorna o menor valor dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.min("<dynamic_data_grid>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.min("hidro", 0, b1)
```

Zonal: **Máximo**

Retorna o maior valor dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

```
grid.zonal.max("<dynamic_data_grid>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.max("hidro", 0, b1)
```

Zonal: **Mediana**

Retorna a mediana dos valores dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

grid.zonal.median("<dynamic_data_grid>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.median("radar", 0, b1)
```

Zonal: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

grid.zonal.standard_deviation("<dynamic_data_grid>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.standard_deviation("radar", 0, b1)
```

Zonal: **Variância**

Retorna a variância dos valores dentre os "pixels" da última matriz (ou mais atual) que interceptam o objeto monitorado ou sua área de influência (buffer).

Sintaxe:

grid.zonal.variance("<dynamic_data_grid>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.variance("grade_chuva", 0, b1)
```

IV.2- Zonal Histórico

Grupo de operadores que consideram os últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

grid.zonal.history.<operator>("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

onde:

- **operator**: num, list, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de observação;
- **time**: String com o intervalo de tempo, a partir da hora atual, para filtrar as grades. Este intervalo será aberto (< x) no valor informado e fechado (=) na hora atual. Ver utilitário unidades de tempo;
- **band**: [Opcional] Banda da grade ser utilizada. Parâmetro obrigatório somente se o parâmetro seguinte (buffer) for utilizado. Se não informado será considerado a primeira banda (0). Não utilizar com operador "num e list";

- **buffer**: [Opcional] Buffer para ser aplicado ao objeto monitorado. Parâmetro não obrigatório. Ver utilitário Buffer.

Segue a descrição de cada operador.

Zonal Histórico: **Número**

Retorna o número de matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado**.**

Sintaxe:

```
grid.zonal.history.num("<dynamic_data_grid>", "<time>", [<buffer>])
```

Exemplo:
```
buf1 = Buffer()
x = grid.zonal.history.num("hidro", "12h", buf1)
```

Zonal Histórico: **Lista**

Retorna a lista com data/hora das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado**.**

Sintaxe:

```
grid.zonal.history.list("<dynamic_data_grid>", "<time>", [<buffer>])
```

Exemplo:
```
buf1 = Buffer()
x = grid.zonal.history.list("hidro", "12h", buf1)
```

Zonal Histórico: **Soma**

Retorna a soma dos valores dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.sum("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
x = grid.zonal.history.sum("hidro", "12h", 0)
```

Zonal Histórico: **Média**

Retorna a média dos valores dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.mean("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
x = grid.zonal.history.mean("hidro", "12h", 0, b1)
```

Zonal Histórico: **Mínimo**

Retorna o menor valor dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.min("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.history.min("hidro", "12h", 0, b1)
```

Zonal Histórico: **Máximo**

Retorna o maior valor dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.max("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.history.max("hidro", "12h", 0, b1)
```

Zonal Histórico: **Mediana**

Retorna a mediana dos valores dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.median("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.median("radar", "12h", 0, b1)
```

Zonal Histórico: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.standard_deviation("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.standard_deviation("radar", "12h", 0, b1)
```

Zonal Histórico: **Variância**

Retorna a variância dos valores dentre os pixels de todas as matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.variance("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.history.variance("grade_chuva", "12h", 0, b1)
```

IV.3- Zonal Histórico de Precipitação

Grupo de operadores que consideram os últimos dados dinâmicos matriciais (em mm/h de precipitação) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

grid.zonal.history.prec.<operator>("<dynamic_data_grid>", "<time>", <band>, <buffer>)

onde:

- **operator**: sum, mean, median, min, max, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de observação.
- **time**: String com o intervalo de tempo, a partir da hora atual, para filtrar as grades. Este intervalo será aberto (< x) no valor informado e fechado (=) na hora atual. Ver utilitário unidades de tempo;
- **band**: [Opcional] Banda da grade ser utilizada. Parâmetro obrigatório somente se o parâmetro seguinte (buffer) for utilizado. Se não informado será considerado a primeira banda (0).
- **buffer**: [Opcional] Buffer para ser aplicado ao objeto monitorado. Parâmetro não obrigatório. Ver utilitário Buffer.

Segue a descrição de cada operador.

Zonal Histórico de Precipitação: **Soma**

Retorna a soma dos valores dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.sum("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo: x = grid.zonal.history.prec.sum("hidro", "12h")

Zonal Histórico de Precipitação: **Média**

Retorna a média dos valores dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.mean("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
x = grid.zonal.history.prec.mean("hidro", "12h", 0, b1)
```

Zonal Histórico de Precipitação: **Mínimo**

Retorna o menor valor dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.min("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.history.prec.min("hidro", "12h", 0, b1)
```

Zonal Histórico de Precipitação: **Máximo**

Retorna o maior valor dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.max("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.history.prec.max("hidro", "12h", 0, b1)
```

Zonal Histórico de Precipitação: **Mediana**

Retorna a mediana dos valores dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.median("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.prec.median("radar", "12h", 0, b1)
```

Zonal Histórico de Precipitação: **Desvio Padrão**

Retorna o desvio padrão dos valores dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.prec.standard_deviation("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.prec.standard_deviation("radar", "12h", 0, b1)
```

Zonal Histórico de Precipitação: **Variância**

Retorna a variância dos valores dos últimos dados dinâmicos matriciais (em **mm/h de precipitação**) que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.prec.variance("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.history.prec.variance("grade_chuva", "12h", 0, b1)
```

IV.4- Zonal Histórico Acumulado

Grupo de operadores que consideram valores acumulados dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado. Primeiro é realizada a operação de soma dos pontos da matriz no intervalo de tempo para depois realizar a operação zonal.

SINTAXE GERAL:

```
grid.zonal.history.accum.<operator>("<dynamic_data_grid>", "<time>", <band>, <buffer>)
```

onde:

- **operator**: sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de observação;
- **time**: String com o intervalo de tempo, a partir da hora atual, para filtrar as grades. Este intervalo será aberto (< x) no valor informado e fechado (=) na hora atual. Ver utilitário unidades de tempo;
- **band**: [Opcional] Banda da grade ser utilizada. Parâmetro obrigatório somente se o parâmetro seguinte (buffer) for utilizado. Se não informado será considerado a primeira banda (0).
- **buffer**: [Opcional] Buffer para ser aplicado ao objeto monitorado. Parâmetro não obrigatório. Ver utilitário Buffer.

Segue a descrição de cada operador.

Zonal Histórico Acumulado: **Soma**

Retorna a **soma da soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.accum.sum("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
x = grid.zonal.history.accum.sum("hidro", "12h")
```

Zonal Histórico Acumulado: **Média**

Retorna a **média da soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.accum.mean("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
x = grid.zonal.history.accum.mean("hidro", "12h", 0, b1)
```

Zonal Histórico Acumulado: **Mínimo**

Retorna a **menor soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.accum.min("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.history.accum.min("hidro", "12h", 0, b1)
```

Zonal Histórico Acumulado: **Máximo**

Retorna a **maior soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.accum.max("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.history.accum.max("hidro", "12h", 0, b1)
```

Zonal Histórico Acumulado: **Mediana**

Retorna a **mediana da soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

```
grid.zonal.history.accum.median("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.accum.median("radar", "12h", 0, b1)
```

Zonal Histórico Acumulado: **Desvio Padrão**

Retorna o **desvio padrão da soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.accum.standard_deviation("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.accum.standard_deviation("radar", "12h", 0, b1)

Zonal Histórico Acumulado: **Variância**

Retorna a **variância da soma acumulada** dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.zonal.history.accum.variance("<dynamic_data_grid>", "<time>", [<band>], [<buffer>])

Exemplo: b1 = Buffer(BufferType.ln_diff, 800, "m")
x = grid.zonal.history.accum.variance("grade_chuva", "12h", 0, b1)

IV.5- Zonal Histórico por Intervalo

Grupo de operadores que consideram intervalo de tempo de valores dos últimos dados dinâmicos matriciais que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual**.**

SINTAXE GERAL:

grid.zonal.history.interval.<operator>("<dynamic_data_grid>", "<time_begin>", "<time_end>", <band>, <buffer>)

onde:

- **operator**: num, list, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais;
- **time_begin** : String inicial (mais antigo) do intervalo de tempo para filtrar as grades. Este valor será aberto (< tempo mais antigo) no tempo informado;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar as grades. Este valor será fechado (<= tempo mais recente) no tempo informado;
- **band**: [Opcional] Banda da grade ser utilizada. Parâmetro obrigatório somente se o parâmetro seguinte (buffer) for utilizado. Se não informado será considerado a primeira banda (0);
- **buffer**: [Opcional] Buffer para ser aplicado ao objeto monitorado. Parâmetro não obrigatório. Ver utilitário Buffer.

Segue a descrição de cada operador.

Zonal Histórico por Intervalo: **Número**

Retorna o número de matrizes no intervalo de tempo que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual**.**

Sintaxe:

grid.zonal.history.interval.num("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo: buf1 = Buffer()
x = grid.zonal.history.interval.num("hidro", "12h", "2h", buf1)

Zonal Histórico por Intervalo: **Lista**

Retorna a lista com data/hora das matrizes no intervalo de tempo que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual**.**

Sintaxe:

grid.zonal.history.interval.list("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo: buf1 = Buffer()
x = grid.zonal.history.interval.list("hidro", "12h", "2h", buf1)

Zonal Histórico por Intervalo: **Soma**

Retorna a soma dos valores dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

grid.zonal.history.interval.sum("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>], [<buffer>])

Exemplo: x = grid.zonal.history.interval.sum("hidro", "12h", "2h", 0)

Zonal Histórico por Intervalo: **Média**

Retorna a média dos valores dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

grid.zonal.history.interval.mean("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>], [<buffer>])

Exemplo: b1 = Buffer(BufferType.Level, 10, "km", 5, "km")
x = grid.zonal.history.interval.mean("hidro", "12h", "2h", 0, b1)

Zonal Histórico por Intervalo: **Mínimo**

Retorna o menor valor dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

grid.zonal.history.interval.min("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>], [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.history.interval.min("hidro", "12h", "2h", 0, b1)

Zonal Histórico por Intervalo: **Máximo**

Retorna o maior valor dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

```
grid.zonal.history.interval.max("<dynamic_data_grid>", "<time_begin>", "<time_end>",
[<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.history.interval.min("hidro", "12h", "2h", 0, b1)
```

Zonal Histórico por Intervalo: **Mediana**

Retorna a mediana dos valores dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

```
grid.zonal.history.interval.median("<dynamic_data_grid>", "<time_begin>", "<time_end>",
[<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.interval.median("radar", "12h", "2h", 0, b1)
```

Zonal Histórico por Intervalo: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

```
grid.zonal.history.interval.standard_deviation("<dynamic_data_grid>", "<time_begin>",
"<time_end>", [<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.history.interval.standard_deviation("radar", "12h", "2h", 0, b1)
```

Zonal Histórico por Intervalo: **Variância**

Retorna a variância dos valores dentre os pixels, no intervalo de tempo, das matrizes que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

```
grid.zonal.history.interval.variance("<dynamic_data_grid>", "<time_begin>", "<time_end>",
[<band>], [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.ln_diff, 800, "m")
x = grid.zonal.history.interval.variance("grade_chuva", "12h", "2h", 0, b1)
```

IV.6- Zonal de Previsão

Grupo de operadores que consideram as próximas camadas de dados dinâmicos matriciais de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

SINTAXE GERAL:

grid.zonal.forecast.<operator>("<dynamic_data_grid>", "<time>", [<buffer>])

onde:

- **operator**: num, list, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de previsão;
- **time**: String com o intervalo de tempo, a partir da hora atual para o futuro, para filtrar as camadas de previsão. Ver utilitário Unidades de tempo;
- **buffer**: [Opcional] Objeto Buffer para ser aplicado ao objeto monitorado. Ver utilitário Buffer. Não obrigatório. Não utilizar com operador "num e list".

Segue a descrição de cada operador.

Zonal de Previsão: **Número**

Retorna o número de camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro**.**

Sintaxe:

grid.zonal.forecast.num("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo:
```
b1 = Buffer()
x = grid.zonal.forecast.num("Eta15km", "12h", b1)
```

Zonal de Previsão: **Lista**

Retorna a lista com data/hora das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro**.**

Sintaxe:

grid.zonal.forecast.list("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo:
```
b1 = Buffer()
x = grid.zonal.forecast.list("Eta15km", "12h", b1)
```

Zonal de Previsão: **Soma**

Retorna a soma dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.sum("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo:
```
x = grid.zonal.forecast.sum("Eta15km", "12h")
```

Zonal de Previsão: **Média**

Retorna a média dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.median("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Level, 10, "km", 5, "km")
x = grid.zonal.forecast.mean("Eta15km", "12h", b1)

Zonal de Previsão: **Mínimo**

Retorna o menor valor dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.min("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.forecast.min("Eta15km", "12h", b1)

Zonal de Previsão: **Máximo**

Retorna o maior valor dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.max("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.forecast.max("Eta15km", "12h", b1)

Zonal de Previsão: **Mediana**

Retorna a mediana dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.median("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.median("Eta15km", "12h", b1)

Zonal de Previsão: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.standard_deviation("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.standard_deviation("Eta15km", "12h", b1)

Zonal de Previsão: **Variância**

Retorna a variância dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.variance("<dynamic_data_grid>", "<time>", [<buffer>]

Exemplo:
```
b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.forecast.variance("Eta15km", "12h", b1)
```

IV.7- Zonal de Previsão Acumulado

Grupo de operadores que consideram as próximas camadas de dados dinâmicos matriciais de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro. Primeiro é realizada a operação de soma dos pontos da matriz no intervalo de tempo para depois realizar a operação zonal.

SINTAXE GERAL:

grid.zonal.forecast.accum.<operator>("<dynamic_data_grid>", "<time>", [<buffer>])

onde:

- **operator** : count, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid** : String com o nome da série de dados matriciais de previsão;
- **time** : String com o intervalo de tempo, a partir da hora atual para o futuro, para filtrar as camadas de previsão. Ver utilitário Unidades de tempo;
- **buffer** : [Opcional] Objeto Buffer para ser aplicado ao objeto monitorado. Ver utilitário Buffer. Não obrigatório.

Segue a descrição de cada operador.

Zonal de Previsão Acumulado: **Contagem**

Retorna a quantidade de "pixels" das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.count("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.forecast.accum.count("Eta20km", "12h", b1)
```

Zonal de Previsão Acumulado: **Soma**

Retorna a soma dos valores dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.sum("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: x = grid.zonal.forecast.accum.sum("Eta20km", "12h")

Zonal de Previsão Acumulado: **Média**

Retorna a média dos valores dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.mean("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Level, 10, "km", 5, "km")
x = grid.zonal.forecast.accum.mean("Eta20km", "12h", b1)

Zonal de Previsão Acumulado: **Mínimo**

Retorna o menor valor dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.min("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.forecast.accum.min("Eta20km", "12h", b1)

Zonal de Previsão Acumulado: **Máximo**

Retorna o maior valor dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.max("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.forecast.accum.max("Eta20km", "12h", b1)

Zonal de Previsão Acumulado: **Mediana**

Retorna a mediana dos valores dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.zonal.forecast.accum.median("<dynamic_data_grid>", "<time>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.accum.median("Eta20km", "12h", b1)

Zonal de Previsão Acumulado: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

```
grid.zonal.forecast.accum.standard_deviation("<dynamic_data_grid>", "<time>", [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.accum.standard_deviation("Eta20km", "12h", b1)
```

Zonal de Previsão Acumulado: **Variância**

Retorna a variância dos valores dentre os pixels das camadas de previsão acumulado que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

```
grid.zonal.forecast.accum.variance("<dynamic_data_grid>", "<time>", [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.forecast.accum.variance("Eta20km", "12h", b1)
```

IV.8- Zonal de Previsão por Intervalo

Grupo de operadores que consideram intervalo de tempo das próximas camadas de dados dinâmicos matriciais de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

SINTAXE GERAL:

```
grid.zonal.forecast.interval.<operator>("<dynamic_data_grid>", "<time_begin>",
                                        "<time_end>", [<buffer>])
```

onde:

- **operator**: num, list, sum, mean, min, max, median, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de previsão;
- **time_begin**: String inicial (mais próximo da hora atual) do intervalo de tempo para filtrar as camadas de previsão. Este valor será fechado (<= tempo mais próximo) no tempo informado;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar as camadas de previsão. Este valor será aberto (< tempo mais distante) no tempo informado;
- **buffer**: [Opcional] Objeto Buffer para ser aplicado ao objeto monitorado. Ver utilitário Buffer. Não obrigatório.

Segue a descrição de cada operador.

Zonal de Previsão por Intervalo: **Número**

Retorna o número de camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual**.**

Sintaxe:

```
grid.zonal.forecast.interval.num("<dynamic_data_grid>", "<time_begin>", "<time_end>",
                                 [<buffer>])
```

Exemplo:
```
buf1 = Buffer()
x = grid.zonal.forecast.interval.num("Brams15km", "12h", "2h")
```

Zonal de Previsão por Intervalo: **Lista**

Retorna a lista com data/hora das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual**.**

Sintaxe:

```
grid.zonal.forecast.interval.list("<dynamic_data_grid>", "<time_begin>", "<time_end>",
                                  [<buffer>])
```

Exemplo:
```
buf1 = Buffer()
x = grid.zonal.forecast.interval.list("Brams15km", "12h", "2h")
```

Zonal de Previsão por Intervalo: **Soma**

Retorna a soma dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

```
grid.zonal.forecast.interval.sum("<dynamic_data_grid>", "<time_begin>", "<time_end>",
                                 [<buffer>])
```

Exemplo:
```
x = grid.zonal.forecast.interval.sum("Brams15km", "12h", "2h")
```

Zonal de Previsão por Intervalo: **Média**

Retorna a média dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

```
grid.zonal.forecast.interval.mean("<dynamic_data_grid>", "<time_begin>", "<time_end>",
                                  [<buffer>])
```

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
x = grid.zonal.forecast.interval.mean("Brams15km", "12h", "2h", b1)
```

Zonal de Previsão por Intervalo: **Mínimo**

Retorna o menor valor dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

grid.zonal.forecast.interval.min("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
x = grid.zonal.forecast.interval.min("Brams15km", "12h", "2h", b1)
```

Zonal de Previsão por Intervalo: **Máximo**

Retorna o maior valor dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

grid.zonal.forecast.interval.max("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
x = grid.zonal.forecast.interval.max("Brams15km", "12h", "2h", b1)
```

Zonal de Previsão por Intervalo: **Mediana**

Retorna a mediana dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

grid.zonal.forecast.interval.median("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.interval.median("Brams15km", "12h", "2h", b1)
```

Zonal de Previsão por Intervalo: **Desvio Padrão**

Retorna o desvio padrão dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

grid.zonal.forecast.interval.standard_deviation("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
x = grid.zonal.forecast.interval.standard_deviation("Brams15km", "12h", "2h", b1)
```

Zonal de Previsão por Intervalo: **Variância**

Retorna a variância dos valores dentre os pixels das camadas de previsão que interceptam o objeto monitorado ou sua área de influência (buffer) no intervalo de tempo inicial e final informado no futuro em função da data/hora atual.

Sintaxe:

grid.zonal.forecast.interval.variance("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<buffer>])

Exemplo: b1 = Buffer(BufferType.In_diff, 800, "m")
x = grid.zonal.forecast.interval.variance("Brams15km", "12h", "2h", b1)

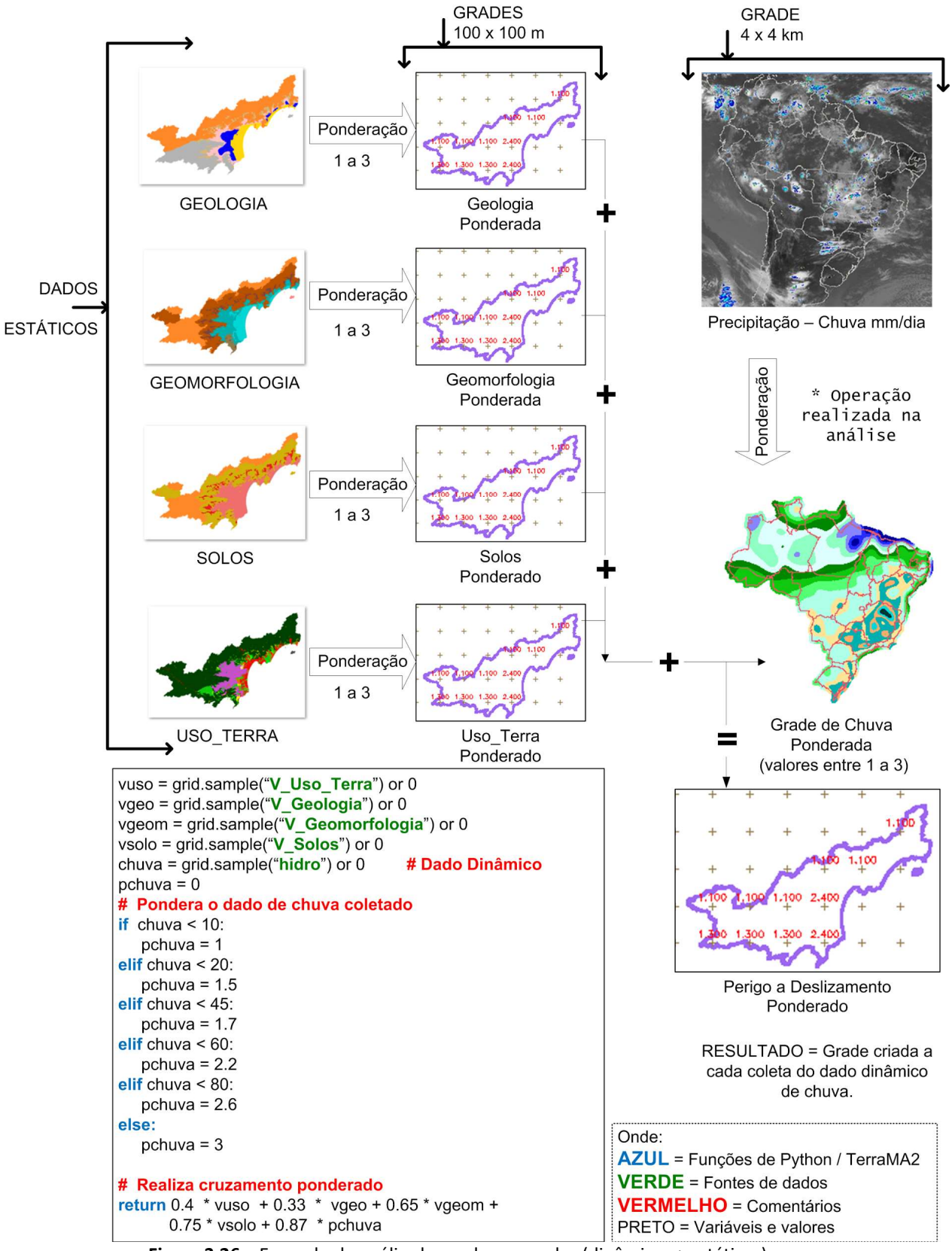
3.4 - ANÁLISES BASEADAS EM GRADES

Em geral, análises baseadas em grade são comparadas as técnicas de inferência espacial realizadas em SIG como cruzamentos de mapas por regras booleanas, ponderadas, entre outras. A diferença desses cruzamentos feitos num SIG tradicional (eminentemente estático) e a plataforma TerraMA^2 é que neste último pelo menos um dos mapas pode ser um dado ambiental coletado dinamicamente na forma de matrizes (grades retangulares). As grades estáticas como dados adicionais também podem ser utilizadas. A saída será uma nova matriz criada na mesma frequência dos dados dinâmicos ou com programação definida pelo usuário.

3.4.1 – EXEMPLO TÍPICO

A seguir apresentamos um exemplo típico da utilização de análise baseada em grades que faz o cruzamento entre quatro mapas matriciais (grades) estáticos e uma grade de precipitação por satélite GOES dinamicamente coletada (Figura 3.26). Assim, no exemplo abaixo considere o seguinte cenário:

1. Os mapas temáticos de **geologia**, **geomorfologia**, **solos** e **uso_terra** foram criados em um Sistema de Informações Geográficas (SIG) e convertidos para grades numéricas com valores (pesos) entre 1 e 3. Os pesos atribuídos às classes de cada mapa temático foram definidos em função da vulnerabilidade a deslizamentos (escorregamentos) de terra em encostas.
2. Os dados matriciais das grades ponderadas dos mapas temáticos estão disponíveis em uma base de dados na forma de arquivos que são identificados como dados adicionais estáticos.
3. A variável ambiental dinâmica a ser cruzada com os mapas temáticos é a precipitação acumulada diária (‘**hidro’**), que também é ponderada com valores entre 1 a 3.
4. A soma ponderada entre os cinco mapas (4 estáticos e 1 dinâmico) foi estabelecida utilizando pesos definidos com ajuda da técnica AHP (Processo Analítico Hierárquico), sendo 0.4 x uso_solo + 0.33 x geologia + 0.65 x geomorfologia + 0.75 x solo + 0.87 x chuva_ponderada, de modo que os valores da grade final fiquem também entre 1 a 3.




```
vuso = grid.sample("V_Uso_Terra") or 0
vgeo = grid.sample("V_Geologia") or 0
vgeom = grid.sample("V_Geomorfologia") or 0
vsolo = grid.sample("V_Solos") or 0
chuva = grid.sample("hidro") or 0        # Dado Dinâmico
pchuva = 0
#  Pondera o dado de chuva coletado
if  chuva < 10:
    pchuva = 1
elif chuva < 20:
    pchuva = 1.5
elif chuva < 45:
    pchuva = 1.7
elif chuva < 60:
    pchuva = 2.2
elif chuva < 80:
    pchuva = 2.6
else:
    pchuva = 3

#  Realiza cruzamento ponderado
return 0.4  * vuso  + 0.33  *  vgeo + 0.65 * vgeom +
        0.75 * vsolo + 0.87  * pchuva
```

Figura 3.26 – Exemplo de análise baseada em grades (dinâmicas + estáticas).

Para uma melhor compreensão da regra de análise do exemplo vamos analisá-la por partes.

```
vuso = grid.sample("V_Uso_Terra") or 0
vgeo = grid.sample("V_Geologia") or 0
vgeom = grid.sample("V_Geomorfologia") or 0
```

```
vsolo = grid.sample("V_Solos") or 0
```

Estas linhas definem uma variável para cada grade numérica correspondente aos mapas temáticos (geologia, geomorfologia, solo e uso da terra) que foram ponderados em um SIG. As variáveis "**vuso**", "**vgeo**", "**vgeom**" e "**vsolo**" recebem os valores dos pontos de cada grade através do operador "**grid.sample**".

```
chuva = amostra("hidro") or 0          # Dado Dinâmico
pchuva = 0
```

O conteúdo da variável "**chuva**" receberá os valores dos pontos da grade do último dado ambiental coletado, no caso, a precipitação acumulada em 24 horas por satélite (fonte de nome "hidro"). Uma outra variável de nome "**pchuva**" será inicializada com valor 0 (zero) para receber o resultado da ponderação a seguir.

```
# Pondera o dado de chuva coletado
if    chuva < 10:
       pchuva = 1
elif chuva < 20:
       pchuva = 1.5
elif chuva < 45:
       pchuva = 1.7
elif chuva < 60:
       pchuva = 2.2
elif chuva < 80:
        pchuva = 2.6
else:
       pchuva = 3
```

Nas linhas acima, utilizamos o comando "if" para realizar a ponderação dos valores de chuva. Se a chuva for < 10 o peso será 1, se chuva for < 20 ou >= 10 o peso será 1.5, se chuva for < 45 ou >= 20 o peso será 1.7, se chuva for < 60 ou >= 45 o peso será 2.2, se chuva for < 80 ou >= 60 o peso será 2.6 e qualquer valor de chuva > 80 o peso será 3.

```
# Realiza o cruzamento ponderado
return = return 0.4  * vuso  + 0.33  *  vgeo + 0.65 * vgeom +
          0.75 * vsolo + 0.87  * pchuva
```

Na linha acima utilizamos o comando "return" para criar a grade de saída ponderada. Note que cada mapa receberá um peso diferente dependendo do propósito do cruzamento, neste caso, o risco a deslizamento de terra. A definição dos pesos pode ser realizada por meio da técnica AHP (Analytic Hierarchy Process) muito comum em sistemas de geoprocessamento.

NOTA: Ao criar uma nova grade dinâmica como resultado desse tipo de análise, o usuário deverá informar na interface qual o sistema de projeção, qual o tamanho (retângulo envolvente) e qual a resolução espacial dessa grade. O dado dinâmico resultante dessa análise poderá ser utilizado em outras análises como qualquer outro dado coletado externamente.

Para uma maior compreensão da sintaxe da linguagem ou uso de opções avançadas, recomenda-se a leitura do anexo A2, a documentação disponível em português em https://wiki.python.org.br ou site oficial em https://www.python.org/ .

A seguir serão apresentadas em detalhes as opções disponíveis para acesso aos dados desse tipo de análise, os operadores históricos e os de previsão.

3.4.2 – EDITANDO ANÁLISE BASEADAS EM GRADES

A Figura 3.27 mostra a área de trabalho utilizada para se definir uma análise baseada em dados matriciais (grades). Nesta interface deve-se definir um nome único para este tipo de análise, onde será armazenado a grade de saída, quais as grades de entrada serão utilizadas, a programação para execução de análises, configuração da grade de saída e do modelo de análise escrito em Python. Descrevemos a seguir cada um dos campos dessa interface.

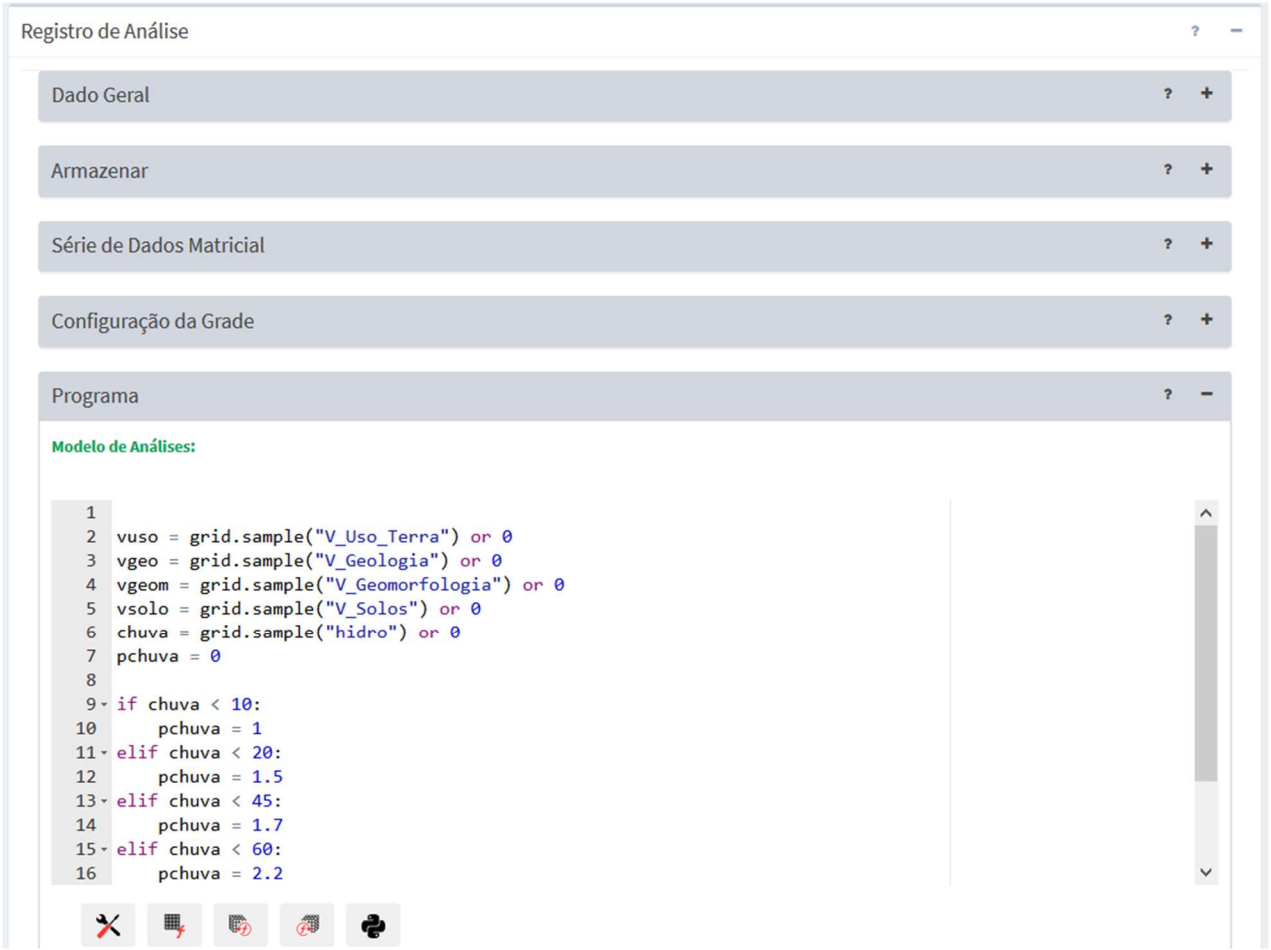


Figura 3.27– Análise com base em dados matriciais.

Registro de Análise – Dado Geral:

- **Nome**: Defina o nome da análise (campo obrigatório). O tamanho máximo do nome é de 100 caracteres. Não é permitido nomes duplicados.
- **Tipo**: Escolha o tipo "**Grade"**. A opção "Objeto Monitorado" está descrita no item 3.3. e "PCD" no item 3.5. IMPORTANTE: Após salvar a análise o tipo não poderá ser alterado.
- **Descrição**: Campo não obrigatório para descrição da análise. O tamanho máximo do texto é de 250 caracteres.
- **Serviço**: Escolha o serviço de análise que estará associado a cada análise. Se necessário adicionar novos serviços de análise (local ou remoto) consulte Capítulo 2 – item 2.3.
- **Ativo**: Botão ativo executará a análise de acordo com a programação (ver abaixo) definida para a análise. Se o botão estiver desmarcado a análise não será executada. Uma análise que não esteja ativa poderá ser executada apenas manualmente pelo botão "► Executar" disponível na lista de análises da área de trabalho.

Registro de Análise – Armazenar:

Utilize os parâmetros desta seção para definir o local de armazenamento dos dados. No caso de uma análise baseada em grades, será solicitado um servidor do tipo arquivo previamente definido (um diretório) para armazenar os arquivos matriciais dinamicamente criados.

- **Formato de saída**: Para este tipo de análise apenas a opção "Matriz Geotiff" encontra-se disponível.
- **Armazenar Dados**: Escolha o servidor de arquivos (FILE) que irá armazenar as matrizes de saída.
- **Máscara**: Máscara do nome dos arquivos a serem armazenados. Essa máscara utiliza partes constantes para capturar prefixos utilizados nos nomes dos arquivos e sequências especiais para indicar ao sistema como interpretar informações de data e hora contidas nos nomes dos arquivos. Utilizar %YYYY para anos de 4 dígitos, %YY para anos de 2 dígitos, %MM para mês, %DD para dia, %hh para hora, %mm para minutos e %ss para segundos. Se desejar informar um subdiretório a partir do servidor de arquivos escolhido, forneça o nome do subdiretório na frente do nome da máscara. Note que ano, mês e dia as letras devem ser **maiúsculas** e para hora, minuto e segundo as letras devem ser **minúsculas**.

Exemplo: Arquivos a serem armazenados no subdiretório "risco_fogo" com prefixo risco, seguido da data e hora como os arquivos abaixo "risco200805271030.tif", a máscara a ser utilizada deve ser : "risco_fogo/risco%YYYY%MM%DD%hh%mm.tif"

- **Fuso Horário**: Fuso horário do dado dinâmico foi gerado. Esse parâmetro garante que as datas e horas dos dados coletados se mantenham consistentes com a base de dados. Dados em horário GMT devem utilizar o valor 0.
- **Projeção (SRID)**: Valor numérico dos parâmetros de projeção e datum dos dados a serem armazenados. Obrigatório para análise tipo "Dados Matriciais". Valores mais utilizados ver no anexo A1.

Registro de Análise – Agendamento

Nesta seção o usuário deve definir quando será executada a análise.

- **Tipo**: Escolha tipo "**Manual**", "**Agendamento**", "**Reprocessamento de dados históricos**", ou "**Automático**". Se "Manual" a execução da análise só será realizada se o usuário utilizar o botão "► Executar" no item da lista de análises que desejar, ou ainda em "Salvar e executar" da análise aberta. Se "**Agendamento**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua. Se "**Reprocessamento de dados históricos**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua, porém em um período inicial e final no passado. Se "**Automático**" dependerá da chegada de qualquer dos dados dinâmicos que uma análise utilizar.

- **Data Inicial** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para início do reprocessamento.

- **Data Final** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para fim do reprocessamento.

- **Unidade de tempo**: Escolha um item entre "Segundos, Minutos, Horas e Semanalmente".

- **Frequência** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Digite um valor de um número inteiro.

- **Tempo Inicial** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Clique no campo para escolher o valor de hora, minuto e segundo que será utilizado como referência para executada a coleta e armazenamento do dado dinâmico.

- **Agendamento** *(somente se Unidade de tempo for Semanalmente)*: escolha uma das opções entre "Domingo, Segunda-feira, Terça-feira, Quarta-feira, Quinta-Feira, Sexta-feira e Sábado"

- **Hora** *(somente se Unidade de tempo for Semanalmente)*: clique no campo para escolher o valor de hora, minuto e segundo que será executada a coleta e armazenamento do dado dinâmico.

Registro de Análise – Série de Dados Matriciais

Nesta seção o usuário deve escolher qual ou quais dados estáticos (matriciais somente) ou dinâmicos (Matriz) serão cruzados (ou sobrepostos espacialmente) para gerar um novo dado matricial dinâmico.

- **+ :** Clique no botão para selecionar um dado estático ou dinâmico na janela que será apresentada.

    - **Dado Estático**: Clique para abrir a lista de dados estáticos matriciais a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.

- **Dado Dinâmico**: Clique para abrir a lista de dados dinâmicos matriciais a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.

Após a inclusão de um dado na lista, o campo de pseudônimo pode ser alterado. Use o botão "X Remover" para excluir um dado da lista (Figura 3.28).

- **Pseudônimo:** Ao escolher um dado estático ou dinâmico os conteúdos dos campos **Nome** e **Pseudônimo** são iguais. Clique no campo correspondente que deseja alterar. Os operadores nas regras de análise farão uso dos conteúdos apresentados nos pseudônimos (ver item 3.4.3).

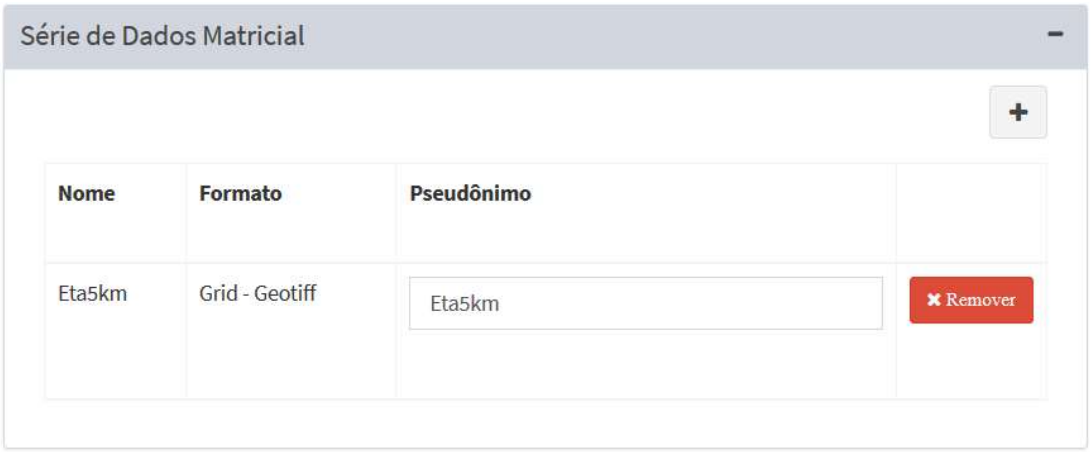


Figura 3.28 – Módulo de Administração: Análise – Lista de Dados Adicionais

Registro de Análise – Configuração da Grade

Nesta seção o usuário deve definir o tamanho da área da matriz de saída, resolução e o interpolador para ajustes da resolução entre os dados de entrada e saída.

- **Método de Interpolação**: Escolha o método de interpolação, se "Vizinho mais próximo", "Bi linear" ou "Bi cúbico".
  - **Interpolação nula:** Digite o valor desejado para valores nulos da matriz de saída.
- **Área de Interesse**: Escolha como será definida a área da matriz de saída, se "União", a área de interesse corresponde a área da união de duas ou mais imagens inseridas na série de dados matriciais, "Mesmo da série de dados" ou "Personalizada", parâmetros descritos abaixo.
  - **Série de Dados** *(somente se Área de interesse for "Mesmo da série de dados")*: Escolha um dos dados matriciais selecionados para a análise utilizar como referência do tamanho da área para a matriz de saída".
  - **X min** *(somente se Área de interesse for "Personalizada")*: Digite o valor mais a esquerda da área. Valor depende do sistema de projeção utilizado.
  - **Y min** *(somente se Área de interesse for "Personalizada")*: Digite o valor mais abaixo da área. Valor depende do sistema de projeção utilizado.

- **X max** *(somente se Área de interesse for "Personalizada")*: Digite o valor mais a direita da área. Valor depende do sistema de projeção utilizado.

- **Y max** *(somente se Área de interesse for "Personalizada")*: Digite o valor ponto mais acima da área. Valor depende do sistema de projeção utilizado.

- **Projeção SRID** *(somente para Filtrar for "Personalizada")*: Valor numérico dos parâmetros de projeção e datum a ser utilizado pelo par de coordenadas acima. Valores mais utilizados ver no anexo A1.

- **Resolução**: Escolha o tamanho do pixel, se "Grade menor", "Grade maior", "Mesmo da série de dados" ou "Personalizada".

  - Série de Dados *(Somente se Resolução for "Mesmo da série de dados")*: Escolha um dos dados matriciais selecionados para a análise como referência do tamanho da área para a matriz de saída".

  - X: Y: *(Somente se Resolução for "Personalizada")*: Digite os valores do tamanho do pixel. Este deve ser determinado conforme a projeção SRID definida anteriormente, ou seja, em graus decimais se a projeção for em coordenadas geográficas, ou metros se a projeção SRID for em coordenadas planas (UTM, cônica, polar, etc.). Valores mais utilizados ver no anexo A1.

Registro de Análise – Programa

Nesta seção o usuário deve editar o programa de análise. A edição do programa utiliza a linguagem Python, assim siga rigorosamente a sintaxe do comando definidos para esta linguagem. Além dos comandos e funções de Python você pode utilizar os utilitários e os operadores criados especialmente para a plataforma TerraMA².

Para facilitar a edição do programa, botões na parte inferior da janela permite escolher atalhos de alguns itens específicos. Ao escolher um item entre os botões disponíveis o conteúdo será incluído na posição em que estiver o cursor. Os atalhos disponíveis são:

 - Atalho para os utilitários da plataforma, tais como "Get date" que recupera a data/hora de execução da análise, unidades de distância e unidades de tempo.

 - Atalho para o operador "Amostra" que recupera os "pixels" da atual matriz dinâmica ou de uma matriz estática.

 - Atalho para os operadores históricos que trabalham com dados dinâmicos matriciais de observação. Os operadores utilizam o dado atual e o passado disponível.

 - Atalho para os operadores que trabalham com dados dinâmicos matriciais de previsão. Os operadores utilizam o dado atual e o futuro disponível.

 - Atalho para algumas funções, operadores e comandos de Python.

Após editar o programa, poderá utilizar o botão "Validar" para identificar se há erros de sintaxe nos comando, funções e operadores utilizados. O botão "Salvar e executar" grava as últimas alterações e executa a análise mesmo que esta esteja inativa. Se

desejar apenas gravar as alterações clique na seta do botão e escolha "Salvar". Para as análises que estiverem ativas as próximas execuções seguirão as regras definidas na seção "Agendamento".

3.4.3 – OPERADORES PARA ANÁLISES BASEADAS EM GRADES

Os operadores espaciais disponíveis para serem utilizados com os dados matriciais são utilizados para dados dinâmicos matriciais de observação ou previsão, assim como para dados estáticos também matriciais. A Figura 3.29 mostra que há um grupo de operadores que operam sobre um dado estático matricial ou um conjunto de dados matriciais dinâmicos para a grades de observação ou camadas de previsões mais recente, um grupo de operadores históricos com dados dinâmicos de observações e outro grupo de operadores sobre dados dinâmicos de previsão.

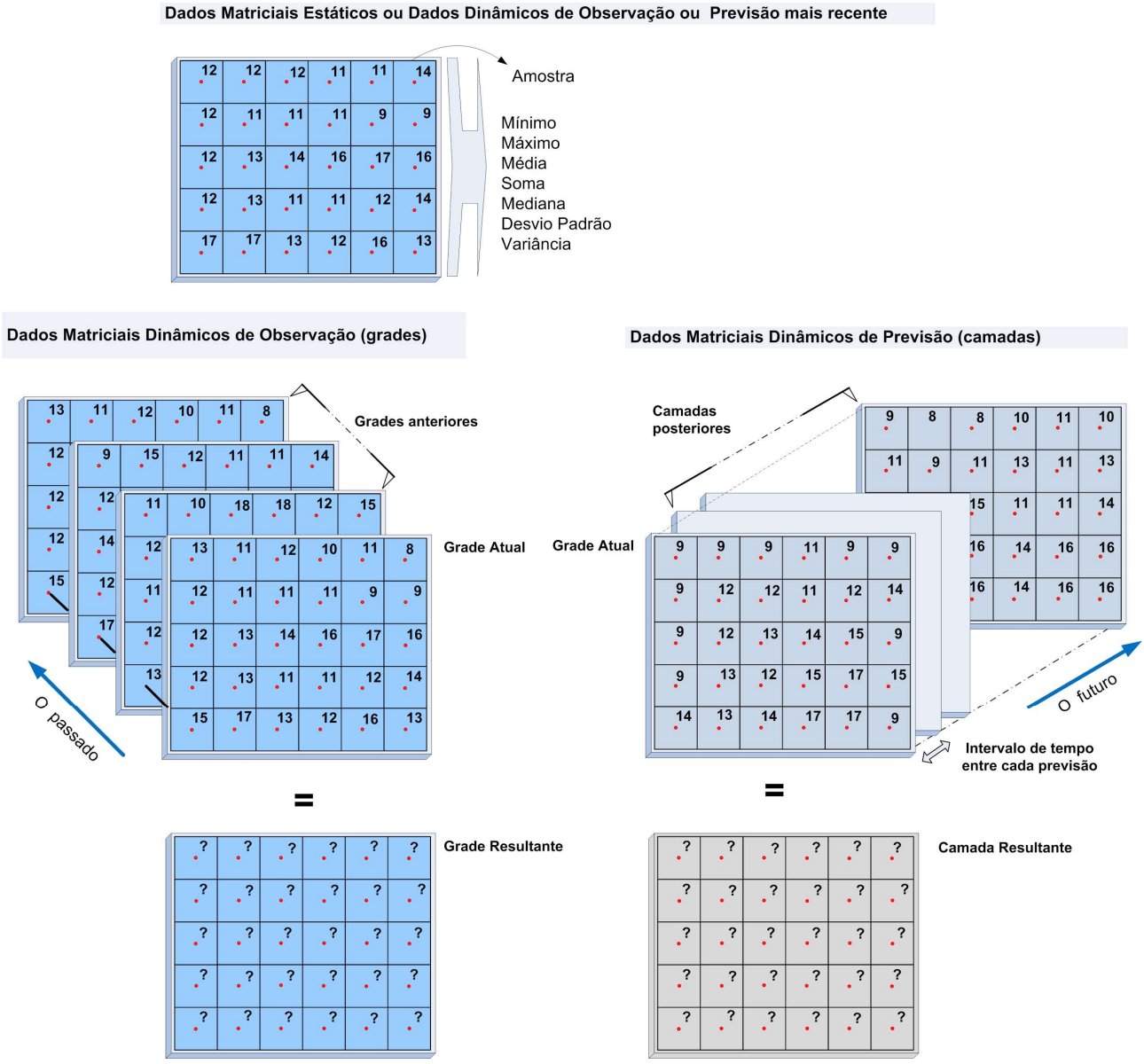


Figura 3.29– Análise com base em dados matriciais: Operações sobre grades/camadas estáticas ou dinâmicas.

3.4.3.1- Utilitários para operadores com dados matriciais

Estes utilitários são os mesmos apresentados na análise baseada em objeto monitorado, portanto, maiores detalhes veja o item 3.3.3.1.

I.1- Unidade de distância

Para operadores que utilizam unidades de distância devem usar as letras entre aspas duplas (“<unidade>”). As seguintes opções estão disponíveis:

- “cm”: centímetros
- “m” : metros
- “km”: quilômetros

I.2- Unidade de tempo

Para operadores que utilizam unidades de tempo devem usar as letras imediatamente após o valor numérico, ambos entre aspas dupla (“<num><unidade>”). As seguintes opções estão disponíveis:

- s: Second – tempo em segundos a parir da data/hora atual.
- min: Minute – tempo em minutos a partir da data/hora atual.
- h: Hour – tempo em horas a partir da data/hora atual.
- d: Day – tempo em dias a partir da data/hora atual.
- d+: Day (Extended) – tempo em dias a partir da data/hora atual até a zero horas do número de dias.
- w: Week – tempo em semanas a partir da data/hora atual.
- w+: Week (Extended) – tempo em semanas a partir da hora atual até a zero horas do número de semanas.

I.3- Utilitário “ Get analysis date”

- **get_analysis_date()** : retorna a data/hora de execução da análise, podendo ser um reprocessamento de dado histórico ou valor atual.

3.4.3.2 – Operador Matriz Atual

Operador matriz atual é utilizado para obter os valores dos “pixels” de um dado matricial, dinâmico ou estático.

II.1- Amostra

Retorna os valores da grade mais recente em relação a hora atual no caso de uma série de dados históricos de observação ou da camada mais recente de previsão. Para dados matriciais estático retorna os valores da grade própria grade.

SINTAXE GERAL:

grid.sample("<dynamic_data_grid>", [<band>])

onde:

- **dynamic_data_grid** : String com o nome da série de dados dinâmicos matriciais ou dados estáticos matriciais.
- **band** : [Opcional] Banda da grade utilizada. Se não informado será considerado a primeira banda (0).

Exemplo:
```
x = grid.sample("hidroestimador")      # matriz dinâmica
y = grid.sample("declividade")         # matriz estática
```

3.4.3.3 – Operadores Históricos de Observação

Operadores históricos de observação são operadores utilizados para obter estatísticas sobre os pontos da grade, produzindo um novo valor da grade de saída.

A Figura 3.30 ilustra um conjunto de dados matriciais dinâmicos coletados sistematicamente que são utilizados para obter um novo valor em cada ponto da grade a partir de um operador escolhido pelo usuário, como por exemplo calcular o mínimo, máximo, média, etc.

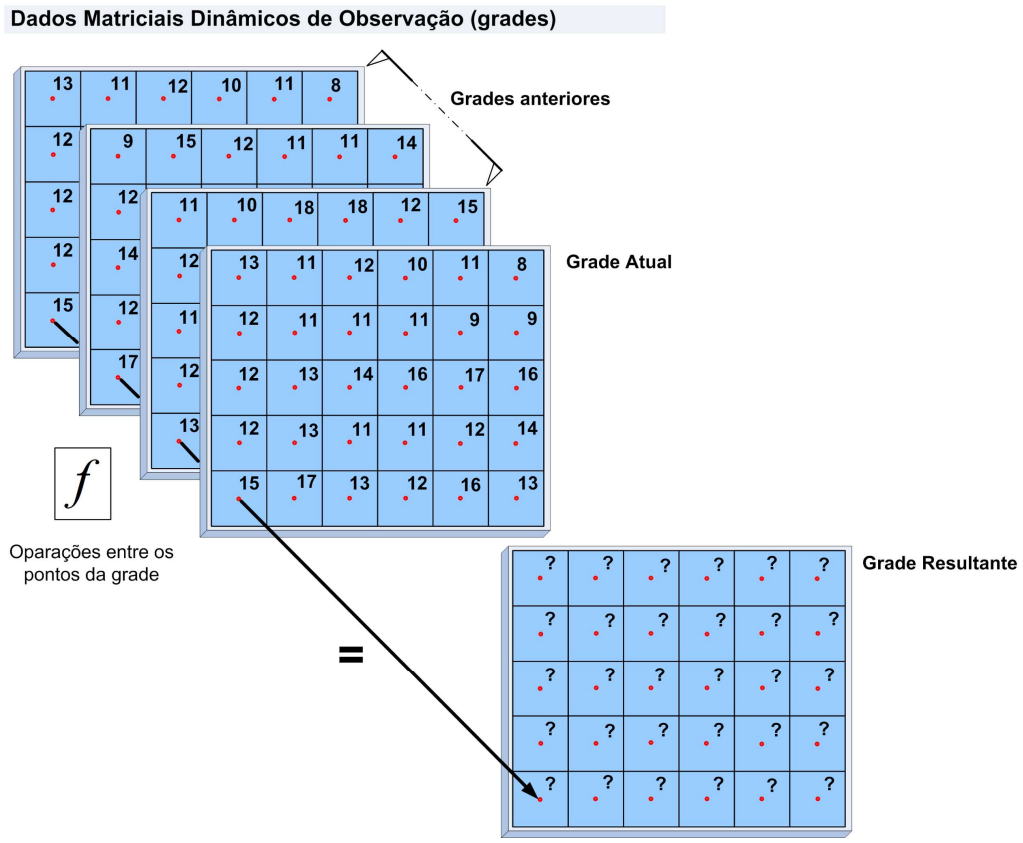


Figura 3.30 – Análise com base em dados matriciais: Uso de operadores sobre grades dinâmicas de observação.

Estes operadores são divididos em dois tipos: **Histórico** e **Histórico por Intervalo**. A descrição de cada tipo a seguir.

III.1- Histórico

Grupo de operadores que retornam valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

grid.history.<operator>("<dynamic_data_grid>", "<time>", [<band>])

onde:

- **operator** : min, max, mean, sum, median, standard_deviation, variance;
- **dynamic_data_grid** : String com o nome da série de dados matriciais .
- **time** : String com o intervalo de tempo, a partir da hora atual. Ver utilitário Unidades de tempo.
- **band** : [Opcional] Banda da grade utilizada. Se não informado será considerado a primeira banda (0).

Segue a descrição de cada operador.

Histórico : **Mínimo**

Retorna os menores valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.min(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.min("chuva", "1d", 0)

Histórico : **Máximo**

Retorna os maiores valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.max(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.max("chuva", "1d", 0)

Histórico : **Média**

Retorna as médias dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.mean(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.mean("chuva", "1d", 0)

Histórico : **Soma**

Retorna as somas em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.sum(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.sum("chuva", "1d", 0)

Histórico : **Mediana**

Retorna as medianas em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.median(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.median("chuva", "1d", 0)

Histórico : **Desvio Padrão**

Retorna os desvios padrões em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.standard_deviation(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.standard_deviation("chuva", "1d", 0)

Histórico : **Variância**

Retorna as variâncias em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

grid.history.variance(“<dynamic_data_grid>”, “<time>”, [<band>])

Exemplo: x = grid.history.standard_deviation("chuva", "24h", 0)

III.2- Histórico por intervalo

Grupo de operadores que retornam os valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual**.**

SINTAXE GERAL:

grid.history.interval.<operator>("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

onde:

- **operator** : min, max, mean, median, sum, standard_deviation, variance;
- **dynamic_data_grid** : String com o nome da série de dados matriciais;
- **time_begin** : String inicial (mais antigo) do intervalo de tempo para filtrar as grades. Este valor será aberto (< tempo mais antigo) no tempo informado;
- **time_end** : String final (mais recente) do intervalo de tempo para filtrar as grades. Este valor será fechado (<= tempo mais recente) no tempo informado;
- **band** : [Opcional] Banda da grade a ser utilizada. Se não informado será considerado a primeira banda (0);

Segue a descrição de cada operador.

Histórico por intervalo : **Mínimo**

Retorna os menores valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.min("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.min("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Máximo**

Retorna os maiores valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.max("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.max("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Média**

Retorna as médias dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.mean("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.mean("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Mediana**

Retorna as medianas dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.median("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.median("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Soma**

Retorna as somas dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.sum("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.sum("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Desvio padrão**

Retorna os desvios padrões dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.standard_deviation("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.standard_deviation("chuva", "10d", “5d”, 0)

Histórico por intervalo : **Variância**

Retorna as variâncias dos valores em cada ponto da grade sobre dados matriciais históricos no intervalo de tempo inicial e final informado no passado em relação a data/hora atual.

Sintaxe:

grid.history.interval.variance("<dynamic_data_grid>", "<time_begin>", "<time_end>", [<band>])

Exemplo: x = grid.history.interval.variance("chuva", "10d", “5d”, 0)

3.4.3.4 – Operadores de Previsão

Operadores de previsão são operadores utilizados para obter estatísticas sobre os pontos da grade, produzindo um novo valor da grade de saída.

A Figura 3.31 ilustra um conjunto de camadas de um dado matricial dinâmico coletado sistematicamente para serem utilizados para obter um novo valor em cada ponto da grade a partir de um operador escolhido pelo usuário, como por exemplo calcular o mínimo, máximo, média, etc.

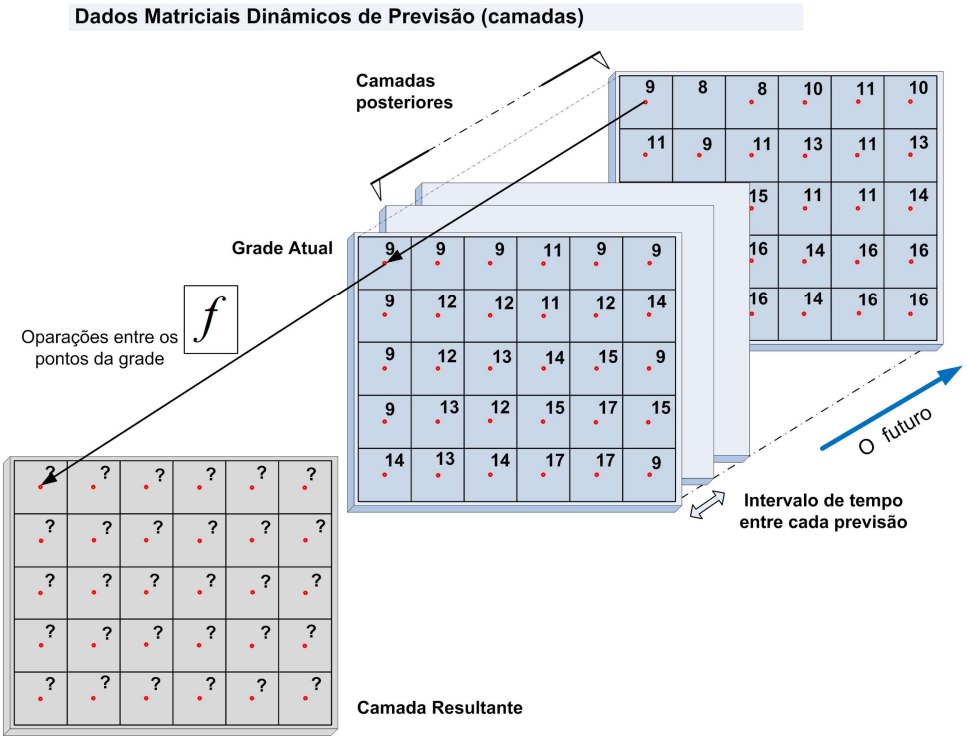


Figura 3.31 – (a) Exemplo dos pontos de uma grade.

OBS: As camadas de um dado de previsão numérica de tempo produzida pelo CPTEC-INPE são atualizadas duas vezes ao dia, a zero horas e do meio dia.

Estes operadores são divididos em dois tipos: **Previsão** e **Previsão por Intervalo**. A descrição de cada tipo a seguir.

IV.1- Previsão

Grupo de operadores que consideram os valores dos "pixels" dos dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

SINTAXE GERAL:

grid.forecast.<operator>("<dynamic_data_grid>", "<time>")

onde:

- **operator** : min, max, mean, median, sum, standard_deviation, variance;
- **dynamic_data_grid** : String com o nome da série de dados matriciais .
- **time** : String com o intervalo de tempo, a partir da hora atual. Ver utilitário Unidades de tempo.

Segue a descrição de cada operador.

Previsão: **Mínimo**

Retorna os menores valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.min(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.min("precipitacao", "1d")

Previsão: **Máximo**

Retorna os maiores valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.max(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.max("precipitacao", "1d")

Previsão: **Média**

Retorna as médias dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.mean(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.mean("precipitacao", "1d")

Previsão : **Soma**

Retorna as somas dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.sum(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.sum("precipitacao", "1d")

Previsão : **Mediana**

Retorna as medianas dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.median(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.median("precipitacao", "1d")

Previsão: **Desvio Padrão**

Retorna os desvios padrões dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.standard_deviation(“<dynamic_data_grid>”, “<time>”)

Exemplo: x = grid.forecast.standard_deviation("precipitacao", "1d")

Previsão: **Variância**

Retorna as variâncias dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no futuro.

Sintaxe:

grid.forecast.variance("<dynamic_data_grid>", "<time>")

Exemplo: x = grid.forecast.standard_deviation("precipitacao", "24h")

IV.2- Previsão por intervalo

Grupo de operadores que consideram os valores dos "pixels" dos dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

SINTAXE GERAL:

grid.forecast.interval.<operator>("<dynamic_data_grid>", "<time_begin>", "<time_end>")

onde:

- **operator**: min, max, mean, median, sum, standard_deviation, variance;
- **dynamic_data_grid**: String com o nome da série de dados matriciais de previsão;
- **time_begin**: String inicial (mais próximo da hora atual) do intervalo de tempo para filtrar as camadas de previsão. Este valor será fechado (<= tempo mais próximo) no tempo informado;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar as camadas de previsão. Este valor será aberto (< tempo mais distante) no tempo informado;

Segue a descrição de cada operador.

Previsão por intervalo: **Mínimo**

Retorna os menores valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.min("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.min("precipitacao", "10d", "5d")

Previsão por intervalo: **Máximo**

Retorna os maiores valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.max("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.max("precipitacao", "10d", "5d")

Previsão por intervalo: **Média**

Retorna as médias dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.mean("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.mean("precipitacao", "10d", “5d”)

Previsão por intervalo: **Mediana**

Retorna as medianas dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.median("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.median("precipitacao", "10d", “5d”)

Previsão por intervalo: **Soma**

Retorna as somas dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.sum("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.sum("precipitacao", "10d", “5d”)

Previsão por intervalo: **Desvio padrão**

Retorna os desvios padrões dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.standard_deviation("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.standard_deviation("precipitacao", "10d", “5d”)

Previsão por intervalo: **Variância**

Retorna as variâncias dos valores em cada ponto da grade sobre dados matriciais de previsão no intervalo de tempo inicial e final informado no futuro em relação a data/hora atual.

Sintaxe:

grid.forecast.interval.variance("<dynamic_data_grid>", "<time_begin>", "<time_end>")

Exemplo: x = grid.forecast.interval.variance("precipitacao", "10d", “5d”)

3.5 - ANÁLISES BASEADAS EM PCD

Análises baseadas em PCD utilizam os valores dos atributos (atuais e históricos) para tomada de decisões através de um conjunto de operadores próprios para este tipo de análise. Porém, as PCD´s podem ser utilizadas como um mapa de pontos estáticos que são cruzados com outros dados dinâmicos da mesma forma que uma análise baseada em objetos monitorados. Neste caso, a sobreposição das PCD´s com os dados dinâmicos é realizada com uso dos operadores geográficos apresentados no item 3.3.3 acima. Todos os operadores foram criados exclusivamente para uso na plataforma, juntamente com os recursos que a linguagem de programação python proporciona.

Os resultados desses operadores são armazenados em variáveis locais do python para serem armazenadas em uma nova tabela de atributos que estará associada a tabela de localização das PCD´s. Nessa nova tabela são armazenados ainda a data/hora toda vez que é realizada a análise.

3.5.1 – EXEMPLO TÍPICO

A seguir apresentamos um exemplo típico da utilização de análise baseada em PCD . Assim, no exemplo abaixo (figura 3.32) considere o seguinte cenário:

1. A série de dados dinâmicos do tipo PCD recebe as medidas de precipitação com frequência de 1 hora que serão utilizadas para calcular o valor acumulado no período de 24 horas.
2. Os dados dinâmicos matriciais com valores de precipitação estimada por satélite serão adquiridos com frequência de 30 minutos, porém os valores destes estão em mm/h (dados “hidro”).
3. O valor acumulado em 24 horas de precipitação por satélite será calculado para uma área de influência de 3 km em torno de cada PCD.
4. Ambos valores de precipitação acumulada em 24 horas, medido pela PCD e calculado a partir das imagens de satélite, serão armazenados como resultado para posterior uso.

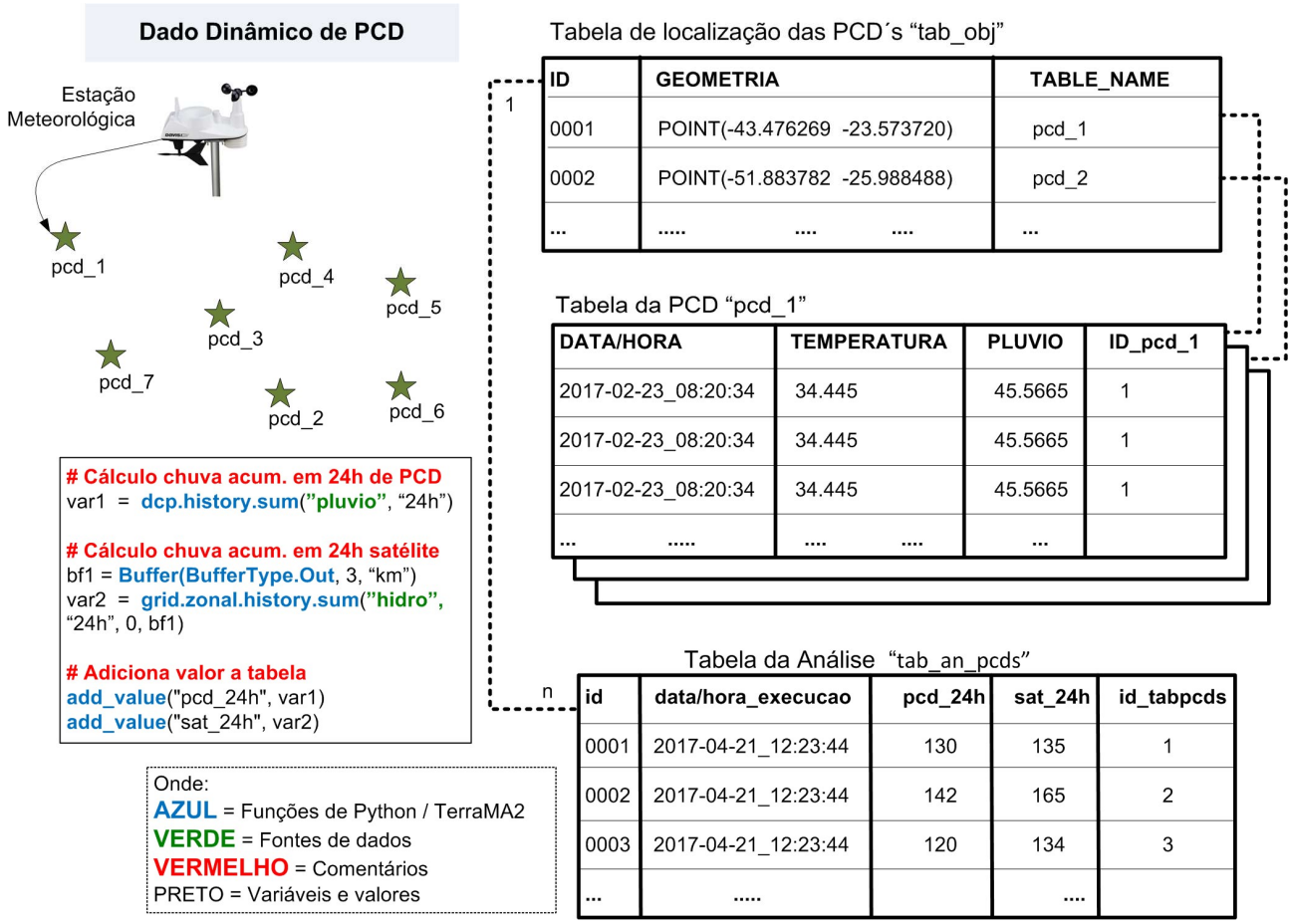


Figura 3.32 – Exemplo de análise com PCD.

Para uma melhor compreensão do programa de análise do exemplo, vamos analisar as partes do programa da figura.

```
# Cálculo chuva acum. em 24h de PCD
var1 = dcp.history.sum("pluvio", "24h")
```

Essa linha define uma nova variável denominada de "**var1**" que será inicializada com o valor do operador "**soma**" aplicado a cada uma das PCD´s. Observe que "**pluvio**" é o nome do atributo que contém as informações de chuva horária e deve estar cadastrado em cada PCD.

```
# Cálculo chuva acum. em 24h satelite
bf1 =  Buffer(BufferType.Out, 3, "km")
var2 = grid.zonal.history.sum("hidro", "24h", 0, bf1)
```

Essas duas linhas definem o tipo de buffer a ser aplicado a cada PCD. A variável "**bf1**" define um buffer de 3 km externo a geometria em questão, no caso em torno dos pontos de cada PCD. A variável "**var2**" utiliza a área do buffer de cada PCD para sobrepor a um conjunto de dados matriciais dinâmicos de nome "**hidro**" num período de 24 horas passadas a partir da hora atual.

```
# Adiciona valor a tabela
add_value("pcd_24h", var1)
add_value("sat_24h", var2)
```

Nas duas linhas acima são definidos os nomes dos atributos que serão utilizados para armazenar os resultados das variáveis "**var1**" e "**var2**", isto é, nos atributos "**pcd_24h**" e "**sat_24h**" respectivamente. O nome da tabela que deverá receber os dois atributos é um dos parâmetros definidos para este tipo de análise.

Outro ponto importante a ser observado no exemplo, é a presença de comentários que não tem qualquer significado na execução do programa, apenas para fins de documentação. Use o sinal # para adicionar uma linha de comentário.

Ainda na Figura 3.32 mostra a tabelas envolvidas nesse tipo de análise. A tabela "**tab_obj**" mantém a localização de cada PCD da série de dados dinâmico definida. As tabelas "**pcd_1**", "**pcd_2**" entre outras armazenam em cada uma os dados coletados de cada PCD. A tabela "**tab_an_pcds**" é criada pela análise e armazena os resultados de todas as PCD´s.

Para uma maior compreensão da sintaxe da linguagem ou uso de opções avançadas, recomenda-se a leitura do anexo A2, a documentação disponível em português em https://wiki.python.org.br ou site oficial em https://www.python.org/ .

3.5.2 – EDITANDO ANÁLISE BASEADAS EM PCD

A Figura 3.33 mostra a área de trabalho utilizada para se definir uma análise baseada em PCD. Nesta interface deve-se definir um nome único para este tipo de análise, quais planos de entrada serão utilizados (NÃO OBRIGATÓRIO), a programação para execução de análises, e do modelo de análise escrito em Python. Descrevemos a seguir cada um dos campos dessa interface.

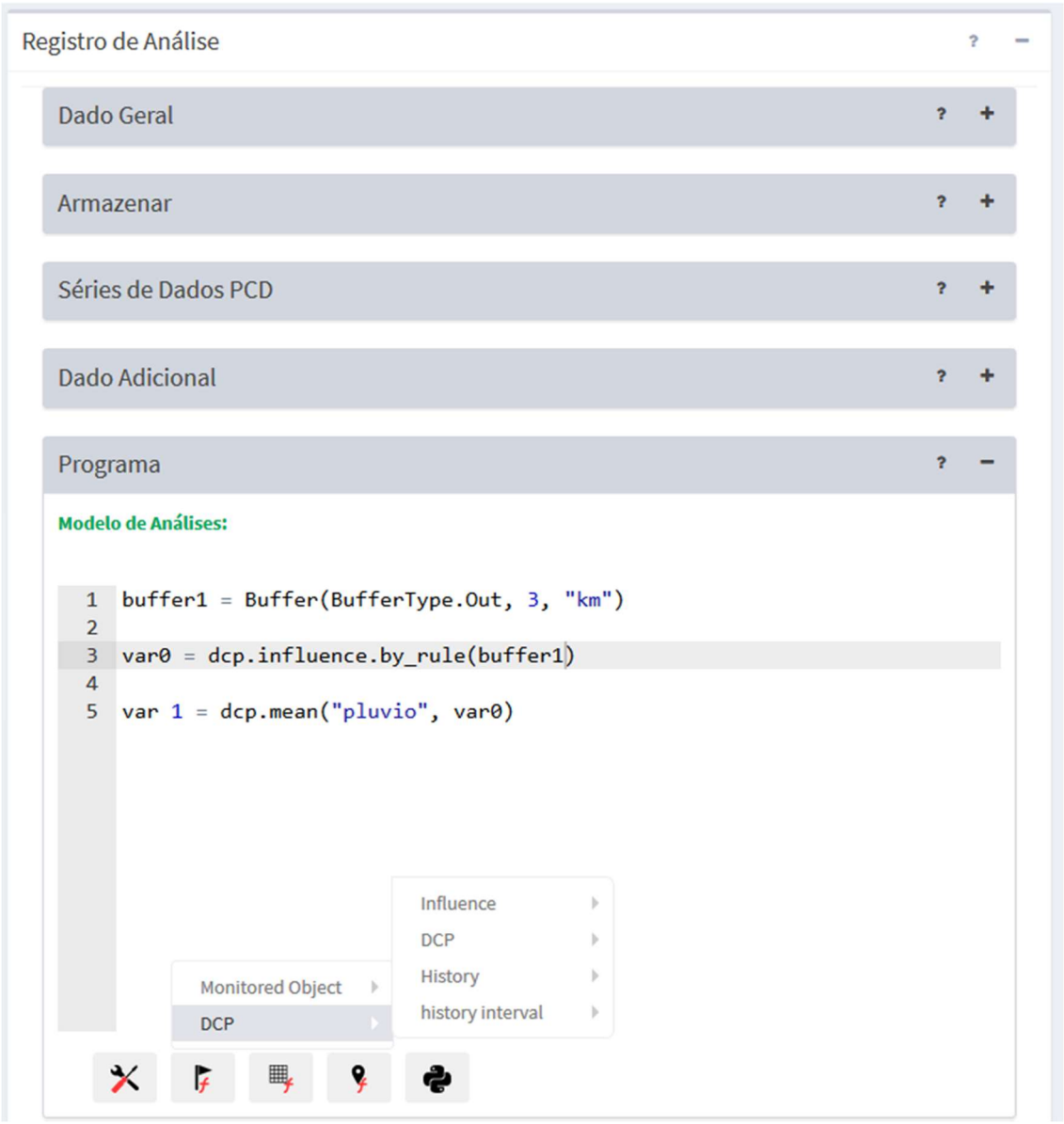


Figura 3.33– Análise com base em dados de PCD.

Registro de Análise – Dado Geral:

- **Nome**: Defina o nome da análise (campo obrigatório). O tamanho máximo do nome é de 100 caracteres. Não é permitido nomes duplicados.
- **Tipo**: Escolha o tipo "**PCD"**. A opção "**Grade**" está descrita no item 3.4 e "Objeto Monitorado" no item 3.3. IMPORTANTE: Após salvar a análise o tipo não poderá ser alterado.
- **Descrição**: Campo não obrigatório para descrição da análise. O tamanho máximo do texto é de 250 caracteres.
- **Serviço**: Escolha o serviço de análise que estará associado a cada análise. Se necessário adicionar novos serviços de análise (local ou remoto) consulte Capítulo 2 – item 2.3.
- **Ativo**: Botão ativo executará a análise de acordo com a programação (ver abaixo) definida para a análise. Se o botão estiver desmarcado a análise não será executada. Uma análise que não esteja ativa poderá ser executada apenas manualmente pelo botão "►Executar" disponível na lista de análises da área de trabalho.

Registro de Análise – Armazenar

Utilize os parâmetros desta seção para definir o local de armazenamento dos dados. No caso de uma análise baseada em PCD, será solicitado o nome de uma tabela de banco de dados.

- **Formato de saída**: Para este tipo de análise apenas a opção "Análise de Objeto Monitorado" encontra-se disponível.
- **Nome da tabela**: Digite o nome da tabela a ser criada para armazenar os resultados. Esta tabela tem um relacionamento de "n" para "1" com a tabela de localização das PCDs. Em outras palavras, toda vez que a análise for executada será armazenado data/hora e resultados dos cálculos para cada PCD.

Registro de Análise – Armazenar - Agendamento

Nesta seção o usuário deve definir quando será executada a análise.

- **Tipo**: Escolha tipo "**Manual**", "**Agendamento**", "**Reprocessamento de dados históricos**", ou "**Automático**". Se "Manual" a execução da análise só será realizada se o usuário utilizar o botão "► Executar" no item da lista de análises que desejar, ou ainda em "Salvar e executar" da análise aberta. Se "**Agendamento**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua. Se "**Reprocessamento de dados históricos**" a execução da análise será por intervalos pré-definidos e em um tempo inicial de forma continua, porém em um período inicial e final no passado. Se "**Automático**" dependerá da chegada de qualquer dos dados dinâmicos que uma análise utilizar.

NOTA: Em todas opções do agendamento a tabela da análise armazena de forma contínua os resultados, exceto em "**Reprocessamento de dados históricos**" que a cada execução da análise os registros serão apagados para que os valores sejam atualizados.

- **Data Inicial** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para início do reprocessamento.
- **Data Final** *(somente se Tipo for "Reprocessamento de dados históricos")*: Clique no campo para escolher a data e hora que será utilizada para fim do reprocessamento.
- **Unidade de tempo**: Escolha um item entre "Segundos, Minutos, Horas e Semanalmente".
- **Frequência** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Digite um valor de um número inteiro.
- **Tempo Inicial** *(somente se Unidade de tempo for Segundos, Minutos, Horas)*: Clique no campo para escolher o valor de hora, minuto e segundo que será utilizado como referência para executar a análise.
- **Agendamento** *(somente se Unidade de tempo for Semanalmente)*: escolha uma das opções entre "Domingo, Segunda-feira, Terça-feira, Quarta-feira, Quinta-Feira, Sexta-feira e Sábado"

- **Hora** *(somente se Unidade de tempo for Semanalmente)*: clique no campo para escolher o valor de hora, minuto e segundo que será executada para iniciar a análise.

Registro de Análise – Série de Dados PCD

Nesta seção o usuário deve escolher qual será a série de dados dinâmicos de PCD previamente cadastrada (ver Capítulo 2 – item 2.6.2).

- **Série de Dados**: Escolha um dado dinâmico de PCD previamente cadastrado como dado dinâmico.

Registro de Análise – Dado Adicional

Além dos dados dinâmicos da própria PCD, nesta seção o usuário pode escolher qual ou quais dados estáticos (matriciais), dinâmicos (matriciais), ou resultado de análises (Grades e Objetos monitorados) serão cruzados (ou sobrepostos espacialmente) com as geometrias das PCD´s.

- **+ :** Clique no botão para selecionar um dado estático ou dinâmico na janela que será apresentada.
    - ➧ **Estático**: Clique para abrir a lista de dados estáticos a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.
    - ➧ **Dinâmico**: Clique para abrir a lista de dados dinâmicos a escolher. Note que uma vez escolhido o mesmo será retirado dessa lista. A lista de dados escolhidos fica disponível na área de trabalho.

Após a inclusão de um dado na lista, o campo de pseudônimo pode ser alterado. Use o botão “X Remover” para excluir um dado da lista (Figura 3.34).

- **Pseudônimo:** Ao escolher um dado a lista automaticamente mostra o nome e o pseudônimo com mesmo conteúdo. Clique no campo correspondente que deseja alterar. Nas regras de análise serão estes pseudônimos que deverão ser utilizados pelo operadores.

Figura 3.34 – Módulo de Administração: Análise – Lista de Dados Adicionais

Registro de Análise – Programa

Nesta seção o usuário deve editar o programa de análise. A edição do programa utiliza a linguagem Python e portanto, siga rigorosamente a sintaxe do comando

definidos para esta linguagem. Além dos comandos e funções de Python você pode utilizar os utilitários e os operadores zonais criados especialmente para a plataforma TerraMA².

Para facilitar a edição do programa, botões na parte inferior da janela possibilitam escolher atalhos de alguns itens específicos. Ao escolher um item entre os botões disponíveis, o conteúdo será incluído na posição em que estiver o cursor. Os atalhos disponíveis são:

 - Atalho para os utilitários da plataforma, tais como "buffer" a ser aplicado em geometrias do objeto monitorado (as PCD´s), unidades de distância, unidades de tempo, "Add value", "Get value" e "Get date".

 - Atalho para os operadores que trabalham com dados dinâmicos de PCD. Além do grupo de operadores que atuam com a série de dados para este tipo de análises, está disponível também o grupo de operadores que consideram as PCD´s como objeto monitorado.

 - Atalho para os operadores que trabalham com dados dinâmicos de matrizes. Disponível nesse tipo de análise caso queira utilizar as PCD´s como objeto monitorado.

 - Atalho para os operadores que trabalham com dados dinâmicos de ocorrências. Disponível nesse tipo de análise caso queira utilizar as PCD´s como objeto monitorado.

 - Atalho para algumas funções, operadores e comandos de Python.

Após editar o programa, poderá utilizar o botão "Validar" para identificar se há erros de sintaxe nos comando, funções e operadores utilizados. O botão "Salvar e executar" grava as últimas alterações e executa a análise mesmo que esta esteja inativa. Se desejar apenas gravar as alterações clique na seta do botão e escolha "Salvar".

Importante: O programa de análise definido pelo usuário é executado individualmente para cada PCD da série de dados. É obrigatório que o programa faça uso pelo menos uma vez do utilitário "add_value".

3.5.3 – OPERADORES DE PCD

Os operadores espaciais disponíveis para serem utilizados com os dados dinâmicos de PCD´s são utilizados sobre os atributos de cada PCD isoladamente ou em grupos seguindo a regra de influência. Cada PCD pode ainda ser utilizada como um objeto monitorado para ser cruzado com qualquer dos três tipos de dados dinâmicos (ver descrição dos operadores no item 3.3.3).

3.5.3.1- Utilitários para operadores sobre PCD

Estes utilitários são os mesmos apresentados na análise baseada em objeto monitorado, portanto, maiores detalhes veja o item 3.3.3.1.

I.1- Unidade de distância

Para operadores que utilizam unidades de distância devem usar as letras entre aspas duplas ("<unidade>"). As seguintes opções estão disponíveis:

- "cm": centímetros
- "m" : metros
- "km": quilômetros

I.2- Unidade de tempo

Para operadores que utilizam unidades de tempo devem usar as letras imediatamente após o valor numérico, ambos entre aspas dupla ("<num><unidade>"). As seguintes opções estão disponíveis:

- s: Second – tempo em segundos a parir da data/hora atual.
- min: Minute – tempo em minutos a partir da data/hora atual.
- h: Hour – tempo em horas a partir da data/hora atual.
- d: Day – tempo em dias a partir da data/hora atual.
- d+: Day (Extended) – tempo em dias a partir da data/hora atual até a zero horas do número de dias.
- w: Week – tempo em semanas a partir da data/hora atual.
- w+: Week (Extended) – tempo em semanas a partir da hora atual até a zero horas do número de semanas.

I.3- Utilitário de "buffer" (Equidistâncias de um objeto)

Nesse tipo de análise é considerada apenas a geometrias de pontos das PCD´s, portanto, só faz sentido utilizar o utilitário "buffer" com as seguintes opções:

- **Buffer( )** : Sem buffer. Será considerado a própria geometria do ponto.
- **BufferType.Out** : Será considerada a área do "buffer" externa à geometria do ponto.
- **Buffer.Type.Level** : Área diferença entre dois buffer externos. Será considerada a diferença entre um buffer maior menos o menor, definindo uma área não adjacente a geometria do ponto.

Nota: Detalhes com exemplos do utilitário "buffer" consulte item 3.3.3.1.

I.4- Utilitários "Get Value", "Add Value" e "Get analysis date"

- **get_value**("<attribute_name>") : recupera o atributo de um objeto monitorado. Válido para atributos numéricos ou alfa-numéricos
- **add_value**("<attribute_name>", <value>) : adiciona o valor de uma variável a um atributo na tabela resultante de uma análise baseada em objeto monitorado. Válido para atributos numéricos ou alfa-numéricos.
- **get_analysis_date()** : retorna a data/hora de execução da análise, podendo ser um reprocessamento de dado histórico ou valor atual.

3.5.3.2- Operadores sobre PCD

II.1- Regra de Influência

A regra de influência é utilizada para definir se serão consideradas outras PCD´s próximas da PCD que estiver sendo analisada.

SINTAXE GERAL:

dcp.influence.by_rule(<buffer>)

onde:

- **buffer** : "Buffer" para ser aplicado a cada PCD, considerada como objeto monitorado. Ver utilitário "buffer" em item 3.3.3.1.

II.2- PCD

Grupo de operadores que consideram as últimas medidas de cada PCD´s.

SINTAXE GERAL:

dcp.<operator>(<buffer>, "<attribute>", [<list_dcp>], [<isActive>])

onde:

- **operator** : value, count, min, max, mean, median, sum, standard_deviation, ou variance;
- **buffer** : Buffer para ser aplicado ao ponto da PCD. Parâmetro obrigatório somente para operador "count". Ver utilitário Buffer;
- **attribute** : String com o nome do atributo da PCD que deve ser utilizado para recuperar valores numéricos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador "count";
- **list_dcp** : [Opcional] Lista contendo a identificação das PCD´s. Ver utilitário "Regra de Influência". Não usar se operador zonal for "count". Se não utilizado nos demais operadores, **todas** as PCD´s ativas da série serão consideradas.
- **isActive** : [Opcional] utiliza-se uma expressão do tipo booleana, True ou False, que considera como parâmetro o botão Ativo, disponível em Registro de Dado Dinâmico. [True] botão Ativo desmarcado não são considerados os valores da pcd na análise. [False] considera os valores da pcd na análise, mesmo que o botão Ativo esteja desmarcado.

Segue a descrição de cada operador.

PCD: **Valor**

Retorna o valor da última leitura de cada PCD.

Sintaxe:

dcp.value("<attribute>")

Exemplo: x = dcp.value("pluvio")

PCD: **Contagem**

Retorna o número de PCD´s que influenciam cada PCD (como objeto monitorado).

Sintaxe:

dcp.count(<buffer>, [isActive])

Exemplo:
```
buf1 = Buffer()
x = dcp.count(buf1, True)
```

PCD: **Mínimo**

Retorna o menor valor de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.min("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.min("Pluvio", ids)
```

PCD: **Máximo**

Retorna o maior valor de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.max("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.max("Pluvio", ids)
```

PCD: **Média**

Retorna a média dos valores de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.mean("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.mean("Chuva", ids)
```

PCD: **Mediana**

Retorna a mediana dos valores de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.median("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.median("temperatura", ids)
```

PCD: **Soma**

Retorna a soma dos valores de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.sum("<attribute>", [<list_dcp>])

Exemplo:
```
ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.sum("temperatura", ids)
```

PCD: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.standard_deviation("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.standard_deviation("temperatura", ids)
```

PCD: **Variância**

Retorna a variância dos valores de um atributo de todas as PCD´s disponíveis na lista.

Sintaxe:

dcp.variance("<attribute>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.variance("temperatura", ids)
```

II.3- PCD histórico

Grupo de operadores que consideram as PCD´s individualmente e utilizam as últimas medidas obtidas por cada uma, no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

SINTAXE GERAL:

dcp.history.<operator>("<attribute>", "<time>", [<list_dcp>])

onde:

- **operator**: min, max, mean, sum, median, standard_deviation ou variance;
- **attribute**: String com o nome do atributo da PCD que deve ser utilizado para recuperar valores estatísticos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long);
- **time**: String com o intervalo de tempo para filtrar os valores de cada PCD no intervalo de tempo desejado. Ver utilitário **Unidades de tempo**;
- **list_dcp**: [Opcional] Lista contendo a identificação das PCD´s que influenciam a PCD em análise no momento, como se fosse um objeto monitorado. Ver utilitário "Regra de Influência". Se não utilizado **somente a PCD** corrente será considerada.

Segue a descrição de cada operador.

PCD histórico: **Mínimo**

Retorna o menor valor de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.min("<attribute>", "<time>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.min("Pluvio“, “1d”, ids)
```

PCD histórico: **Máximo**

Retorna o maior valor de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.max("<attribute>", "<time>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.max("Pluvio", "30h", ids)
```

PCD histórico: **Média**

Retorna a média dos valores de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.mean("<attribute>", "<time>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.mean("Chuva", "3d", ids)
```

PCD histórico: **Mediana**

Retorna a mediana dos valores de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.median("<attribute>", "<time>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.median("temperatura", "360min", ids)
```

PCD histórico: **Soma**

Retorna a soma dos valores de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.sum("<attribute>", "<time>", [<list_dcp>])

Exemplo: ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.history.sum("temperatura", "5h", ids)

PCD histórico: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.standard_deviation("<attribute>", "<time>", [<list_dcp>])

Exemplo: b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.standard_deviation("temperatura", "1w", ids)

PCD histórico: **Variância**

Retorna a variância dos valores de um atributo de cada PCD individualmente no intervalo de tempo definido entre a data/hora atual e o valor de tempo informado no passado.

Sintaxe:

dcp.history.variance("<attribute>", "<time>", [<list_dcp>])

Exemplo: b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.variance("temperatura", "24h", ids)

II.4- PCD histórico por intervalo

Grupo de operadores que consideram as PCD´s individualmente e utilizam as últimas medidas obtidas por cada uma, no intervalo de tempo definido entre dois valores de tempo informado no passado.

SINTAXE GERAL:

dcp.history.interval.<operator>("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

onde:

- **operator** : min, max, mean, sum, median, standard_deviation ou variance;
- **attribute**: String com o nome do atributo da PCD que deve ser utilizado para recuperar valores estatísticos. O atributo deve ser do tipo numérico (Ex. Integer, Float, Double, Long). Não usar para operador "count";
- **time_begin**: String inicial (mais antigo) do intervalo de tempo para filtrar as ocorrências;
- **time_end**: String final (mais recente) do intervalo de tempo para filtrar as ocorrências;
- **list_dcp**: [Opcional] Lista contendo a identificação das PCD´s que influenciam a PCD em análise no momento, como se fosse um objeto monitorado. Ver utilitário "Regra de Influência". Não usar se operador for "count". Se não utilizado nos demais operadores, **somente a PCD** corrente será considerada.

Segue a descrição de cada operador.

PCD histórico por intervalo: **Mínimo**

Retorna o menor valor de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.min(“<attribute>”, “<time_begin>”, “<time_end>”, [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out_union, 2, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.min("Pluvio", "2d", '1d', ids)
```

PCD histórico por intervalo: **Máximo**

Retorna o maior valor de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.max(“<attribute>”, “<time_begin>”, “<time_end>”, [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 400, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.max("Pluvio", "24h", "12h", ids)
```

PCD histórico por intervalo: **Média**

Retorna a média dos valores de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.mean("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Level,  10, "km", 5, "km")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.mean("Chuva", "24d", "12d", ids)
```

PCD histórico por intervalo: **Mediana**

Retorna a mediana dos valores de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.median("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.median("temperatura", "2d", "1d", ids)
```

PCD histórico por intervalo: **Soma**

Retorna a soma dos valores de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.sum("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

Exemplo:
```
ids = dcp.influence.by_attribute("Serra do Mar", [att1, att2, att2, att4])
x = dcp.history.interval.sum("temperatura", "2d", "1d", ids)
```

PCD histórico por intervalo: **Desvio Padrão**

Retorna o desvio padrão dos valores de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.standard_deviation("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.Out, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.standard_deviation("temperatura", "2d", "1d", ids)
```

PCD histórico por intervalo: **Variância**

Retorna a variância dos valores de um atributo de cada PCD individualmente no intervalo de tempo inicial e final informado no passado em função da data/hora atual.

Sintaxe:

dcp.history.interval.variance("<attribute>", "<time_begin>", "<time_end>", [<list_dcp>])

Exemplo:
```
b1 = Buffer(BufferType.In, 800, "m")
ids = dcp.influence.by_rule("Serra do Mar", b1)
x = dcp.history.interval.variance("temperatura", "2d", "1d", ids)
```